PAPER

Authentication Binding between SSL/TLS and HTTP

Takamichi SAITO[†a)], ***Member***, **Kiyomi SEKIGUCHI**[††], ***and*** **Ryosuke HATSUGAI**[††], ***Nonmembers***

SUMMARY While the Secure Socket Layer or Transport Layer Security (SSL/TLS) is assumed to provide secure communications over the Internet, many web applications utilize *basic* or *digest authentication* of Hyper Text Transport Protocol (HTTP) over SSL/TLS. Namely, in the scheme, there are two different authentication schemes in a session. Since they are separated by a layer, these are not convenient for a web application. Moreover, the scheme may also cause problems in establishing secure communication. Then we provide a scheme of authentication binding between SSL/TLS and HTTP without modifying SSL/TLS protocols and its implementation, and we show the effectiveness of our proposed scheme.

key words: *web information systems, web services, security protocol, authentication protocol, SSL/TLS*

1. Introduction

Secure Socket Layer [1] or Transport Layer Security [2] (SSL/TLS) is a de facto standard for secure communications over the Internet. SSL/TLS works in the transport layer of the Open Systems Interconnection (OSI) model and provides *communication security* for the upper layers. However, since SSL/TLS works in the transport layer, its original specification does not provide a password-based authentication scheme operating in the application layer. For example, when a web application server needs to identify a connecting user, it utilizes a Hyper Text Transport Protocol (HTTP) authentication scheme such as basic or digest over SSL/TLS in *server authentication mode* [3], [4]. In this case, however, the web server manages two security channels separately in both application and transport layers. In other words, the web browser authenticates the server by SSL/TLS while the web server authenticates a user of the client by an HTTP authentication scheme. Although it is popular, the scheme designed specially for a web application is not the best way to obtain security. As described above, there are two authentications in a session worked in two layers. Since authentication of SSL/TLS is independent of that of HTTP, SSL/TLS could be lead to man-in-the-middle (MITM) attacks used with *Phising* [5]. On the other, when we need stronger security for communication, we can select to use mutual authentication by using SSL/TLS in *client authentication mode*. However, since this mode forces a client to require its X.509 certificate in accordance with Public Key Infrastructure (PKI) [6], a web application server tends to avoid utilizing this mode. Moreover, a web application often needs its own user authentication scheme for identifying a client. In the case, there is a demand to utilize HTTP authentication. Due to duplications of user authentications, one of user authentications should be eliminated from the scheme.

Therefore, *built-in* ways of password-based authentication are proposed to modify an SSL/TLS handshake protocol such as an RFC standard [7] or Secure Remote Password (SRP) [8], [9]. However, these approaches have the following disadvantages. Since SSL/TLS works in the transport layer, a client as a browser needs to prepare a user interface mechanism to obtain the user's ID and its password when establishing the SSL/TLS channel. The password itself that is input into the client needs to be confirmed before establishing the channel. Then, if the provided password is incorrect due to human error, the browser and the SSL/TLS server need to re-start the handshake from the beginning. This process wastes much computational cost for maintaining the *SSL/TLS context*, especially in the web server.

In this paper, we review related works, and provide a way of binding authentications worked in an application layer rather than in transport layer, for overcoming the disadvantages.

2. Preliminary

2.1 Terms and Symbols

2.1.1 Hosts

SSL/TLS communication consists of a client C and server S. An attacker that could be active or passive is denoted as $\mathcal{I}$. If the attacker is masquerading as server S, the attacker is denoted as $\mathcal{I}(S)$. If it is masquerading as client C, the attacker is denoted as $\mathcal{I}(C)$.

2.1.2 Exchanged Messages

The symbol Msg means an arbitrary message. For example, an expression in which C sends Msg to S is denoted as follows:

$$C \rightarrow S \ : \ Msg$$

When $\mathcal{I}$ masquerading as S sends Msg to C, it is de-

Manuscript received September 16, 2009.
Manuscript revised October 9, 2010.

[†]The author is with Meiji University, Kawasaki-shi, 214–8571 Japan.
[††]The authors are with Nomura Research Institute, Ltd., Tokyo, 100–0005 Japan.
a) E-mail: saito@cs.meiji.ac.jp

DOI: 10.1587/transinf.E95.D.797

noted as follows:

$$\mathcal{I}(S) \rightarrow C \ : \ Msg$$

Note that the following two expressions have different meanings:

$$\begin{array}{ll} C \rightarrow \mathcal{I}(S) & : \ Msg \\ C \rightarrow \mathcal{I} & : \ Msg \end{array}$$

The first one means that C regards the receiver as S, but in fact, C sends it to $\mathcal{I}$. The second means that C regards $\mathcal{I}$ as a legitimate responder and sends Msg to it; namely, C trusts $\mathcal{I}$ as a legitimate server.

2.1.3 Messages

A parameter generated by host C is expressed as X_C, which means X is generated by C. SA is the *cipher suite list*, composed of *protocol version*, *public key algorithm*, *compressed algorithm*, *bulk cipher algorithm* and *MAC algorithm*, which specifies the ciphers supported by each host. RN consists of three components: *date*, *time* and 28 bytes of a *random number*. S_ID is the *SSL/TLS session identifier*, which is held by each host, and it is used in session resumption. The symbol *cert_list* is an *X.509 certificate chain*. *CN* is a *common name* of the server specified in an *X.509 certificate*. The symbol *cert_auth* is a parameter for requesting the client to send back the client certificate, and *cert_type* indicates the type of acceptable certificate. The symbol *pre_master_secret*, from which the session key is derived, is the concatenation of 46 bytes of the *random number* and *protocol version*. The symbol *msg_all* shows all messages exchanged between client and server in that time. The symbol *Ack* is a parameter that informs the opposite side that all necessary messages were transmitted to establish the SSL/TLS connection. The symbol *uid* and *pass*, which are utilized in HTTP authentication, denote the identifier of the user and its password, respectively.

2.1.4 Encryption

A public key is denoted as P, and so P_C means a public key owned by C. $\{Msg\}_Y$ means that Msg is encrypted with an encryption key Y. A secret key corresponding to the public key P is denoted as P^{-1}, then $\{Msg\}_{P^{-1}}$ means that Msg is signed with P^{-1}. KS is a *session key* generated with RNs and *pre_master_secret*, e.g., KS_{CS} or KS_{SC} is a session key between C and S. $h(Msg_1)$ is the hash value of Msg_1. $h(Msg_1, Msg_2, \cdots, Msg_i)$ is also a hash value, which is calculated after concatenating $Msg_1, Msg_2, \cdots,$ and Msg_i.

2.2 HTTP Authentication over SSL/TLS

Although SSL/TLS supports RSA, DH (Diffie-Hellman) and Fortezza as *key exchange algorithms*, not all are discussed here since they are regarded to be same. We only discuss the case of SSL/TLS handshake using RSA as a key

Table 1 SSL/TLS handshake in server authentication mode.

$M1)$	$C \rightarrow S : SA_C, S\_ID_C, RN_C$
$M2)$	$S \rightarrow C : SA_S, S\_ID_S, RN_S$
$M3)$	$S \rightarrow C : cert\_list_S$
$M4)$	$S \rightarrow C : Ack_S$
$M5)$	$C \rightarrow S : \{pre\_master\_secret_C\}_{P_S}$
$M6)$	$C \rightarrow S \ : \{h(KS_{CS}, h(msg\_all_{CS}, C, KS_{CS}))\}_{KS_{CS}}$
$M7)$	$S \rightarrow C \ : \{h(KS_{CS}, h(msg\_all_{CS}, S, KS_{CS}))\}_{KS_{CS}}$

Table 2 Basic authentication of HTTP over SSL/TLS.

$M8)$	$C \rightarrow S \ : \ \{[Request\ Some\_page]\}_{KS_{CS}}$
$M9)$	$S \rightarrow C \ : \ \{[Authorization\ Request]\}_{KS_{CS}}$
$M10)$	$C \rightarrow S \ : \ \{uid_C, pass_C\}_{KS_{CS}}$
$M11)$	$S \rightarrow C \ : \ \{[Requested\_page]\}_{KS_{CS}}$

Table 3 Digest authentication of HTTP over SSL/TLS.

$M8)$	$C \rightarrow S \ : \ \{[Request\ Some\_page]\}_{KS_{CS}}$
$M9)$	$S \rightarrow C \ : \ \{[Authorization\ Request]\}_{KS_{CS}}$
$M10)$	$C \rightarrow S \ : \ \{h(uid_C, pass_C)\}_{KS_{CS}}$
$M11)$	$S \rightarrow C \ : \ \{[Requested\_page]\}_{KS_{CS}}$

exchange algorithm. Table 1 shows SSL/TLS handshake protocol using RSA in server authentication mode.

When using the server authentication mode, the web server cannot identify a connecting user. Therefore, when it requires user authentication, it can utilize an HTTP authentication scheme, basic or digest, after SSL/TLS handshake.

After establishing SSL/TLS connection, triggered by the message requesting user's credential, e.g., `"401 Authorization Required"` from the web server, the browser pops up a window to obtain ID and password of the user. After the user inputs them, the browser sends them to the server. Hereafter, when the authenticated client sends a request massage, it attaches the client's credential to the message. This is shown in Table 2. Where the message `"401 Authorization Required"` is in $M9$.

In addition, the web server can select the protocol in which ID and its password are hashed (cf. Table 3). The option can be specified in the configuration file of a web server such as Apache.

2.3 Consideration of Related Work and Our Approach

As mentioned above, some studies have proposed the integration of password-based user authentication into the SSL/TLS protocol itself. In this subsection, we consider them and show our approach.

2.3.1 Protocol Extensions

One of the most promising proposals is the SSL/TLS extension using SRP [8], [9]. In addition, there is more standardized protocol [7].

These schemes are just extensions of SSL/TLS handshake in order to include password-based user authentication. These approaches force the SSL/TLS protocol to be modified in both client and server. An authenticated SSL/TLS server can authenticate the connecting user just

Table 4 SSL/TLS handshake in client authentication mode.

$M1)$	$C \to S : SA_C, S\_ID_C, RN_C$
$M2)$	$S \to C : SA_S, S\_ID_S, RN_S$
$M3)$	$S \to C : cert\_lists_S, cert\_auth$
$M4)$	$S \to C : cert\_type, Acks_S$
$M5)$	$C \to S : \{pre\_master\_secret_C\}_{P_S}$
$M6)$	$C \to S : \{h(KS_{CS}, h(msg\_all_{CS}, KS_{CS}))\}_{P_C^{-1}}$
$M7)$	$C \to S : \{h(KS_{CS}, h(msg\_all_{CS}, C, KS_{CS}))\}_{KS_{CS}}$
$M8)$	$S \to C : \{h(KS_{CS}, h(msg\_all_{CS}, S, KS_{CS}))\}_{KS_{CS}}$

as in SSL/TLS handshake. As described above, therefore, when the user inputs the wrong password, SSL/TLS handshake could be re-negotiated from the beginning. Moreover, the server must hold connecting status in the handshake.

Another standardized choice is using SSL/TLS in client authentication mode (cf. Table 4). This is commonly regarded as the most secure protocol. When the client signs the message $M6$, the client needs to prepare its public key pair as its certificate for the web server. However, it could be troublesome to do this in many cases, since the user has to obtain an X.509 certificate for the web application.

2.3.2 Authentication Binding

There is a concept to unify two security channels by integrating an upper channel into a lower one [10], [11]. For example, when the web server requires SSL/TLS authentication over Internet Protocol Security (IPsec) in a Virtual Private Network (VPN), this duplication of security wastes computing power and resources. Then, it integrates the upper into the lower. We adopt the approach to integrate SSL/TLS and HTTP authentication scheme. However, for convenient to manage the schemes in web application, we integrate them in the application layer.

To bind two channels, especially from the point of view of authentication, there are the following three combinations:

1. Binding two mutual authentications
2. Binding a mutual authentication and a one-way authentication
3. Binding two opposite one-way authentications.

Although the first and second concepts involve the duplication of security protocols, they can be integrated by channel binding. On the other, in HTTP authentication over SSL/TLS in server authentication mode, a client authenticates a server by SSL/TLS while the server authenticates the client by HTTP. Then, HTTP authentication over SSL/TLS in server authentication mode is regarded as the third case. Note that, in HTTP authentication, the client does not authenticate the server while the server authenticates the client. And, HTTP itself does not encrypt its channel. Then, for binding the two opposite directions of authentications, we propose a new scheme in the application layer. In this paper, we call this integration *authentication binding*, and propose a way of authentication binding between SSL/TLS and HTTP authentication in a secure manner.

3. Design and Implementation

3.1 Requirements

To design a new scheme, we define the following requirements:

1. SSL/TLS is not modified.
2. A server can obtain user's credential after the SSL/TLS connection is established.

Based on SSL/TLS and HTTP authentication, we can apply the combined benefits to network performance, connectivity and security. For example, a programmer of a web application can utilize legacy softwares and libraries.

A client and a server establish an SSL/TLS connection in server authentication mode, and then the server authenticates a user of the client by using HTTP authentication in an arbitrary time. The server acting as a web application can control the timing to obtain the user's credential, when user authentication is required. This feature is more convenient than fixing the timing of authentication in SSL/TLS handshake. Moreover, for example, when utilizing the scheme with *reverse proxy* in server authentication mode, a proxy does not need a password file due to separating password authentication from the SSL/TLS protocol.

To satisfy the above requirements, we need the following features in our proposed system:

1. A web application server and a browser can obtain SSL/TLS session information including the *session key* shared in the SSL/TLS handshake.
2. A web application server can obtain a user password in plaintext.

In the first feature, it is for utilizing SSL/TLS session key and a user's credential in the application layer. In the second, contrary to standard management of a password file in the UNIX system, a web application server here needs to access a password in plaintext, since the client and the server need to bind the above authentications. The detail of binding is described in a later section.

3.2 System Preliminary

In this paper, we implement our proposed scheme with the following software component modules:

- Browser with Network Security Service (NSS)
- Apache web server with `mod_ssl`
- Web application using PHP

NSS [12] is a security library for developing a web client and a server. It supports the development of a browser with SSL, TLS, PKCS, S/MIME, X.509 certificate and other security standards. Using the library, we modify the browser to obtain the credentials of a user and a server, and a session key shared by SSL/TLS handshake. On the server side, we implement a web application, in which a session key can be extracted via PHP [13] and `mod_ssl` [14]. Therefore, we prepare to implement an interface to obtain them on both the server and client side.

3.3 Proposed Scheme

The proposed scheme of authentication binding consists of the following three steps:

1. Establishing SSL/TLS handshake in server authentication mode
2. HTTP authentication over the SSL/TLS channel
3. Exchanging and verifying the credential for binding authentications

While the first and second ones are utilizing an existing way, the third is implemented in this paper. To achieve the binding, the server and the client exchange the messages (from the message 8 to 11 in Table 5) after HTTP authentication over SSL/TLS in server authentication mode:

After establishing SSL/TLS handshake, the client and server share the session key KS_{CS} derived from $pre\_master\_secret_C$. After the client requests a page in $M8$, in an application layer, the server requests the user's credential from the client in $M9$. Then the client sends back ID and $hc_{C \to S}$ in $M10$. In our proposed scheme for binding two authentications, the server and client each compute a hash value locally. They compare them in each side. That is, the server computes $hc_{C \to S}$ and compares it with the received one in $M10$, while the client computes $hc_{S \to C}$ and compares it with the received one in $M11$. If the local value is the same as the received one in each side, we can conclude the channel is integrated securely.

3.4 Exchanging Messages among Modules

In this subsection, we explain the exchange of messages and related computations in the software modules by referring to Fig. 1. In this figure, the numbers correspond to those in the following descriptions, the dotted line is the SSL/TLS communication, and the solid line is an inside exchange in plaintext. Note that, the message 4, 5, 8, 9, 10 and 11 are internal messages of the server.

[1,2] The browser and server establish SSL/TLS channel by a handshake. These correspond to the messages from $M1$ to $M7$ in Table 5.

[3] The client sends the following message to request a page over SSL/TLS. In this case, authentication is required for obtaining `some_page`. This corresponds to $M8$ in Table 5:

```
"GET" <some_page>
```

[4,5] After receiving the message, Apache requests the user's credential with the following message, which corresponds to $M9$ in Table 5:

```
"HTTP/1.1 401 (PJ_Late_Bind_over_ADH-RSA)"
```

where `PJ_Late_Bind_over_ADH-RSA` is a trigger of authentication binding.

[6,7] The Apache module `mod_ssl` encrypts it for sending over the SSL/TLS channel. When receiving this trigger, the client starts to bind the authentication. The client extracts CN and its public key of the server from the server certificate received in the SSL/TLS handshake. With the CN and its public key, the client constructs a part of credential message of the user, i.e., $hc_{C \to S}$, to bind the authentications. The client sends the following message with the credential message:

```
GET <some_page> Authorization: Basic "XXX"
```

This corresponds to $M10$ in Table 5, where `XXX` is the hashed credential $hc_{C \to S}$ that is encoded by Base64.

[8] The module `mod_ssl` decrypts it, and delivers the credential via Apache to the web application.

[9,10] The web application decodes the Base64 data and extracts $hc_{C \to S}$ to compare it with the local $hc_{C \to S}$. If they are the same, authentication binding has succeeded. If not, authentication has failed or has possibly

Table 5 Complete image of proposed scheme.

$M1)\ C \to S$	$: SA_C, S\_ID_C, RN_C$
$M2)\ S \to C$	$: SA_S, S\_ID_S, RN_S$
$M3)\ S \to C$	$: cert\_lists_S$
$M4)\ S \to C$	$: Ack_S$
$M5)\ C \to S$	$: \{pre\_master\_secret_C\}_{P_S}$
$M6)\ C \to S$	$: \{h(KS_{CS}, h(msg\_all_{CS}, C, KS_{CS}))\}_{KS_{CS}}$
$M7)\ S \to C$	$: \{h(KS_{CS}, h(msg\_all_{CS}, S, KS_{CS}))\}_{KS_{CS}}$
$M8)\ C \to S$	$: \{[Request\ Some\_page]\}_{KS_{CS}}$
$M9)\ S \to C$	$:$ `{ HTTP/1.1 401 (PJ_Late_Bind_over_ADH-RSA) }`$_{KS_{CS}}$
$M10)\ C \to S$	$: \{uid_C, hc_{C \to S}\}_{KS_{CS}}$
$M11)\ S \to C$	$: \{CN_S, hc_{S \to C}, [Requested\_page]\}_{KS_{CS}}$

where $hc_{C \to S} = h(CN_S, P_S, h(uid_C, pass_C, KS_{CS}))$
$hc_{S \to C} = h(uid_C, pass_C, h(CN_S, P_S, KS_{CS})))$

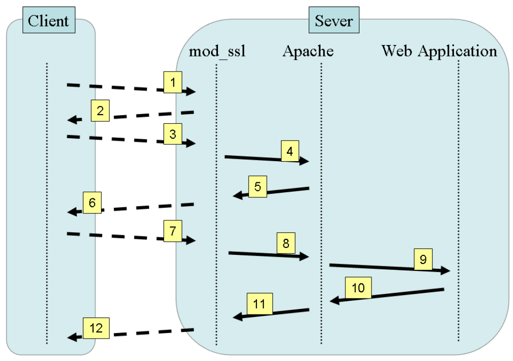

Fig. 1 Binding process.

been attacked.

[11,12] When it is successful, Apache passes the requested page to `mod_ssl`. It then sends the message with $hc_{S \to C}$ encoded by Base64 over the SSL/TLS channel. After receiving the message, the client decodes it by Base64 and verifies it to compare the received $hc_{S \to C}$ and the one computed locally. This message corresponds to $M11$ of Table 5.

4. Security Considerations

4.1 Possibility of MITM Attack

In the case of using an HTTP authentication after establishing SSL/TLS in server authentication mode, it is possible not to protect a password of HTTP authentication [5]. Here we show an attack process in which the password is plundered by an attacker masquerading as a legitimate server. On the other, it should be noted that the proposed scheme can prevent it, since a pre-shared key can be effect to detect an attacker.

4.1.1 Attack Process over SSL/TLS Handshake

As a precondition, a client C has its ID and password registered in a legitimate server S. An attacker $\mathcal{I}$ wants to know the password. And, $\mathcal{I}$ as a web site prepares a server certificate which is tied to the *trust anchor* in a browser of the client C. Then there is a situation that C can trust $\mathcal{I}$ as one of the legitimate servers. Namely, it is possible that C intentionally connects with $\mathcal{I}$. In this case especially, there is no warning message on the browser. Therefore, the attacker can control the situation by cheating the client, for example, by setting up a *phishing* site.

1: C sends the following HTTP request to $\mathcal{I}$.

```
GET /index.html HTTP/1.1
```

1′: Masquerading as C, the attacker $\mathcal{I}(C)$ forwards it to S.

2: S sends `index.html` back to $\mathcal{I}(C)$ with the HTTP response:

```
HTTP/1.1 200 OK
```

2′: $\mathcal{I}$ forwards them to C.

3: Now, it assuming that an HTTPS link written in `index.html`, C sends a next HTTP request over SSL/TLS. Then, C is forced to connect with $\mathcal{I}$ and initiate an SSL/TLS handshake in server authentication mode with the following message:

```
GET /basic_auth/test.html HTTP/1.1
```

Where, this massage is prepared for this explanation as an example page, which requests user's ID and its password registered in S.

3′: After establishing the SSL/TLS connection with C, $\mathcal{I}(C)$ forwards the HTTP request to S. Namely, $\mathcal{I}(C)$ also initiates another SSL/TLS handshake with S.

4: S sends an HTTP response with the status code `401` to $\mathcal{I}(C)$ to state that it requires the user's id uid_C and its password $pass_C$:

```
HTTP/1.1 401 Authorization Required
```

4′: $\mathcal{I}$ forwards it to C.

5: After receiving the HTTP response, in case of HTTP basic authentication, C again sends the HTTP request with uid_C and $pass_C$ to $\mathcal{I}$:

```
GET /basic_auth/test.html HTTP/1.1
Authorization:Basic (uid_C : pass_C)
```

5′: $\mathcal{I}(C)$ forwards it to S. Here, the attacker $\mathcal{I}$ obtains the security credential of the user.

6: After receiving the credential, S verifies it and sends the HTTP response to $\mathcal{I}(C)$:

```
HTTP/1.1 200 OK
```

6′: $\mathcal{I}$ forwards it to C.

It should be noted that the attacker can provide the same service as the legitimate server S does. Therefore, C cannot detect the attack without avoiding the site. Table 6 explains the attack process in more detail. Where, the page requesting messages are omitted here.

Table 6 Attack process against SSL/TLS handshake.

$M1)$	$C \to \mathcal{I}:$	SA_C, S_ID_C, RN_C
$M1')$	$\mathcal{I}(C) \to S:$	$SA_{\mathcal{I}}, S\_ID_{\mathcal{I}}, RN_{\mathcal{I}}$
$M2)$	$S \to \mathcal{I}(C):$	SA_S, S_ID_S, RN_S
$M2')$	$\mathcal{I} \to C:$	$SA_{\mathcal{I}}, S\_ID_{\mathcal{I}}, RN_{\mathcal{I}}$
$M3)$	$S \to \mathcal{I}(C):$	$cert\_list_S$
$M3')$	$\mathcal{I} \to C:$	$cert\_list_{\mathcal{I}}$
$M4)$	$S \to \mathcal{I}(C):$	Ack_S
$M4')$	$\mathcal{I} \to C:$	$Ack_{\mathcal{I}}$
$M5)$	$C \to \mathcal{I}:$	$\{pre\_master\_secret_C\}_{P_{\mathcal{I}}}$
$M5')$	$\mathcal{I}(C) \to S:$	$\{pre\_master\_secret_{\mathcal{I}}\}_{P_S}$
$M6)$	$C \to \mathcal{I}:$	$\{h(KS_{C\mathcal{I}}, h(msg\_all_{C\mathcal{I}}, C, KS_{C\mathcal{I}}))\}_{KS_{C\mathcal{I}}}$
$M6')$	$\mathcal{I} \to C:$	$\{h(KS_{C\mathcal{I}}, h(msg\_all_{C\mathcal{I}}, \mathcal{I}, KS_{C\mathcal{I}}))\}_{KS_{C\mathcal{I}}}$
$M7)$	$\mathcal{I}(C) \to S:$	$\{h(KS_{\mathcal{I}S}, h(msg\_all_{\mathcal{I}S}, \mathcal{I}, KS_{\mathcal{I}S}))\}_{KS_{\mathcal{I}S}}$
$M7')$	$S \to \mathcal{I}(C):$	$\{h(KS_{\mathcal{I}S}, h(msg\_all_{\mathcal{I}S}, S, KS_{\mathcal{I}S}))\}_{KS_{\mathcal{I}S}}$
$M8)$	$C \to \mathcal{I}:$	$\{uid_C, pass_C\}_{KS_{C\mathcal{I}}}$
$M8')$	$\mathcal{I}(C) \to S:$	$\{uid_C, pass_C\}_{KS_{\mathcal{I}S}}$

Table 7 Detecting and preventing the attack.

$M1)$	$C \to \mathcal{I}:$	SA_C, S_ID_C, RN_C
$M1')$	$\mathcal{I}(C) \to S:$	$SA_{\mathcal{I}}, S\_ID_{\mathcal{I}}, RN_{\mathcal{I}}$
$M2)$	$S \to \mathcal{I}(C):$	SA_S, S_ID_S, RN_S
$M2')$	$\mathcal{I} \to C:$	$SA_{\mathcal{I}}, S\_ID_{\mathcal{I}}, RN_{\mathcal{I}}$
$M3)$	$S \to \mathcal{I}(C):$	$cert\_list_S$
$M3')$	$\mathcal{I} \to C:$	$cert\_list_{\mathcal{I}}$
$M4)$	$S \to \mathcal{I}(C):$	Ack_S
$M4')$	$\mathcal{I} \to C:$	$Ack_{\mathcal{I}}$
$M5)$	$C \to \mathcal{I}:$	$\{pre\_master\_secret_C\}_{P_{\mathcal{I}}}$
$M5')$	$\mathcal{I}(C) \to S:$	$\{pre\_master\_secret_{\mathcal{I}}\}_{P_S}$
$M6)$	$C \to \mathcal{I}:$	$\{h(KS_{C\mathcal{I}}, h(msg\_all_{C\mathcal{I}}, C, KS_{C\mathcal{I}}))\}_{KS_{C\mathcal{I}}}$
$M6')$	$\mathcal{I} \to C:$	$\{h(KS_{C\mathcal{I}}, h(msg\_all_{C\mathcal{I}}, \mathcal{I}, KS_{C\mathcal{I}}))\}_{KS_{C\mathcal{I}}}$
$M7)$	$\mathcal{I}(C) \to S:$	$\{h(KS_{\mathcal{I}S}, h(msg\_all_{\mathcal{I}S}, \mathcal{I}, KS_{\mathcal{I}S}))\}_{KS_{\mathcal{I}S}}$
$M7')$	$S \to \mathcal{I}(C):$	$\{h(KS_{\mathcal{I}S}, h(msg\_all_{\mathcal{I}S}, S, KS_{\mathcal{I}S}))\}_{KS_{\mathcal{I}S}}$
$M8)$	$C \to \mathcal{I}:$	$\{uid_C, h(CN_{\mathcal{I}}, P_{\mathcal{I}}, h(uid_C, pass_C, KS_{C\mathcal{I}}))\}_{KS_{C\mathcal{I}}}$
$M8')$	$\mathcal{I}(C) \to S:$	$\{uid_C, h(CN_{\mathcal{I}}, P_{\mathcal{I}}, h(uid_C, pass_C, KS_{C\mathcal{I}}))\}_{KS_{\mathcal{I}S}}$
		or
$M8'')$	$\mathcal{I}(C) \to S:$	$\{uid_C, h(CN_S, P_S, h(uid_C, pass_C, KS_{\mathcal{I}S}))\}_{KS_{\mathcal{I}S}}$

4.1.2 Detecting the Attack

In this subsection, we explain how our proposed scheme can detect the attack. The preconditions are the same as ones in the case of Sect. 4.1.1 (cf. Table 7). Where, the page-requesting messages are omitted here.

After receiving the message $M8$, the attacker has to send the valid message to the server. If the attacker tries to send the message $M8'$, the server S can detect the existence of an attack since S expects to receive $M8''$. However, it cannot construct the valid message $M8''$ to complete the attack in the next step, since the attacker does not have or cannot know the valid password of the user.

4.1.3 Misleading Attack

This kind of attack such like phising seems to be prevented by a user's carefulness when connecting a web server about to be deployed by the attacker. A client cannot prevent the attack without checking the server certificate every access. However, since the attacker can utilize a legitimate certificate which is tied to the trust anchor in a browser and there are a lot of web sites, without a warning from a browser, it is not easy for most users to decide if there is an attacker.

As shown in Sect. 4.1.2, a server in the scheme can detect the attacker even when connecting a phishing site under the same conditions. Therefore, the proposed scheme is one of solutions for this kind of attack. It should be noted that the attacker cannot obtain user's password in this scheme, even in case of connecting with the attacker's phishing site, contrary to an HTTP authentication over SSL/TLS. In an HTTP authentication over SSL/TLS, a client sends a user's ID and its password in plaintext. Then, if misconnect with phishing site, ID and its password are snatched.

4.2 Off-Line and Replay Attack

Due to using a hashed message rather than over SSL/TLS encryption, the proposed scheme could not reveal a password. Namely, in the scheme, since the password is actually hashed as an authentication key, it cannot be obtained by the attacker if the SSL/TLS connection is compromised.

Moreover, each SSL/TLS session utilizes a different key to prevent the attacker from guessing an encryption key. That is, SSL/TLS holds Perfect Forward Security (PFS). Then, a replay attack cannot be effective against the scheme, which means the scheme is invulnerable to these attacks.

5. Conclusion

In this paper, we introduce the concept of authentication binding between SSL/TLS and HTTP authentication without modifying SSL/TLS protocol, and we implement our proposed scheme to show its effectiveness. In summary, our proposed scheme provides yet another method to establish a secure channel for a web application.

As we had dealt only with binding authentications in SSL/TLS in server authentication mode, we will apply our scheme to SSL/TLS in client authentication mode in the future work.

References

[1] A. Frier, P. Karlton, and P. Kocher, "The SSL 3.0 protocol," Netscape Communications Corp., Nov. 1996.
[2] T. Dierks and C. Allen, "The TLS protocol Version 1.0," RFC 2246, Jan. 1999.
[3] E. Rescorla, "HTTP over TLS," RFC 2818, May 2000.
[4] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, E. Sink, and L. Stewart, "HTTP authentication: Basic and digest access authentication," RFC 2617, June 1999.
[5] T. Saito, "A scenario-based protocol checker for the public-key authentication scheme," IEICE Trans. Inf. & Syst., vol.E92-D, no.6, pp.1268–1279, June 2009.
[6] R. Housley, W. Ford, W. Polk, and D. Solo, "Internet public key infrastructure: Part I: X.509 certificate and CRL profile," RFC 2459, Jan. 1999.
[7] P. Eronen and H. Tschofenig, "Pre-shared key ciphersuites for transport layer security (TLS)," RFC4279, Dec. 2005.
[8] T. Wu, "SRP-6: Improvements and refinements to the secure remote password protocol," Oct. 2002, http://srp.stanford.edu/srp6.ps
[9] T. Wu, "The SRP authentication and key exchange system," RFC 2945, Sept. 2000.
[10] J. Linn, "Generic security service application program interface Version 2," RFC2743, Jan. 2000.
[11] N. Williams, "On the use of channel bindings to secure channels," RFC 5056, Nov. 2007.
[12] http://www.mozilla.org/projects/security/pki/nss/
[13] http://www.php.net/
[14] http://www.modssl.org/

Takamichi Saito Associate Professor, Department of Computer Science, Faculty of Science, Meiji Univertsity.

Kiyomi Sekiguchi April 2009, Nomura Research Institute, Ltd.

Ryosuke Hatsugai IT Security Architect, Managed Security Services Division, NRI SecureTechnologies, Ltd.