Browser Enhancements to Support SSL/TLS Session-Aware User Authentication

Rolf Oppliger[1], Ralf Hauser[2], and David Basin[3]

[1] eSECURITY Technologies Rolf Oppliger
Beethovenstrasse 10, CH-3073 Gümligen, Switzerland
Phone/Fax: +41 (0)79 654 8437
E-mail: rolf.oppliger@esecurity.ch
[2] PrivaSphere AG
Fichtenstrasse 61, CH-8032 Zürich, Switzerland
Phone: +41 (0)43 299 5588, Fax: +41 (0)1 382 2133
E-mail: hauser@privasphere.com
[3] Department of Computer Science, ETH Zurich
Haldeneggsteig 4, CH-8092 Zürich, Switzerland
Phone: +41 (0)44 632 7245, Fax: +41 (0)44 632 1172
E-Mail: basin@inf.ethz.ch

Abstract. SSL/TLS session-aware user authentication is a new approach that can be used to protect browser- and SSL/TLS-based online applications, like Internet banking, against man-in-the-middle (MITM) attacks. In this position paper, we suggest three browser enhancements that simplify the implementation of MITM-resistant authentication with minimum changes to the SSL/TLS protocol stacks in use.

1 Introduction

Man-in-the-middle (MITM) attacks pose a serious threat to browser- and SSL/TLS-based online applications, like Internet banking, one reason being that the average user has difficulties validating a server certificate. There are only a few technologies available to mitigate these risks. In [OHB06a], we introduced the notion of SSL/TLS session-aware user authentication to protect SSL/TLS-based online applications against MITM attacks. The main idea is to make the user authentication depend not only on the user's (secret) credentials, such as a password or personal identification number (PIN), but also on state information related to the SSL/TLS session in which the credentials are being transferred to the server. The rationale behind this idea is that the server should have the possibility to determine whether the SSL/TLS session in which it receives the credentials is the same as the user employed when he sent out the credentials in the first place.

– If the two sessions are the same, then there is probably no MITM involved.
– If the two sessions are different, then something abnormal is taking place. It is likely that a MITM is located between the user's client system and the server.

Using SSL/TLS session-aware user authentication, the user authenticates himself by providing a user authentication code (UAC) that depends on both his credentials and the SSL/TLS session (in particular, information from the SSL/TLS session state). A MITM who gets hold of the UAC can no longer misuse it by simply retransmitting it. The key point is that the UAC is bound to a particular SSL/TLS session, and if the UAC is submitted on another session, then the server can easily recognize this fact and drop the session. As such, SSL/TLS session-aware user authentication provides a lightweight alternative to the deployment and rollout of a public key infrastructure (PKI) to protect against MITM attacks.[4]

There are a number of possibilities to implement SSL/TLS session-aware user authentication. In [OHB06a], we argued (i) that software-based implementations are inherently vulnerable, (ii) that one should therefore pursue hardware-based implementations in the first place, and (iii) that a particularly promising possibility is the use of hardware tokens, preferably in the form of impersonal PKCS #11-compliant authentication tokens. In [OHB06b], we broadened the scope of our ideas and we also considered possibilities to implement (parts of) SSL/TLS session-aware user authentication in software, and investigated the security implications of doing this. We discussed possibilities to make 2- and 3-factor approaches, such as one-time password (OTP) and/or challenge-response (C/R) systems be SSL/TLS session-aware. This includes, for example, one-time passwords that are distributed on scratch lists (an approach that is still widely used in Internet banking applications).

In the rest of this position paper, we suggest three browser enhancements that would simplify the implementation and deployment of SSL/TLS session-aware user authentication considerably. The first enhancement is very simple and consists of making an internally used hash value available. The two other enhancements are more involved and require internal modifications in the browser or its SSL/TLS protocol implementation. It goes without saying that the server side must also be enhanced to support SSL/TLS session-aware user authentication. These enhancements, however, are quite simple and can be restricted to the servers that actually employ SSL/TLS session-aware user authentication. The handshake protocol messages remain the same. Only an "observer" is needed that calculates extra values using the messages exchanged. In one variant, the signature of the CertificateVerify message is overloaded with additional information, thus the implementation of the SSL/TLS protocol stack must allocate sufficiently large buffers to transparently pass-on this amended value.

2 Enhancement I: Making the Session-Identifying *Hash* Available

In [OHB06a, OHB06b], we described several possibilities to implement SSL/TLS session-aware user authentication that logically link the user authentication to an

[4] Of course, the deployment of a PKI often has other objectives, e.g., the ability to provide nonrepudiation services.

SSL/TLS session by using the cryptographic hash value *Hash* that is computed from all messages previously exchanged during the execution of the SSL/TLS handshake protocol.

If the user authentication method under consideration has access to *Hash*, then it is simple to make the user authentication be SSL/TLS session-aware. In the case of a one-time password system, for example, *Hash* is taken as another input argument to generate the one-time password (in addition to the secret key). In the case of a C/R system, an appropriately compressed version of *Hash* may serve as challenge. In the case of a scratch list, again an appropriately compressed version of *Hash* can determine the currently valid scratch list entry. It seems to be the case that—given a possibility to make *Hash* available to the user authentication method—there is always a possibility to make a user authentication be SSL/TLS session-aware.

Against this background, we suggest that browsers should be modified in a way that they can be configured to make *Hash* available to user authentication methods. A simple possibility is to have the browser show (parts of) *Hash* next to the SSL/TLS padlock and to copy it to the clipboard when appropriately clicking on the padlock. It goes without saying that the encoding of (parts of) *Hash* must be configurable by the user. For example, it may be necessary to show only the leading characters of the hash value in a specific encoding. A simple step-by-step scenario with an external soft-C/R device has been described in RFE `https://bugzilla.mozilla.org/show_bug.cgi?id=322661`.

3 Enhancement II: Integrating OTP and C/R Mechanisms

Most browsers in use today have mechanisms in place to manage user passwords (in a password store). Instead of using an external password store for an OTP or C/R mechanism, it seems reasonable to adapt the internal one. In fact, the password store can be enhanced to also support OTP and C/R mechanisms. If the browser supports such a mechanism, then it is straightforward to make it SSL/TLS session-aware. Note that the browser always has access to its SSL/TLS session state, and hence it can easily take into account the *Hash* value to generate a UAC.

One possibility that looks promising and deserves further study is to redefine HTTP Digest Access Authentication (RFC 2617) to be SSL/TLS session-aware. The use of HTTP Digest Access Authentication on top of an SSL/TLS session is somehow unorthodox; if made SSL/TLS session-aware, however, it may still provide an interesting alternative to the use of client-certificate based SSL/TLS. The major advantage we see is that users can keep on using their current passwords and that they do not need any additional hardware, software, or public key certificate.

4 Enhancement III: Overloading the SSL/TLS CertificateVerify Message

Provided the allocated, opaque buffers in the browser's current SSL/TLS protocol stack implementation (that transport, for example, the PKCS#11- or CAPI-based external signatures) are sufficiently large, the CertificateVerify message can be overloaded within a specially crafted device driver as per [OHB06a]. For broader user acceptance, it is preferable not to require such a MITM-resistance-providing external device driver to be installed by each user, but to have the corresponding logic available within the browser. So, if one modifies the SSL/TLS CertificateVerify message in a way that it is able to additionally encode the UAC when receiving the request for client certificate authentication from a set of reserved CA DNs,[5] then SSL/TLS session-aware user authentication can be implemented in a transparent way (from the user's viewpoint). In fact, there are at least three possibilities:

1. The browser can digitally sign both the $Hash$ value and the UAC.
2. The browser can digitally sign a keyed hash value (instead of the hash value $Hash$), where the UAC represents the key, $Hash$ represents the argument, and the HMAC construction is used to key the hash function.
3. The browser can only send the keyed hash value (instead of the digital signature). This possibility is similar to the second possibility—the only difference is that the keyed hash value is not digitally signed. Consequently, there is no need to have a private signing key on the token.
4. The browser encrypts the $Hash$ and the scratch-list OTP (plus some nonces) with a hard-coded server public key.

Among the advantages of doing this are:

- MITM-resistant authentication is not only available to HTTPS, where the UAC can be provided to the server in HTML forms, but also other protocols a client may support, such as SFTP.
- There is a new set of protocols possible that do not require a shared secret between the personally transferable device, but having a pre-distributed server public key should be sufficient (see option 4).

5 Conclusions and Outlook

In this position paper, we suggested three browser enhancements that simplify the implementation of SSL/TLS session-aware user authentication that is resistant against MITM attacks. In the short-term, we think that enhancement I is appropriate and sufficiently simple to implement and deploy (because it neither requires modifying any content in the SSL/TLS handshake protocol messages nor altering the sequence of them—it only observes them). In fact, we are currently

[5] Note that all browsers should handle the DNs in a similar way.

working on a proof-of-concept implementation for a large financial institution in Switzerland. In the medium- or long-term, however, we think that browser manufacturers should consider the possibility to implement enhancement II or III. These possibilities are interesting, mainly because they can be implemented in a way that negligibly changes in the user behaviour. The user only enters his credentials, and the rest goes under the hood. In particular, no new hardware or software is needed.

References

[**OHB06a**] Oppliger, R., Hauser, R., and D. Basin, "SSL/TLS Session-Aware User Authentication—Or How to Effectively Thwart the Man-in-the-Middle," submitted for publication, `http://www.esecurity.ch/OHB06a.pdf`

[**OHB06b**] Oppliger, R., Hauser, R., and D. Basin, "SSL/TLS Session-Aware User Authentication Revisited," submitted for publication, `http://www.esecurity.ch/OHB06b.pdf`