ARTICLE IN PRESS

COMPUTERS & SECURITY XXX (2012) 1–19

Available online at www.sciencedirect.com

SciVerse ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

Systematically breaking and fixing OpenID security: Formal analysis, semi-automated empirical evaluation, and practical countermeasures

San-Tsai Sun [a], *Kirstie Hawkey* [b,1], *Konstantin Beznosov* [a,*]

[a] *University of British Columbia, 2332 Main Mall, Vancouver BC, Canada V6T 1Z4*

[b] *Dalhousie University, 6050 University Avenue, Halifax NS, Canada B3H 4R2*

ARTICLE INFO

Article history:
Received 12 September 2011
Received in revised form
12 January 2012
Accepted 7 February 2012

Keywords:
OpenID
Web single sign-on
Authentication
Security protocol analysis
Empirical evaluation
Web application security

ABSTRACT

OpenID 2.0 is a user-centric Web single sign-on protocol with over one billion OpenID-enabled user accounts, and tens of thousands of supporting websites. While the security of the protocol is clearly critical, so far its security analysis has only been done in a partial and ad-hoc manner. This paper presents the results of a systematic analysis of the protocol using both formal model checking and an empirical evaluation of 132 popular websites that support OpenID. 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 (81%), or force a victim to sign in as the attacker in order to spoof the victim's personal information (77%). With additional capabilities (e.g., controlling a wireless access point), the adversary can impersonate the victim on 80% of the evaluated websites, and manipulate the victim's profile attributes by forging the extension parameters on 45% of those sites. 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.

© 2012 Elsevier Ltd. All rights reserved.

1. Introduction

Back in 2007, a typical web user had about 25 password-protected accounts, and entered approximately eight passwords per day (Florencio and Herley, 2007). In order to address this problem, a number of web single sign-on (WSSO) solutions were introduced. Their architectures separate the role of identity provider (IdP) from that of relying party (RP). An IdP maintains the identity information of the user and authenticates it, while an RP relies on the authenticated identity to make authorization decisions and customize the user experience. This separation enables the use of the same centrally-managed authenticated

* *Corresponding author*. 2332 Main Mall, Vancouver BC, Canada V6T 1Z4. Tel.: +1 604 822 9181.
E-mail addresses: santsais@ece.ubc.ca (S.-T. Sun), hawkey@cs.dal.ca (K. Hawkey), beznosov@ece.ubc.ca (K. Beznosov).
[1] Tel.: +1 902 494 1599.

0167-4048/$ – see front matter © 2012 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cose.2012.02.005

Please cite this article in press as: Sun S-T, et al., Systematically breaking and fixing OpenID security: Formal analysis, semi-automated empirical evaluation, and practical countermeasures, Computers & Security (2012), doi:10.1016/j.cose.2012.02.005

ARTICLE IN PRESS

credentials across multiple providers of web services, reducing the number of passwords a user has to deal with.

OpenID 2.0 (Recordon and Fitzpatrick, 2007) is an open and promising user-centric WSSO solution. According to the OpenID Foundation (2009), there are currently more than one billion OpenID-enabled user accounts provided by major service providers (e.g., Google, Yahoo, and AOL), and over 50,000 websites that accept OpenID for logins. In addition, the US Government has collaborated with the OpenID Foundation to support the Open Government Initiative's pilot adoption of OpenID technology.[3]

Even though OpenID is rapidly being adopted, its security has yet to be demonstrated. Besides the risks documented in the OpenID specification itself (e.g., phishing, IdP masquerade, replay, denial-of-service attacks), several security issues have been reported in the literature. Tsyrklevich and Tsyrklevich (2007) demonstrated how to insidiously log a user into her RP via a cross-site request forgery (we refer to this attack as a "*SSO CSRF*"), and how a fast network attacker could sniff an authentication response and reset the user's TCP connection to masquerade as that user (an "*impersonation*" attack). Barth et al. (2008) showed that the OpenID protocol is vulnerable to *session swapping*, which forces the user's browser to initialize a session authenticated as the attacker. Sovis et al. (2010) examined the OpenID extension framework and found that the extension parameters could be forged when the communication channel is not SSL-protected (a "*parameter forgery*" attack). However valuable these findings are, 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. This paper fills this gap.

We started our investigation of OpenID from the perspectives of the RP's business incentives (Sun et al., 2010a), and the user's usability and security concerns (Sun et al., 2010b, 2011a,b). In this work, we aimed at furthering the understand of the following questions:

- What are the root causes of the published vulnerabilities of OpenID? How could those systemic weaknesses be exploited by new attack vectors?
- How prevalent are those weakness in real-word implementations? If they are rare then these vulnerabilities would only be of academic interest.
- What are the limitations of existing countermeasures and recommendations? How can we improve them in order to address effectively the root causes of those weaknesses?

Finding new ways to attack the OpenID protocol is not the focus of this work. Instead, we believe that understanding the root weaknesses in the protocol is more important than finding new attacks. For instance, one of the attack traces from our formal model revealed that the RP may accept an authentication response from another session. This particular weakness can be exploited in many ways, such as session swapping, sniffing and resetting the user connection, DNS/ARP cache poisoning, or exploiting the exception handling vulnerability in the browser; but the root cause remains the same—the protocol lacks bindings among the protocol messages and the browser that issued those requests. Understanding the protocol's root weaknesses also leads us to identify new attack vectors, examine the limitations of existing defense mechanisms, and design effective countermeasures.

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 (Vigano, 2006). 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, which are further discussed in Section 2.2. 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.[4]

To summarize, this work makes the following contributions: (1) a formal specification and analysis of the OpenID protocol that identifies three weaknesses and correlates six types of possible attack vectors, (2) a semi-automatic OpenID vulnerability assessment tool, (3) an empirical evaluation of 132 OpenID-enabled websites, and (4) two proposed and evaluated countermeasures for the attacks that exploit the uncovered weaknesses in the protocol.

The rest of the paper is organized as follows: The next section discusses related work, and Section 3 provides an overview of our approach and the adversary model. The

[3] http://www.whitehouse.gov/open.

[4] http://sourceforge.net/projects/openidhack/.

Please cite this article in press as: Sun S-T, et al., Systematically breaking and fixing OpenID security: Formal analysis, semi-automated empirical evaluation, and practical countermeasures, Computers & Security (2012), doi:10.1016/j.cose.2012.02.005

ARTICLE IN PRESS

OpenID protocol and its formalization are presented in Section 4 and 5. In Section 6, the results of our evaluation of existing RP implementations are presented. We describe our proposed countermeasures in Section 7, and summarize the paper and outline future work in Section 8.

2. Background and related work

The Web as we know it today is site-centric, which results in users having multiple passwords and profiles. Web users face the burden of managing this increasing number of accounts and passwords, which leads to "password fatigue".[5] Aside from the burden on human memory, password fatigue may cause users to devise coping strategies that degrade the security of the protected information (Gaw and Felten, 2006; Florencio and Herley, 2007). In addition, the site-centric Web makes online profile management and personal content sharing difficult, as each user account is created and managed in a separate administrative domain (Sun et al., 2009).

To address the problems resulted from the site-centric Web, major content-hosting and service providers have implemented various protocols that allow other websites to accept user credentials from their domains (e.g., Microsoft Live ID (Oppliger, 2004), Yahoo BBAuth (Yahoo Inc, 2008), AOL OpenAuth (AOL LLC, 2008), and Facebook Connect (Facebook Inc, 2010)). However, not only these systems are proprietary, they are also centralized (i.e., identity information is maintained and controlled by a single administrative domain). In contrast, federated identity solutions such as Liberty Alliance Project[6] and Shibboleth[7] enable cross-domain single sign-on. Nevertheless, these solutions require pre-established agreements between organizations in the federation, making them hard to scale to the Web.

OpenID 2.0 (Recordon and Fitzpatrick, 2007) is a decentralized WSSO solution. One key scalability feature of OpenID is that it does not require any pre-registration of RP to IdP, and users are free to choose or even setup their own OpenID providers. Fig. 1 illustrates the following steps, which demonstrate a high level view of how the OpenID protocol works:

1. A user selects an IdP or enters her OpenID URL via a login form presented by an RP. We refer to this HTTP request as a "*Login Request*".
2. The RP fetches the document on the given OpenID URL to discover the IdP's endpoint, and then redirects the user to the IdP for authentication (an "authentication request" or "*Auth Request*," for short).
3. The user authenticates to the IdP by entering her user name and password, and then consents to the release of her profile information.
4. The IdP redirects the user back to the RP with the user's OpenID identifier and profile attributes, both of which are signed by the IdP (an "authentication response", "assertion", or "*Auth Response*").

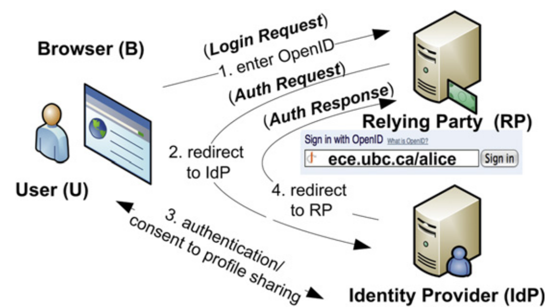

Fig. 1 – A high level view of how OpenID works.

2.1. *Known security issues*

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 and Tsyrklevich (2007) 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; (3) an IdP could track all the websites a user has logged into via the `return_to` parameter; (4) a network attacker could sniff the wire to intercept an authentication response to log into the RP as the victim user; and (5) a Web attacker could insidiously log a user into her RP via a cross-site forged login request.

Barth et al. (2008) introduce *login CSRF*, 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. (2010) 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. Wang et al. (2011) found some RP implementations do not check that 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,[8] and Delft and

[5] http://en.wikipedia.org/wiki/Password_fatigue.
[6] http://www.projectliberty.org/.
[7] http://shibboleth.internet2.edu.
[8] https://sites.google.com/site/openidreview/issues.

Please cite this article in press as: Sun S-T, et al., Systematically breaking and fixing OpenID security: Formal analysis, semi-automated empirical evaluation, and practical countermeasures, Computers & Security (2012), doi:10.1016/j.cose.2012.02.005

Oostdijk (2010) present the OpenID security issues found by others.

A formal OpenID model in AVISPA was presented by Lindholm (2009), 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 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.

2.2. *Existing defense techniques*

The OpenID Foundation's Security Best Practices page[9] suggests that "any URL that updates data on the user's behalf, or changes the state of the user's account MUST be CSRF protected." Cross-Site Request Forgery (CSRF) is a widely exploited web application vulnerability (OWASP, 2010). A CSRF attack tricks a user into loading a page that contains a malicious request that could disrupt the integrity of the victim's session with a website. The attack URL is usually embedded in an HTML construct (e.g., `<img src=http://bank.com/transfer?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 the 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.

Validating the HTTP Referer header to ensure the request in question was issued by an authorized source is a simple way to prevent SSO CSRF and session swapping attacks. However, due to privacy concerns, many web proxy servers suppress the Referer headers before sending out the requests (Barth et al., 2008), which makes this approach unreliable. Although requiring an IdP to prompt the user to login on every RP sign in attempt could be an effective way to prevent SSO CSRF attacks, this option requires changes to IdPs and, more importantly, would most likely turn users away from OpenID.

Including a hard-to-guess secret token in HTTP requests to ensure that the request in question was initiated by the website itself is another common CSRF defense mechanism. However, based on our formal analysis, the validation token must also be included in the Login and Auth Request, bound to the client, and signed by the IdP in order to ensure the origination of the authentication request. Thus, when using a validation token to protect CSRF, if the token is not embedded in the Auth Request, an attacker could simply launch an SSO CSRF attack through the Auth Request instead of from a token-protected Login Request. When the token is embedded in the Auth Request, but not bound to the client, an attacker could submit an RP login form from her browser with the victim's or an IdP's OpenID identifier to acquire an Auth Request that contains a valid token. Similarly, by using the attacker's identity and intercepting an authentication assertion, a session swapping attack could be launched even when the token is presented in Login and Auth Requests, but not bound to the client. Moreover, if the token is simply a hard-to-guess nonce, the parameters in the Auth Request could be manipulated during the transmission.

To prevent session swapping attacks, Barth et al. (2008) suggest that the RP should generate a fresh nonce at the start of the protocol, set a cookie with the value of the nonce, and then append the nonce to the `return_to` URL of an Auth Request. Upon receiving an Auth Response, the RP could check whether the nonce in the `return_to` URL matches the cookie sent from the browser to ensure that the received Auth Response is from the same browser as the request. However, this defense mechanism cannot prevent SSO CSRF via Login Request, or attacks that manipulate the authentication requests.

SSL provides end-to-end protection and is commonly suggested for mitigating attacks that manipulate network traffic. However, SSL cannot prevent attacks launched from a browser such as SSO CSRF and session swapping attacks. In addition, an SSL server requires an RP to maintain a valid certificate (e.g., setup, renew, key management), needs to run on its own IP address[10] (i.e., no virtual hosting), imposes performance overhead, and introduces undesired side-effects. SSL makes web contents non-cacheable for the proxies and content delivery networks, and prohibits progressive content rendering as web contents in a HTTPS page cannot be displayed by the browser until they are fully loaded and verified. Additionally, to avoid browser warnings about mixed secure (HTTPS) and insecure (HTTP) content, all related resources included in an SSL-protected page must be delivered under a computationally intensive SSL. This introduces an additional computation overhead and non-cacheable latency for static graphical content that typically requires no protection (e.g., images, Flashes), and it might not be practical if some content is from external websites. Moreover, encrypted content cannot be evaluated, scanned, or routed based on content by intermediate security devices deployed by the website. Due to these unwanted complications, many websites use SSL only for login pages or certain services (e.g., credit card processing), but do not use SSL for the rest of the site after authentication (Adida, 2008; Singh et al., 2011). Nevertheless, the lack of protection in the subsequent communication allows network attackers to simply sniff the session cookies to hijack victims' sessions, even though the login process is secured (Graham, 2007; OWASP, 2009).

At the time of this writing, OpenID Foundation is drafting the next version of OpenID, named "OpenID Connect" (Sakimura et al., 2011), which is completely different than the

[9] http://wiki.openid.net/w/page/12995200/.

[10] Server Name Indication (SNI), a TLS extension (Blake-Wilson et al., 2003), supports virtual hosts for SSL, but not fully supported by all browsers and servers yet.

Please cite this article in press as: Sun S-T, et al., Systematically breaking and fixing OpenID security: Formal analysis, semi-automated empirical evaluation, and practical countermeasures, Computers & Security (2012), doi:10.1016/j.cose.2012.02.005

OpenID 2.0 protocol. OpenID Connect uses OAuth 2.0 (Hammer-Lahav et al., 2011) as the basic access authorization protocol and adds identity and interoperability features (e.g., an ID token that contains claims about the authentication event, endpoints for retrieving user profile attributes and session management, dynamic IdP discovery and RP registration) so that a single implementation of an RP website could virtually interact with all OpenID Connect IdPs without a tailored configuration, registration, or implementation. While the adoption of OpenID Connection might be seen in the near future, the security analysis and evaluation is a research topic that requires further investigation.

3. Approach and adversary model

Fig. 3 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 (Caleiro et al., 2005; LSV, 2003; Denker and Millen, 2000; Lowe, 1998). 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 (Vigano, 2006). AVISPA is a security protocol verification tool that has been widely employed to validate authentication and key exchange protocols.[11] 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,[12] and 29 websites from the Google Top 1000 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.

3.1. *Adversary model*

In this paper, we assume that both the RP and IdP are trustworthy 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 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, two different adversary types are considered, which vary on their attack capabilities:

- **A Web attacker** can post comments that include static content (e.g., images, style sheet) 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 have custom HTTP headers, such as cookies.
- **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 pharming" (Stamm et al., 2007) attacks to alter the DNS server settings on the victim's home broadband router.

4. The OpenID protocol

OpenID uses a URL or XRI (OASIS, 2008) 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 paper, the notation described in Table 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

[11] See http://www.avispa-project.org for the library of examined protocols and related papers.

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

ARTICLE IN PRESS

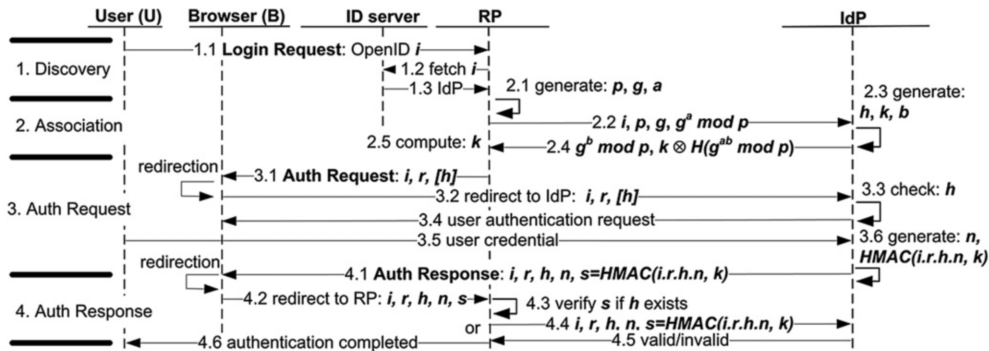

Fig. 2 – The OpenID protocol sequence diagram.

data in the protocol. As illustrated in Fig. 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 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 $g^a \bmod p$ 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 $g^b \bmod p$, h, and an encrypted session key $k_{enc} = (k \otimes H(g^{ab} \text{mod} p))$ to **RP**.
2.5. **RP** computes $k = H(g^{ab} \text{mod} p) \otimes k_{enc}$ and then stores the tuple (h, k, i).

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.
3.6. If the user credentials are correct, 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' = 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

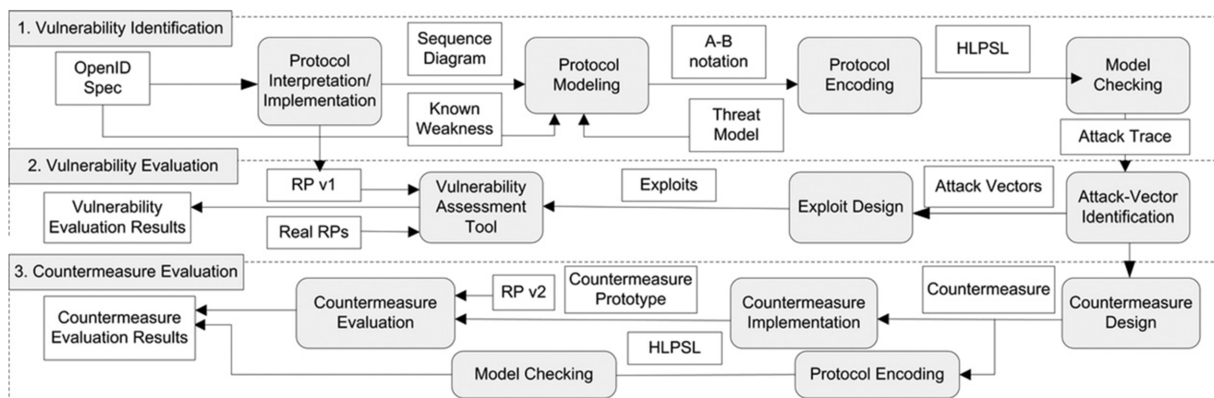

Fig. 3 – Overall approach.

Please cite this article in press as: Sun S-T, et al., Systematically breaking and fixing OpenID security: Formal analysis, semi-automated empirical evaluation, and practical countermeasures, Computers & Security (2012), doi:10.1016/j.cose.2012.02.005

ARTICLE IN PRESS

Table 1 – Notation.

Notation	Description	Notation	Description
i	OpenID identifier	*h*	Session handle
k	Session key	*n*	Nonce
r	RP return url	*x.y*	Concatenation of *x* and *y*
x⊗*y*	XOR of *x* and *y*	*E*(*x*, *k*)	Encrypt *x* with key *k*
H(*x*)	Hash function on *x*	HMAC(*x*, *k*)	**HMAC** on *x* with key *k*
U	User	**B**	Browser
RP	Relying party	**IdP**	Identity Provider

perform local validation on *s* only if it has established a shared session key *k* with **IdP** in the 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** allows **U** to sign in using *i* as her identifier.

5. 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 endpoint URL of the IdP based on a given OpenID identifier. Thus, the discovery steps (Steps 1.2 and 1.3 In Fig. 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 2.1–2.5) 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 → B : E(n_a, k), A sends B a nonce n_a encrypted with a shared key k
2. B → A : E(n_b, k), B sends A another nonce n_b also encrypted with k
3. A → 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.

5.1. *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 Fig. 4 is modeled. Each step in the A-B notation is annotated with the corresponding steps from the protocol sequence described in Section 4. Steps 3 to 5 use the aforementioned authentication protocol to authenticate **UB** to **IdP**.

5.2. *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

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

Fig. 4 – The Alice-Bob formalization of the OpenID protocol. The corresponding steps from the sequence diagram is denoted in the end of each step.

Please cite this article in press as: Sun S-T, et al., Systematically breaking and fixing OpenID security: Formal analysis, semi-automated empirical evaluation, and practical countermeasures, Computers & Security (2012), doi:10.1016/j.cose.2012.02.005

ARTICLE IN PRESS

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 Fig. 5, and the source code and attack traces are listed in Appendix A. 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 Fig. 5, the message sequences between each role have a one-to-one mapping to the A-B notation defined in Fig. 4. 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 (Dolev and Yao, 1983). 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 Fig. 4. 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

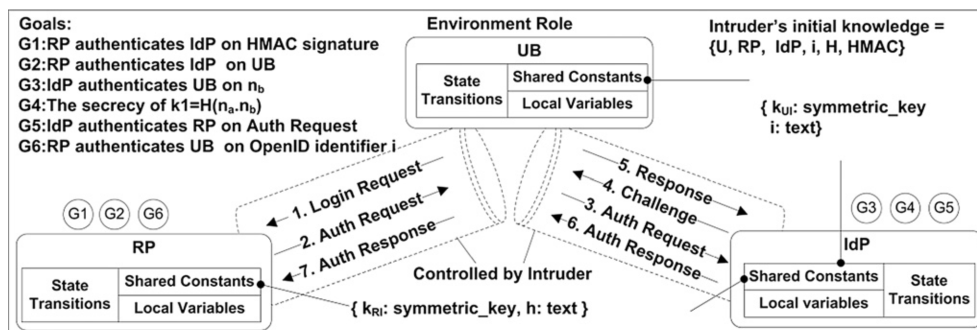

Fig. 5 – The conceptual model of the HLPSL formalization.

Please cite this article in press as: Sun S-T, et al., Systematically breaking and fixing OpenID security: Formal analysis, semi-automated empirical evaluation, and practical countermeasures, Computers & Security (2012), doi:10.1016/j.cose.2012.02.005

ARTICLE IN PRESS

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 victims 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.

6. 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.

6.1. *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 (Skybound Software, 2010). 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 been forced to login successfully.

A2: SSO CSRF via Login Request through POST method (exploits G6): The evaluation procedures for this attack are 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

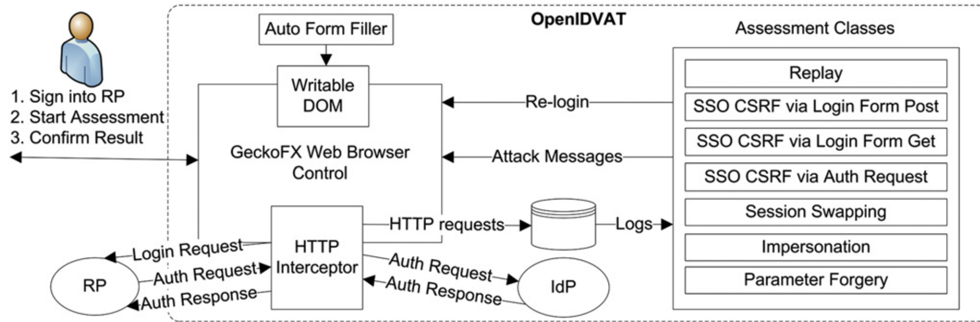

Fig. 6 – Main components of OpenIDVAT.

extensions. The extension parameters contains profile information of the user such as email and date of birth.

2. Re-initiate a login process again. This time, strip out all related extension parameters in the Auth Request, and append forged extension parameters to the Auth Response before sending it to the RP website.

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 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).

6.2. *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 (Skybound Software, 2010) 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.

Fig. 6 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 OpenIDVAT 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.

6.3. *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,[13] Gigya[14]) 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 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

[13] http://www.janrain.com/products/rpx.
[14] http://www.gigya.com/.

Please cite this article in press as: Sun S-T, et al., Systematically breaking and fixing OpenID security: Formal analysis, semi-automated empirical evaluation, and practical countermeasures, Computers & Security (2012), doi:10.1016/j.cose.2012.02.005

ARTICLE IN PRESS

Table 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.

No. of RPs	D1	D2	Total
	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%
POST	73%	14%	67%
GET	44%	9%	41%
Auth Req	69%	13%	64%
Session Swap	76%	83%	77%
Impersonation	88%	55%	80%
Replay	10%	21%	12%
Support Extension	N = 76	N = 26	N = 102
Parameter Forgery	54%	7%	45%

replay attacks (within 5–10 min) in addition to session swapping and SSO CSRF attacks.

As shown in Table 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.

7. 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**. Fig. 7 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.

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.

7.1. 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 session identifier from the session cookie and k_{RP} is an application or session-

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

Fig. 7 – The revised OpenID protocol in Alice-Bob notation.The changes are shown in boldface.

Please cite this article in press as: Sun S-T, et al., Systematically breaking and fixing OpenID security: Formal analysis, semi-automated empirical evaluation, and practical countermeasures, Computers & Security (2012), doi:10.1016/j.cose.2012.02.005

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 $t1' = \text{HMAC}(sid, k_{RP})$ and checks whether $t1' = t1$ from the request. If it is, then RP initiates an Auth Request with parameter $t2 = \text{HMAC}(sid.idp.i.h.r,\ k_{RP})$ 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 $t2' = \text{HMAC}(sid.idp.i.h.r,\ k_{RP})$, 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, and prevents an attacker who learned the token from inferring with the user's 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 identifier 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[15] 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 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.

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

7.2. *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 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 Fig. 8):

Please cite this article in press as: Sun S-T, et al., Systematically breaking and fixing OpenID security: Formal analysis, semi-automated empirical evaluation, and practical countermeasures, Computers & Security (2012), doi:10.1016/j.cose.2012.02.005

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 $t3 = \text{HMAC}(idp.i.h.r,\ k_C)$ 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 $t3' = \text{HMAC}(idp.i.h.r,\ k_S)$ 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 Fig. 9, 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., $k_C = g^{ac} \bmod p$ and $k_S = g^{bc} \bmod p$) 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., $\text{HMAC}(m,\ k_C) \mathrel{!=} \text{HMAC}(m,\ k_S)$). In our defense mechanism, the client appends a validation token $t3 = \text{HMAC}(idp.i.h.r, k_C)$ 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 3). To pass the token validation performed by **RP** in Step 3, the attacker *must* replace $t3$ with $t3' = \text{HMAC}(idp.i.h.r,\ k_S)$ 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 $t3' = \text{HMAC}(idp.i.h.r, k_S)$ 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, it 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

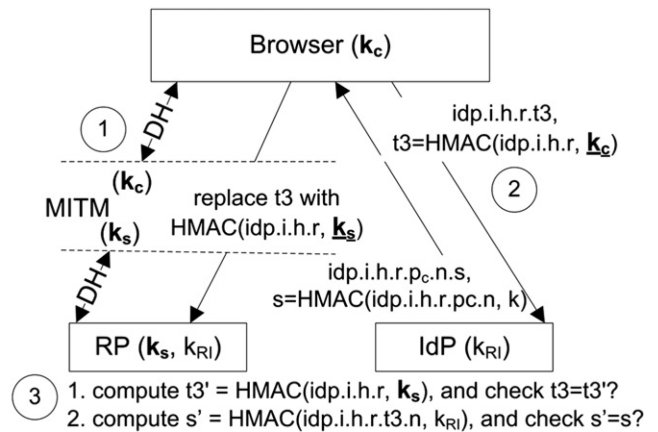

Fig. 9 – The MITM defense mechanism with the presence of an MITM attacker between the browser and the RP server The OpenID authentication protocol will fail if the MITM attacker attempts to interfere the DH key exchange. If the Auth Response is successfully validated, then the DH key shared by the browser and the RP is unknown to the attacker.

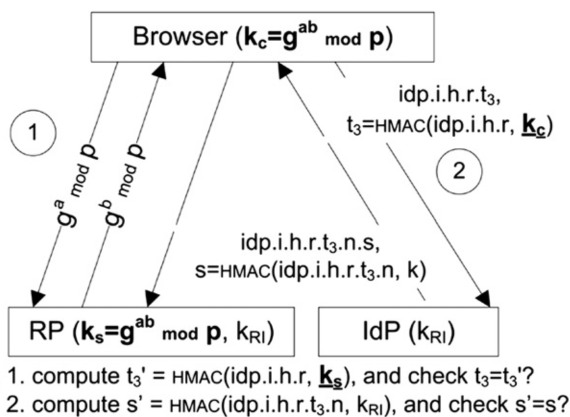

Fig. 8 – The MITM defense mechanism establishes a DH session key (g^{ab} mod p) between the browser and the RP server during the OpenID authentication process. Here, g is the DH generator, p is the modulus, and a and b are random DH private keys for the browser and the RP server respectively.

ARTICLE IN PRESS

SessionLock (Adida, 2008) and other web-based APIs, such as the Google and Facebook Platform APIs, but does not require SSL support from the RP websites.

7.3. Reference implementation

To evaluate the proposed defense mechanisms, we developed a reference implementation. We first used the OpenID4Java (Bufu, 2009) library to augment OpenID support in an open-source J2EE web application,[16] and then implemented the countermeasures on the web application.

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 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).

7.4. Limitations

Our Web attacker defense mechanism could be easily implemented by RPs, 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 paper.

8. Conclusion

Similar to the way credit cards reduce the friction of paying for goods and services, OpenID systems are intended to reduce the friction of using the Web. While OpenID is rapidly gaining adoption, for RPs and IdPs (and possibly users) to entrust the exchange of sensitive information over the OpenID protocol, they need to have confidence in its security properties.

In this work, we conducted a formal model checking analysis of the OpenID 2.0 protocol, and an empirical evaluation of 132 OpenID-enabled websites. Our 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 issued those requests. The results of our empirical evaluation show that the uncovered vulnerabilities are prevalent among the real-word RP implementations, including popular RP websites listed on the Google Top 1000 Websites. In addition, we found existing countermeasures are incomplete (e.g., fail to protect both integrity and authenticity), or have been implemented incorrectly (e.g., neglect GET method or Auth Request when implementing SSO CSRF protection). We also found that OpenID proxy services provide an integrated interface for RPs to interact with various WSSO systems, but many RPs failed to protect the returned access tokens. Furthermore, our evaluation found that only 19% of RP websites in our evaluation employed SSL, and 84% of them are vulnerable to session hijacking via an eavesdropped session cookie after login. We believe that the reasons behind this practice deserve further investigation.

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. Furthermore, the vulnerabilities discussed in this paper may be generalizable to other WSSO protocols. In future research, we plan to employ our analysis methodology for investigating the security of other WSSO systems.

Acknowledgments

We thank members of the Laboratory for Education and Research in Secure Systems Engineering (LERSSE) who provided valuable feedback on the earlier drafts of this paper. Limin Jia has provided helpful feedback on the formal analysis of OpenID, as well as description of attacks and defenses. This research has been partially supported by the Canadian NSERC ISSNet Internetworked Systems Security Network Program.

[16] BookStore from http://gotocode.com.

Please cite this article in press as: Sun S-T, et al., Systematically breaking and fixing OpenID security: Formal analysis, semi-automated empirical evaluation, and practical countermeasures, Computers & Security (2012), doi:10.1016/j.cose.2012.02.005

Appendix A. OpenID HLPSL code

```
%% 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)
```

ARTICLE IN PRESS


```
  % 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
```

Please cite this article in press as: Sun S-T, et al., Systematically breaking and fixing OpenID security: Formal analysis, semi-automated empirical evaluation, and practical countermeasures, Computers & Security (2012), doi:10.1016/j.cose.2012.02.005

ARTICLE IN PRESS


```
  % 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()
```

Please cite this article in press as: Sun S-T, et al., Systematically breaking and fixing OpenID security: Formal analysis, semi-automated empirical evaluation, and practical countermeasures, Computers & Security (2012), doi:10.1016/j.cose.2012.02.005

REFERENCES

Adida B. Sessionlock: securing web sessions against eavesdropping. In: Proceeding of the 17th International Conference on World Wide Web (WWW'08). New York, NY: ACM; 2008. p. 517–24.

AOL LLC. AOL open authentication API, http://dev.aol.com/api/openauth; January 2008 [Online; accessed 23.08.11].

Barth A, Jackson C, Mitchell JC. Robust defenses for cross-site request forgery. In: Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS'08). New York, NY: ACM; 2008. p. 75–88.

Blake-Wilson S, Nystrom M, Hopwood D, Mikkelsen J, Wright T. Transport layer security extensions, http://www.ietf.org/rfc/rfc3546.txt; June 2003 [Online; accessed 23.08.11].

Bufu J. OpenID4Java, http://code.google.com/p/openid4java/; 2009 [Online; accessed 23.08.11].

Caleiro C, Vigan L, Basin D. Deconstructing Alice and Bob. Electronic Notes in Theoretical Computer Science 2005;135(1):3–22. URL: http://www.sciencedirect.com/science/article/pii/S1571066105050498.

Delft B, Oostdijk M. 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.

Denker G, Millen J. Capsl integrated protocol environment. In: Proceedings of DARPA Information Survivability Conference and Exposition, vol. 1; 2000. p. 207–21.

Dolev D, Yao A. On the security of public key protocols. IEEE Transactions on Information Theory Mar. 1983;29(2):198–208.

Facebook, Inc. Facebook for websites: authentication, http://developers.facebook.com/docs/guides/web/; 2010 [Online; accessed 23.08.11].

Florencio D, Herley C. A large-scale study of web password habits. In: Proceedings of the 16th International Conference on World Wide Web (WWW'07). New York, NY, USA: ACM; 2007. p. 657–66.

Gaw S, Felten EW. Password management strategies for online accounts. In: Proceedings of the Second Symposium on Usable Privacy and Security (SOUPS'06); 2006. p. 44–55.

Graham R. Sidejacking with hamster, http://erratasec.blogspot.com/2007/08/sidejacking-with-hamster_05.html; August 2007 [Online; accessed 23.08.11].

Hammer-Lahav E, Recordon D, Hardt D. The OAuth 2.0 authorization protocol, http://tools.ietf.org/html/draft-ietf-oauth-v2-22; September 2011 [Online; accessed 09.12.11].

Lindholm A. Security evaluation of the OpenID protocol. Master of Science Thesis from Royal Institute of Technology; 2009.

Lowe G. Casper: a compiler for the analysis of security protocols. Journal of Computer Security 1998;6(1):53–84. URL: http://iospress.metapress.com/content/ADNXU4KPRPL21RC9.

LSV. Security protocols open repository (SPORE), www.lsv.ens-cachan.fr/spore/; 2003 [Online; accessed 12.12.11].

OASIS. OASIS extensible resource identifier, www.oasis-open.org/committees/xri/; 2008 [Online; accessed 23.12.11].

OpenID Foundation. Promotes, protects and nurtures the OpenID community and technologies, http://openid.net/foundation/; 2009 [Online; accessed 23.08.11].

Oppliger R. Microsoft.NET Passport and identity management. Information Security Technical Report 2004;9(1):26–34.

OWASP. Session hijacking attack, https://www.owasp.org/index.php/Session_hijacking_attack; 2009 [Online; accessed 23.08.11].

OWASP. Open web application security project (OWASP) top ten project, http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project; 2010 [Online; accessed 23.08.11].

Recordon D, Fitzpatrick B. OpenID authentication 2.0, http://openid.net/specs/openid-authentication-2_0.html; December 2007 [Online; accessed 23.08.11].

Sakimura N, Bradley J, de Medeiros B, Jones MB, Jay E. Openid connect standard 1.0-draft 07, http://openid.net/specs/openid-connect-standard-1_0.html; 2011 [Online; accessed 03.01.12].

Singh K, Wang H, Moshchuk A, Jackson C, Lee W. HTTPi for practical end-to-end web content integrity. In: Microsoft technical report; May 2011 [Online; accessed 23.08.11].

Skybound Software. GeckoFX: an open-source component for embedding Firefox in.NET applications, http://www.geckofx.org/; 2010 [Online; accessed 23.08.11].

Sovis P, Kohlar F, Schwenk J. Security analysis of OpenID. In: Proceedings of the Securing Electronic Business Processes–Highlights of the Information Security Solutions Europe 2010 Conference; October 2010.

Stamm S, Ramzan Z, Jakobsson M. Drive-by pharming. In: Information and communications security. Lecture notes in Computer Science, vol. 4861. Springer Berlin/Heidelberg; 2007. p. 495–506.

Sun S-T, Hawkey K, Beznosov K. Secure Web 2.0 content sharing beyond walled gardens. In: Proceedings of the 25th Annual Computer Security Applications Conference (ACSAC'09). IEEE Press; December 7–11 2009. p. 409–18.

Sun S-T, Boshmaf Y, Hawkey K, Beznosov K. A billion keys, but few locks: the crisis of Web single sign-on. In: Proceedings of the New Security Paradigms Workshop (NSPW'10); 2010a. p. 61–72.

Sun S-T, Hawkey K, Beznosov K. 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). New York, NY, USA: ACM; 2010b. p. 49–58.

Sun S-T, Pospisil E, Muslukhov I, Dindar N, Hawkey K, Beznosov K. 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); 2011a. New York, NY, USA.

Sun S-T, Pospisil E, Muslukhov I, Dindar N, Hawkey K, Beznosov K. What makes users refuse web single sign-on? An empirical investigation of OpenID. In: Proceedings of Symposium on Usable Privacy and Security (SOUPS'11); 2011b.

Tsyrklevich E, Tsyrklevich V. Single sign-on for the Internet: a security story. In: Proceedings of the BlackHat'07; July 2007.

Vigano L. Automated security protocol analysis with the AVISPA tool. Electronic Notes in Theoretical Computer Science 2006;155:61–86.

Wang R, Chen S, Wang X. Attribute exchange security alert, http://openid.net/2011/05/05/attribute-exchange-security-alert/; May 2011 [Online; accessed 23.08.11].

Yahoo Inc. Browser-based authentication (BBAuth), http://developer.yahoo.com/auth/; December 2008 [Online; accessed 23.08.11].

San-Tsai Sun is a PhD student in the Electrical and Computer Engineering department (ECE) at the University of British Columbia (UBC). He works in the Laboratory for Education and Research in Secure Systems Engineering (LERSSE) under the supervision of Professor Konstantin Beznosov. His research interests include Web application security, Web 2.0 security and privacy, and distributed access control architecture. His PhD dissertation focuses on improving the usability and security of access control system in Web related systems.

Kirstie Hawkey is an Assistant Professor in the Faculty of Computer Science at Dalhousie University. Her research background is in human-computer interaction and her current interests include usable privacy and security and personal information management, particularly within the context of group work.

Please cite this article in press as: Sun S-T, et al., Systematically breaking and fixing OpenID security: Formal analysis, semi-automated empirical evaluation, and practical countermeasures, Computers & Security (2012), doi:10.1016/j.cose.2012.02.005

ARTICLE IN PRESS

Konstantin Beznosov is an Associate Professor at the Department of Electrical and Computer Engineering, University of British Columbia, where he directs the Laboratory for Education and Research in Secure Systems Engineering. His research interests are usable security, distributed systems security, secure software engineering, and access control. He has served on program committees and/or helped to organize SOUPS, CCS, NSPW, NDSS, ACSAC, SACMAT, CHIMIT. Prof. Beznosov is an associate editor of ACM Transactions on Information and System Security (TISSEC) and International Journal of Secure Software Engineering (IJSSE).

Please cite this article in press as: Sun S-T, et al., Systematically breaking and fixing OpenID security: Formal analysis, semi-automated empirical evaluation, and practical countermeasures, Computers & Security (2012), doi:10.1016/j.cose.2012.02.005