SUPPORTING END-TO-END SECURITY ACROSS PROXIES WITH MULTIPLE-CHANNEL SSL

Yong Song, Victor C.M. Leung, Konstantin Beznosov
Department of Electrical and Computer Engineering, University of British Columbia, 2356 Main Mall, Vancouver, BC V6T 1Z4, Canada

Abstract: Secure Socket Layer (SSL) has functional limitations that prevent end-to-end security in the presence of untrusted intermediary application proxies used by clients to communicate with servers. This paper introduces Multiple-Channel SSL (MC-SSL), an extension of SSL, and describes and analyzes the design of MC-SSL proxy channel protocol that enables the support for end-to-end security of client-server communications in the presence of application proxies. MC-SSL is able to securely negotiate multiple virtual channels with different security characteristics including application proxy and cipher suite.

Keywords: Network security; mobility; authentication; integrity; privacy.

1. INTRODUCTION

In this paper, we propose multiple-channel SSL (MC-SSL), a new protocol based on TLS/SSL [1] (for brevity, referred in this paper as SSL). It enjoys several advantages over current SSL variants. First, MC-SSL can significantly strengthen end-to-end security when application proxies or gateways are involved. Second, MC-SSL supports multiple cipher suites in the same connection so that different degrees of protection are available for client-server communications. Third, MC-SSL can flexibly satisfy various security requirements for content delivery because new factors, such as security policies, device capabilities, and security attributes of contents, are introduced into the security model.

MC-SSL is especially helpful for resource-constrained devices such as PDAs and cellular phones because they tend to need application proxies for functions such as content transformation or virus scanning. Such devices can

also better utilize battery and other resources by selectively choosing the degree of cryptographic protection for data transmission. Moreover, because MC-SSL is a general protocol that can negotiate multiple security channels, it can flexibly meet the requirements from various terminals, servers, applications, and users. In this paper we mainly describe those parts of MC-SSL that are relevant to the support of end-to-end security in the presence of application proxies. For example, we do not present the concrete protocol related to multiple cipher suites. Instead, we focus on the high-level model issues and the detailed design of the proxy channel protocol, which is considered to be a prominent feature of MC-SSL. The MC-SSL architecture supports two types of channels between a client and a server: end-to-end channels and proxy channels. The proxy channel protocol is able to securely set up and use a proxy channel, and hence greatly improves the end-to-end security when a proxy is needed.

The remainder of the paper is organized as follows. Section 2 analyzes the functional limitations of SSL that triggered our work on MC-SSL. Section 3 presents the high-level description of MC-SSL. Section 4 discusses related work. Section 5 presents the proxy protocol, which is the focus of this paper. Section 6 discusses advantages and disadvantages of the proxy protocol. Section 7 concludes the paper, and describes future work.

2. PROBLEM MOTIVATION

Although it is currently a *de facto* security protocol at transport layer for Internet applications, SSL has several functional limitations. First, while SSL can provide an application with a secure point-to-point connection, it does not have facilities to securely deal with application proxies: if a proxy **P** is involved between a client **C** and a server **S**, **C** will normally set up a SSL connection with **P**, and then **P** will act as the delegate of **C** and set up another SSL connection with **S**. These kinds of proxies are needed for functional or performance reasons such as virus scanning, content transforming/filtering, or compression. For instance, in order to enable wireless terminals to access the Internet, WAP gateway architecture adopted the chain proxy model although the connection between **C** and **P** in the architecture is replaced with WTLS, a variant of TLS protocol. The SSL chain proxy model shown in the lower part of Figure 1, in which **P** can read and modify sensitive data at will, requires unconditional trust in **P** at least from one side, **S**'s or **C**'s. This can be satisfied only if **P** is administrated by the organization or individual that administrates **S** or **C** as well. In that case, the trust model is actually equivalent to the end-to-end trust model with two entities, and **P** can be merged with **S** or **C**. In other cases, **S** or **C** must take

risks of information leakage and tampering at **P** because **P** is a third party in the trust model. Since communication through a third party application proxy requires **S** or/and **C** to trust **P** unconditionally with all the information passed through it, we consider the dilemma between passing all the data through **P** and not using a proxy at all as a limitation.

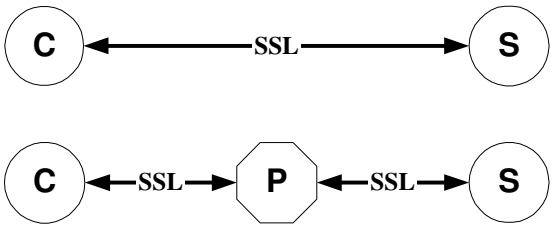

Figure 1. End-to-end model and chain proxy model

The second functional limitation of SSL is that it employs only one cipher suite at any given time. Although SSL allows re-negotiating the cipher suite of a connection, frequent re-negotiations are not practical because it is inefficient to change cipher suites back and forth using relatively expensive handshake protocol. Accordingly, lots of data is overly protected. For instance, a handheld user checks his corporate email inbox all day. He wants encryption for the id/password of his email account but he does not need strict confidentiality for his emails. It might be good enough for him to have good encryption for the id/password transmission, but no encryption for email contents. This way, the battery power will not be drained. However, the SSL architecture gives all or nothing dilemma. As another example, consider a handheld user who accesses the application server of her stock brokerage. She wants the stock prices to be accurate, but does not require them to be encrypted since they are publicly available; however, she requires the best security protection when she is temporarily transmitting id/password or doing transactions. In this case, using two different cipher suites are better than using only the strongest cipher suite because the battery power is greatly reserved. To summarize, the requirement for communication security does not entail the strongest cipher. Security is tightly related to other requirements. As pointed out by Abadi and Needham, encryption is not wholly cheap, and not asking precisely why it is being done can lead to redundancy [2]. The SSL's support for one cipher suite at a time combined with the relatively high cost of changing suites just in time makes it difficult for applications to optimize the strength of data protection according to the changes in the sensitivity of the data in the channel.

Further, to allow **S** to take various requirements into account and optimize a secure channel using power and other constraints, **C** may want to send **S** its terminal capabilities and the security policy configured for a particular application or server. For example, **C** could need to define whether proxies are allowed to be used for passing data with sensitivity below a certain level, and what types of data need proxies. Lack of negotiation support for proxies and multiple cipher suites is the third functional limitation of SSL, which directly results from the first and the second limitations. These functional limitations of SSL form an obvious gap between SSL and the requirements of real world applications and devices. When security-sensitive mobile applications become more popular, the gap will become more apparent.

3. HIGH LEVEL DESCRIPTION OF MC-SSL

Three key features of MC-SSL help us to address the above limitations of SSL. Going in reverse order, MC-SSL supports channel negotiation according to the parties' security policies, device capabilities, and security attributes of data.

To address second limitation (only one cipher suite at a time), MC-SSL provides a set of customizable secure channels in order to meet the practical requirements of different clients, servers, and applications. MC-SSL uses a multiple-channel model, in which each channel can possess its own characteristics including cipher suite and data flow direction.

To address the first limitation (the dilemma between an unconditionally trusted proxy and no proxy at all), we introduce a special type of channel: proxy channel, which enables MC-SSL to support partially trusted proxies, the focus of this paper. Figure 2 shows the conceptual proxy model of MC-SSL, in which three SSL connections form a triangle. **C**-**P**-**S** is a proxy channel, and **C**-**S** is an end-to-end channel. In this model, **C**-**P**-**S** is no longer an independent proxy channel that is shown in Figure 1. Instead, it relies on the **C**-**S** channel to control channel negotiation and application data delivery. Besides, **C** and **S** can deliberately choose **C**-**S** or **C**-**P**-**S** to deliver data according to the requirements for proxy and the sensitivity of data. As a result, sensitive data, such as id/password or credit card number, do not have to be exposed to **P**. The protocol for proxy channels is described in Section 5 and discussed in Section 6.

A MC-SSL session may negotiate zero or more proxy channels. Each of them and the corresponding end-to-end channel form a triangular relationship with the proxy as the third vertex. Theoretically, the maximum number of proxy channels solely depends on the available resources at **C** and

S. However, a proxy channel currently supports only one proxy. We expect that for most practical client-server applications, one-hop proxy channels should suffice because multiple proxies can be transformed into a "proxy cluster" in which one proxy acts as the cluster head. On the other hand, we are planning to extend MC-SSL to support multi-hop proxy channels so as to further generalize the model of MC-SSL.

In SSL, a cipher suite consists of a key exchange algorithm, a cipher, and a hash algorithm, e.g., {RSA, 3DES_EDE_CBC/168, SHA-1}. The hash algorithm is used to compute Message Authentication Code (MAC). In MC-SSL, a cipher suite consists of only two elements: a cipher for data encryption/decryption, and a hash algorithm for MAC. We can define it as a structure as follows:

$$\{\textit{cipher and key size, hash algorithm for MAC}\} \quad (1)$$

A MC-SSL connection can have multiple cipher suites. We can characterize a point-to-point connection as follows: {*point 1, point 2, key exchange algorithm,* {*cipher suite 1, cipher suite 2,* …}}, where each cipher suite forms a channel. Every MC-SSL connection must first negotiate a cipher suite strong enough to form the primary (or backbone) channel, which is employed to set up and control other channels, named as secondary channels. Figure 3 illustrates a sample connection between **A** and **B**, which is characterized by {**A**, **B**, RSA, {CS1, CS2, CS3, CS4}}, where RSA is the key exchange algorithm, and CS1 to CS4 are four different cipher suites. The primary channel is channel 1, for which CS1 is used.

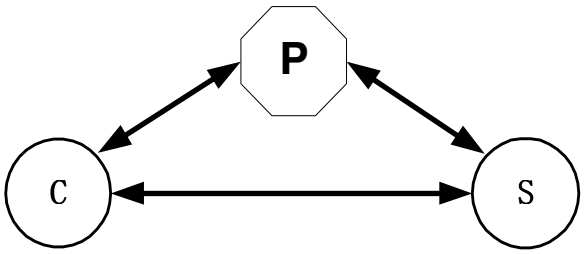

Figure 2. Triangular proxy model of MC-SSL

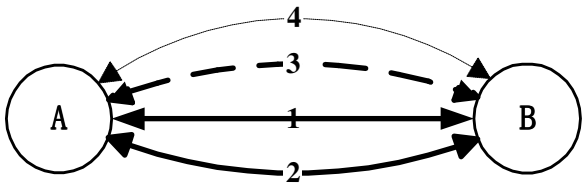

Figure 3. Multiple cipher suites inside a point-to-point connection

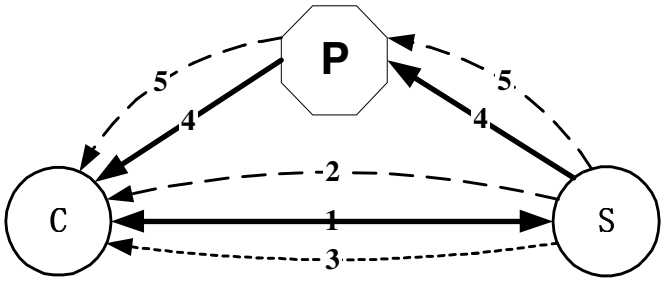

Figure 4. Multiple-channel model of MC-SSL

A combination of the proxy model and the multiple cipher suites produces the multiple-channel model shown in Figure 4. In MC-SSL, a channel can be defined as a virtual communication "pipe" with or without intermediate application proxies. Two MC-SSL endpoints communicate with each other through the pipe using a cipher suite. In addition, a channel can be either duplex, or simplex with a flow direction. We can characterize a MC-SSL channel with a set of attributes:

{*channel id, endpoint1, endpoint2, proxy, direction, cipher suite*} (2)

Channel id is the identifier of a channel in a MC-SSL session context. Endpoint1 and endpoint2 are either DNS names or IP addresses of corresponding machines. Proxy attribute is null if a channel is an end-to-end channel; otherwise, it is the DNS name or IP address of the proxy in a proxy channel. Direction can be one of the following values: D, C, and S. D indicate a duplex channel; C or S indicates a simplex channel pointing to **C** or **S**, respectively. Cipher suite has been defined in expression (1). Figure 4 shows a sample MC-SSL session having five channels. Channel 1 and 4 are primary (or backbone) channels, and channel 2, 3, and 5 are secondary channels. In addition, only channel 1 is a duplex channel for application data; others are simplex channels from **S** to **C**. Typically, **C** can use channel

1 to send encrypted requests to **S**, and **S** can choose one of the five channels to send back the responses according to the contents. In addition, these channels are negotiated in the following order: channel 1 is the first; channel 2 and 3 are negotiated through channel 1; channel 4 is the first proxy channel, which is also negotiated through channel 1; channel 5 is negotiated through channel 4. Except for channel 1, other channels can be set up at any time. In Section 5 we present the protocol to negotiate and use primary channels, especially primary proxy channels.

4. RELATED WORK

We have not found other published research work that resembles MC-SSL protocol and this framework as a whole although there are other approaches that address the issues we are concerned about.

There are a few other approaches that address the end-to-end security for the case of chained proxies. One solution is to make application data unreadable to **P** by end-to-end encryption-based tunneling. For instance, the approach proposed by Kwon et al. [3] requires **C** to encrypt data twice: first for **S** using K_S**,** and then for **P** using K_P. Consequently, functions at the application layer, such as content transformation and virus scanning, cannot be performed by **P**.

Another solution is to simultaneously set up a SSL connection and a SSL chain between **C** and **S**, both shown in Figure 1. This approach is adopted by Kennedy [4]. To provide confidentiality, sensitive data is sent through the end-to-end connection instead of **P**. This is a typical approach, but it is insecure for the following reasons: most web/application servers still authenticate their clients using id/password. If **C** trusts **P** and gives its id/password to **P**, then **P** can impersonate **C** in unconstrained fashion. There are a number of solutions for **C** to avoid exposing id/password to **P** including sharing the master key or the symmetric session keys with **P**, or helping **P** sign the verification data. However, **P** can still impersonate **C** in a session and conduct person-in-the-middle attacks. MC-SSL is securer than this approach. In MC-SSL, every entity authenticates each other with their genuine identities; therefore, there is no impersonation in MC-SSL. Besides, every connection between any pair of **C**, **S**, and **P** has exclusive session keys.

A simple extension to SSL has been proposed by Portmann and Seneviratne [5] in order to get an extra cleartext channel without encryption and MAC protection; however, the security strength of this method is questionable because the cleartext channel is established without strict negotiation, and moreover, a malicious attacker can modify and inject

cleartext data at will if there is no security protocol at the application layer. Besides, their approach is not capable of creating other types of channels except the primary end-to-end channel and the extra cleartext channel.

Finally, we would like to compare MC-SSL with XML security solutions including XML Security [6,7] and Web Services Security (WSS) [8,9]. XML Security is a set of core specifications that define XML syntaxes to represent encryption, hash, and digital signature. WSS is a framework that unites a number of existing and developing specifications for the purpose of constructing comprehensive security solutions for XML-based Web services. WSS is based on XML Security. Compared with XML Security and Web Services Security, MC-SSL is a complete and compact protocol under application layer, which is able to provide authentication, key exchange, and secure data transportation for client-server applications with or without the needs of proxies. On the other hand, both XML Security and WSS are not self-contained protocols, and they do not attempt to specify a fixed security protocol for authentication and key exchange so that they can have the extensibility and flexibilities to integrate existing or new security technologies at different layers. As with SSL, MC-SSL can be combined with XML Security, or adopted by WSS for securing Web service. For example, by combining XML Security with MC-SSL, an application can use MC-SSL to do authentication and key exchange for client, server, and proxies, and use XML Security to perform complex encryptions and/or digital signatures on application data.

5. PROXY CHANNEL PROTOCOL

In this section, we explain the design of proxy channel protocol in MC-SSL. We deliberately designed the proxy protocol as a protocol layer on top of SSL as shown in the right part of Figure 5. The left part shows the current Internet architecture. Such a design can keep underlying SSL protocol unchanged if a client needs only the MC-SSL proxy protocol without multiple cipher suites, or needs only SSL to access an SSL-based server without MC-SSL proxy protocol. The whole protocol described in this section deals with primary channels including the primary end-to-end channel and primary proxy channels, such as channel 1 and 4 shown in Figure 4.

The proxy protocol consists of three sub-protocols: handshake, application data, and alert protocols. Handshake protocol sets up the proxy channel, application data protocol defines messages for transporting application data, and alert protocol conveys warnings and fatal errors. The

alert protocol of MC-SSL is similar to that of SSL. We do not describe the alert protocol in this paper. Please refer to RFC 2246 [1].

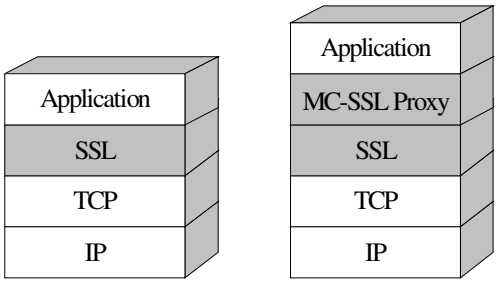

Figure 5. SSL and MC-SSL proxy protocol in the Internet layered architecture

5.1 Handshake protocol

A full handshake is shown in Figure 6. There are four stages: **C**-**S** handshake, **C**-**P** handshake, **P**-**S** handshakes, and confirmations of success. The proxy protocol is based on SSL. Three SSL connections are established to provide basic authentication, confidentiality, and data integrity between peer points. In addition, the **P**-**S** connection may be replaced by a permanent SSL or an IPSec connection.

After setting up a SSL connection, **C** and **S** exchange a pair of hello messages to initiate a MC-SSL session. Both hello messages, i.e. MC_CLIENT_HELLO and MC_SERVER_HELLO, have the following fields: 1) protocol version; 2) session id, which is generated by **S** to identify a MC-SSL session; 3) MAC key, which is used for the MAC in APP_DATA_CONTROL_PROXY messages; 4) the hash algorithm for MAC.

C then sends two messages to **S**: CLIENT_SECURITY_POLICY and CLIENT_CAPABILITIES. They tell **S** about security policy and device capabilities of **C**. Security policy may define whether a proxy is allowed to deliver a certain type of information. Device capabilities include hardware and software information such as screen resolution, power, CPU, memory, OS, browser capabilities, virus scanning capability, etc.

After a MC-SSL session is started, **S** or **C** can start negotiating a proxy channel at any time when necessary. In Figure 6, **S** sends **C** a PROXY_SUGGESTION_S2C message, which contains information such as the purpose of the proxy, the channel direction (simplex or duplex and so on), the DNS name and the certificate of the proxy. **C** sends PROXY_REQUEST_C2S back to **S**. It has similar fields as the previous

message. **S** responds with PROXY_REQUEST_RESPONSE_S2C to give the final decision.

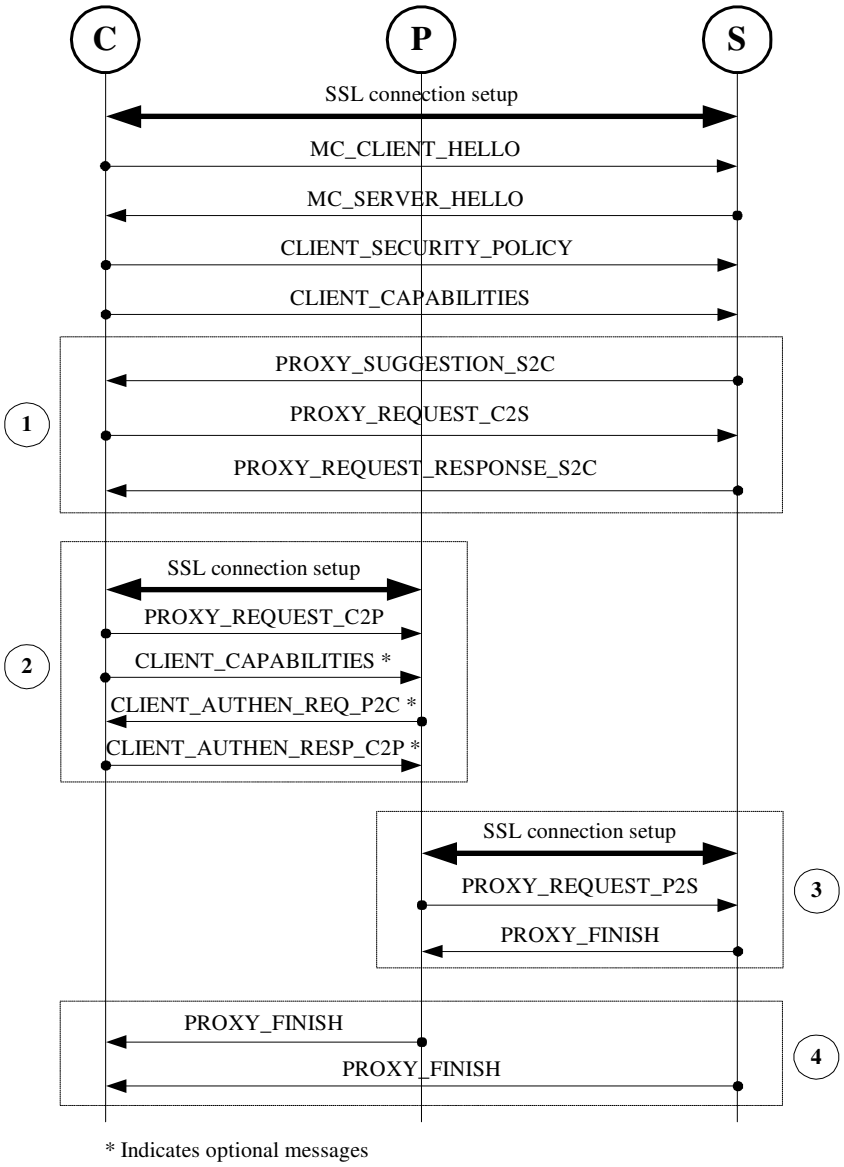

Figure 6. Message flow for a full handshake

The **C**-**P** handshake also starts with a SSL handshake, and then **C** sends **P** a PROXY_REQUEST_C2P message to inform **P** the session id, processing needed, channel direction, preferred authentication methods, handshake type, and the IP address and port number of **S**. In addition, a flag indicates if CLIENT_CAPABILITIES will follow. CLIENT_AUTHEN_REQ_P2C and CLIENT_AUTHEN_RESP_C2P is a pair of messages for **P** to authenticate **C**. The former tells **C** the authentication method, such as user id/password, challenge/answer, or PKI certificate, and the latter returns authentication data.

If the authentication is passed, **P** will start the handshake with **S**. Note that during the SSL handshake **S** and **P** should authenticate each other using their certificates. PROXY_REQUEST_P2S in Figure 6 carries a session id for **S** to bind the proxy channel with the corresponding end-to-end channel. The last three messages return the final result of negotiating a primary proxy channel.

As SSL does, MC-SSL supports the resumption of a cached session, which results in an abbreviated handshake. Some messages in Figure 6, such as the client and server hello messages, the **C-P** and **P-S** proxy request messages, and the final proxy finish message, are still necessary, but others are omitted in an abbreviated handshake.

5.2 Application data protocol

The purpose of the application data protocol is to transport application data between **C** and **S**. There are two ways to do this: one is through the end-to-end channel as shown in the A part of Figure 7; the other is through the proxy channel under the control of the end-to-end channel, as shown in the B part of Figure 7. Figure 7 only shows that **S** sends data to **C**. The transmission in the opposite direction uses the same messages.

To use the end-to-end channel, the sender encapsulates data into an APP_DATA_DIRECT message and sends to the receiver through the end-to-end channel. To transport a piece of content through the proxy channel, three messages are involved: APP_DATA_TO_PROXY, APP_DATA_FROM_PROXY, and APP_DATA_CONTROL_PROXY. They have the same sequence number. The sender wraps the content into APP_DATA_TO_PROXY, and sends it to **P**. After processing the content, **P** generates APP_DATA_FROM_PROXY, and forwards the result to the receiver. Besides, the sender will directly send the receiver an APP_DATA_CONTROL_PROXY message in order to control the behavior of **P**. The rest of this section briefly describes these three messages.

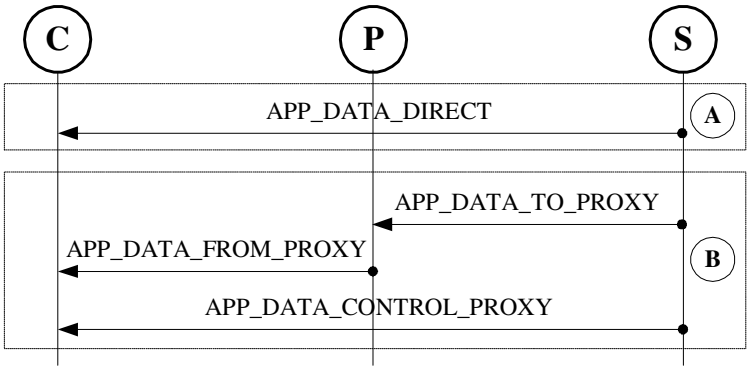

Figure 7. Message flow for transporting application data

APP_DATA_TO_PROXY contains such fields as content, processing request, and change restriction. Field content can be a complete or fragmental piece of content. Processing request tells **P** how to process the content. Change restriction indicates if the content is unchangeable, modifiable, or discardable. APP_DATA_FROM_PROXY includes such fields as content and result. The former is the processed content. The latter outlines the processing result. APP_DATA_CONTROL_PROXY conveys information such as proxy channel id, content attributes, change restriction, and MAC. Proxy channel id indicates which proxy channel the corresponding APP_DATA_TO_PROXY has gone through. Field content attribute is a compound string that describes some attributes of content, e.g. content types. The benefit to know about content types is that the receiver can test if **P** has injected new types of potentially dangerous code. MAC is used for verifying the integrity of unmodifiable content, and is calculated with HMAC hash function described in RFC 2246 [1]. MAC keys and the hash algorithm are negotiated in the initial hello messages.

6. DISCUSSION OF PROXY PROTOCOL

As pointed out in section 2, when an application proxy is needed, some applications allow or force **C** to use a SSL chain to access **S** through a proxy such as a WAP gateway. Consequently, the end-to-end security completely depends on the degree of trustworthiness of **P**. However, it is rare to have an unconditionally trusted **P** administrated by a third party. MC-SSL lets **C** and **S** themselves decide whether or not to have a proxy channel and how to use it after having the secure end-to-end channel in place. Besides other advantages, this feature enables making such a decision after both parties

have determined the degree of trust in each other, which could be a function of the corresponding credentials.

The possible misuse of a proxy is another problem of the SSL chain model. For instance, a user may mistakenly keep using the same proxy configured in his browser even when a proxy or gateway that is hosted (and therefore trusted) by a bank is available for online banking. To avoid such a misuse, when starting a secure session, **C** should always first connect to **S** rather than **P**, then negotiate a proxy with **S**, and finally connect to **P** to set up the proxy channel. That is the order in which MC-SSL proceeds.

The following is our informal discussion of why MC-SSL proxy protocol can enhance end-to-end security in the presence of proxies.

1. **P** is authenticated by **S** as a proxy instead of a client; therefore, **P** cannot impersonate **C**. **P** is not only authenticated by its certificate, but also by the session id received from **C** as a security token especially if session id is implemented to be a cryptographically random string. Moreover, **S** has already obtained the certificate of **P** before verification.
2. As described earlier, because **C** first connects and negotiates with **S** rather than **P**, and the proxy is decided after **S** already has the security policy and terminal capabilities of **C**, the risk of misusing proxies is reduced. **S** can decide if **C** needs a proxy and what type of proxy is needed, and even suggest a proxy.
3. Sensitive data such as id/password and credit card information can be transported through the end-to-end channel, while relatively non-sensitive data including Web pages, software, and email attachments can go through proxies for content scanning or transformation. Note that content scanning can increase system security. Neither does an end-to-end SSL connection nor a SSL proxy chain provide both benefits.
4. A proxy channel can be explicitly negotiated as a one-way channel, which eliminates the chance that **P** turns a response channel into a request channel. For instance, we can have a one-way proxy channel that only allows responses from **S** to **C**; all requests from **C** to **S** have to go through the secure end-to-end channel. As a result, **P** cannot send any fake request to **S** to trigger a transaction.
5. When unmodifiable data is delivered through **P**, the MAC of the data is calculated and sent to the receiver through the end-to-end channel, and hence **P** cannot modify the data without being detected.
6. When modification is permitted, content attributes such as content types are sent to the receiver through the end-to-end channel. Information of content types can prevent **P** from injecting dangerous code or contents.

We also need to analyze security issues still unsettled in MC-SSL proxy protocol. Since the proxy protocol is on top of SSL, the strength of

communication security between two peer points is no weaker than that of SSL. In addition, the cryptographic technique of the end-to-end MAC is also inherited from SSL. Therefore, we do not search for vulnerabilities related to the cryptographic techniques, and we mainly analyze the security issues at the protocol level.

1. It is not easy for **S** and **C** to decide on using which channel to deliver data: end-to-end channel or proxy channel. There are a number of questions to answer such as the following: What is the sensitivity of the data? Is the data too sensitive to be delivered through the proxy? What if sensitive data needs a proxy? What is the worst consequence if **P** modifies the data? What if insensitive data mixes with sensitive data? And so on. For **S**, the security attributes of contents such as sensitivity level need to be defined beforehand. Moreover, **S** needs to get the security policies to help answer those questions. For **C**, it is normally much harder to answer them. A conservative strategy is for **C** to choose the end-to-end channel whenever the answer is unclear. Fortunately, in typical Internet applications, **S** does not have to use a proxy to process the data from **C**. To summarize, **S** and **C** must define their security policies, security attributes of data or contents, and device capabilities. How to define them is beyond the scope of this paper.
2. The proxy protocol alone cannot guarantee that **P** has correctly performed its task. When data is modifiable, there are mainly two types of threats: first, **P** can modify requests or responses between **C** and **S**; second, **P** could inject code if the original content has some embedded code. The strategy to thwart the first threat is neither to send sensitive data through an untrustworthy proxy, nor to use any data from it as sensitive data. To mitigate the second threat, **S** may deliver contents without embedded code, or with embedded code which type is harmless to **C**.

For some of the above problems, XML Security [6,7] is a good solution if both **C** and **S** support it because XML is very flexible to describe data and its attributes. We can use different keys to protect different parts of a XML document. For instance, **S** can use the key for the end-to-end channel to encrypt a XML element that requires end-to-end confidentiality before **S** wraps the XML document into an APP_DATA_TO_PROXY message. As mentioned in Section 4, XML Security can be well combined with MC-SSL to further enhance end-to-end security.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we present Multiple-Channel SSL, a new security protocol extended from SSL. The multiple-channel model of MC-SSL is more flexible and general than SSL, and hence it is able to satisfy diverse requirements for different applications, especially for emerging mobile or wireless applications. MC-SSL exhibits three advantages: first, it improves end-to-end security in the presence of application proxies; second, MC-SSL supports multiple cipher suites in the same connection so that appropriate communication security can be selectively applied to different data or contents; third, MC-SSL supports channel negotiation according to security policies, device capabilities, and security attributes of contents.

We further present the proxy channel protocol including handshake protocol and application data protocol. The handshake protocol completes negotiation, authentication, key exchange, and channel binding, while the application data protocol supports delivering application data through a proxy channel. Besides analyzing the improvement in end-to-end security, our discussion examines some unsettled issues and also suggests solutions to address them.

MC-SSL is our ongoing research work, which we divide into multiple phases. In the first phase, we developed the model and the proxy protocol. In the second phase, we develop the protocol to support multiple cipher suites, and also extend MC-SSL to support multi-hop proxy channels.

REFERENCE

[1] T. Dierks and C. Allen, "The TLS Protocol Version 1.0," RFC 2246, Jan. 1999

[2] M. Abadi and R. Needham, "Prudent Engineering Practice for Cryptographic Protocols," *IEEE Trans. on Software Engineering*, Vol. 22, No. 1, pp. 6-15, Jan. 1996

[3] E.K. Kwon, Y.G. Cho, and K.J. Chae, "Integrated Transport Layer Security: End-to-End Security Model between WTLS and TLS," *Proc. IEEE 15th Int. Conf. on Information Networking*, Jan. 2001

[4] D. J. Kennedy, "An Architecture for Secure, Client-Driven Deployment of Application-Specific Proxies," Master Thesis in Computer Science, University of Waterloo, 2000

[5] M. Portmann and A. Seneviratne, "Selective Security for TLS," *Proc. IEEE 9th Int. Conf. on Networks*, pp. 216-221, Oct. 2001

[6] W3C, *XML Signature Recommendations*, http://www.w3.org/Signature/, Feb. 2002

[7] W3C, *XML Encryption Recommendations*, http://www.w3.org/Encryption/, Dec. 2002

[8] IBM Corp. and Microsoft Corp., "Security in a Web Services World: A Proposed Architecture and Roadmap," http://www-106.ibm.com/developerworks/webservices /library/ws-secmap/, Apr. 2002

[9] OASIS Open, "Web Services Security: SOAP Message Security," http://www.oasis-open.org/committees/documents.php?wg_abbrev=wss, working draft 17, Aug. 2003