SSL/TLS Session-Aware User Authentication—Or How to Effectively Thwart the Man-in-the-Middle

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. Man-in-the-middle attacks pose a serious threat to SSL/TLS-based electronic commerce applications, such as Internet banking. In this paper, we argue that most deployed user authentication mechanisms fail to provide protection against this type of attack, even when they run on top of SSL/TLS. As a possible countermeasure, we introduce the notion of SSL/TLS session-aware user authentication, and present different possibilities for implementing it. More specifically, we start with a basic implementation that employs impersonal authentication tokens. Afterwards, we address extensions and enhancements and discuss possibilities for implementing SSL/TLS session-aware user authentication in software.

Keywords. Security, man-in-the-middle (MITM) attack, SSL/TLS protocol, user authentication, electronic commerce

1 Introduction

Most electronic commerce (e-commerce) applications in use today employ the Secure Sockets Layer (SSL) or Transport Layer Security (TLS) protocol to authenticate the server and to cryptographically protect the communication channel between the client and the server. When setting up a secure channel, the main point where SSL/TLS-based applications differ concerns user authentication. Options available include passwords, personal identification numbers (PINs), transaction authorization numbers (TANs), TAN (or scratch) lists, and more sophisticated one-time password (OTP) or challenge-response systems, such as

Lamport-style OTPs [Lam81] or OTPs generated with SecurID tokens. In contrast, there are only a few applications that employ public key certificates for user authentication as part of the SSL/TLS session establishment. In fact, the deployment of public key certificates on the client side has turned out to be slow—certainly slower than it was originally anticipated (e.g., [LOP05]).

In spite of the fact that many researchers have analyzed the security of the SSL/TLS protocol, relatively few shortcomings and vulnerabilities have actually been identified [WS96, MSS98, Pau99]. Most of these problems are theoretically interesting but not practically relevant. Examples include the adaptive chosen ciphertext attacks against PKCS #1 version 1.5 [Ble98] and version 2.0 [Man01], as well as the vulnerabilities found in the padding scheme of the CBC encryption mode [Vau02]. More problematic in practice is the possibility that the protocol is used in some inappropriate way, or that it is used in an environment that makes its security properties meaningless because the actual threats are different than the ones assumed in the design phase (e.g., [And94]). The practical problems in using SSL/TLS have turned out to be substantial and it is reasonable to assume that the majority of SSL/TLS-based applications are vulnerable to attacks. In the recent past, for example, phishing and Web spoofing attacks have become widespread. They represent special forms of man-in-the-middle (MITM) attacks, and they are particularly powerful if visual spoofing is employed [OG05]. If a MITM can place himself between the user and the server, then he can act as a relay and authenticate himself to the server on behalf of the legitimate user. Even worse, if the MITM operates in real-time (i.e., with a small time delay), then there is hardly any user authentication mechanism (decoupled from SSL/TLS session establishment) that cannot be defeated or misused.

The question we address in this paper is how to protect SSL/TLS-based e-commerce applications (and their users) against MITM attacks. We think that this question is highly relevant and has not received adequate attention, either in the literature or in practice. The solution we propose is to make user authentication depend on the current SSL/TLS session, i.e., user authentication is bound to, and only valid for, a specific SSL/TLS session. This thwarts the MITM by making it ineffective for him to resubmit a user's credentials on a SSL/TLS session other than the one established between the user's Web browser and the application server. We will present in this paper different ways of achieving this.

The rest of this paper is organized as follows: In Section 2, we present background on SSL/TLS and MITM attacks. In Section 3, we survey related work. In Section 4, we introduce the notion of SSL/TLS session-aware user authentication and, in Section 5, we present different possibilities for implementing it. In Section 6, we draw conclusions.

2 Background

In this section, we briefly summarize the SSL/TLS protocol and the notion of MITM attacks.

2.1 SSL/TLS Protocol

The SSL protocol is a transport layer security protocol that was developed and proposed by Netscape Communications in the 1990s.[4] The proposal was the starting-point for the IETF Transport Security Layer (TLS) working group whose task was to develop a transport layer security protocol of the same name. The resulting protocol is known as TLS protocol and it is specified in RFC 2246 [DA99].

The SSL and TLS protocols are essentially the same.[5] Part of the protocol is a handshake protocol that is responsible for (mutual) authentication and key establishment. The SSL/TLS handshake protocol can be summarized as follows:

1: C $\longrightarrow$ S	ClientHello
2: S $\longrightarrow$ C	ServerHello
	Certificate
	ServerKeyExchange
	CertificateRequest
	ServerHelloDone
3: C $\longrightarrow$ S	Certificate
	ClientKeyExchange
	CertificateVerify
	ChangeCipherSpec
	Finished
4: S $\longrightarrow$ C	ChangeCipherSpec
	Finished

First, the client (C) and the server (S) exchange ClientHello and ServerHello messages to synchronize with each other. In step 2, S also provides a public key certificate to C in a Certificate message, an optional ServerKeyExchange message (not further addressed in this paper), an optional CertificateRequest message if the server wants the client also to authenticate itself using a public key certificate, and a ServerHelloDone message to finish the message sequence. In step 3, C optionally provides a public key certificate in a Certificate message (if the server requests the client to authenticate itself using a public key certificate), a pseudo-randomly generated master secret for the SSL/TLS session encrypted with the server's public key (found in the Certificate message) in a ClientKeyExchange message, and an optional CertificateVerify message (again, if the server wants the client to authenticate itself using a public key certificate). Finally, C and S exchange ChangeCipherSpec and Finished messages. Afterwards, all messages that are subsequently transmitted between C and S can be cryptographically protected in terms of authenticity, integrity, and confidentiality with cryptographic keys derived from the master secret.

[4] On August 12, 1997, Netscape Communications was granted U.S. patent 5,657,390 entitled "Secure socket layer application program apparatus and method" for the technology employed by the SSL protocol.

[5] There are some subtle differences as explained, for example, in Chapter 6 of [Opp03].

2.2 MITM Attacks

According to RFC 2828, a MITM attack refers to "a form of active wiretapping attack in which the attacker intercepts and selectively modifies communicated data in order to masquerade as one or more of the entities involved in a communication association" [Shi00]. Consequently, the major characteristics of MITM attacks are (i) that they represent active attacks, and (ii) that they target the associations between the communicating entities (rather than the entities or the communication channels between them). Note that in some literature, a MITM that carries out an active attack in real-time is called adaptive. We will not use this term and a MITM attack will be adaptive by default.

There are many possibilities to implement MITM attacks. Examples include Address Resolution Protocol (ARP) cache poisoning and Domain Name System (DNS) spoofing.[6] The MITM attacks we have in mind operate at or above the transport layer. In either case, a MITM attack is very powerful; the attacker can do literally everything the user is authorized to do on the server side (or everything the server is authorized to do on the client side, respectively).

In a typical setting, the attacker places himself between the user and the server in a way that he can talk to the user and the server separately, whereas the user and the server think that they are talking directly with each other. The best way to think about a MITM attack is to consider an adversary that represents an SSL/TLS proxy server (or relay) between the user and the server. Neither the user nor the server are aware of the MITM. Cryptography makes no difference here as the MITM is in the loop and can decrypt and reencrypt all messages that are sent back and forth. If the user wants to authenticate himself to an application server, then he reveals his credentials to the MITM. Afterwards, the MITM may choose to use the credentials fairly or to misuse them to illegitimately spoof the user. If, for example, the user employs a SecurID token to authenticate himself to a server, then the MITM can grab the SecurID string (that is typically valid for a couple of seconds up to a minute) and reuse it to spoof the user. If the user employs a challenge-response authentication system, then the MITM can simply send back and forth the challenge and response messages. Even if the user employed a zero-knowledge authentication protocol [FS87, GQ88], then the MITM would still be able to forward the messages and spoof the user accordingly. The zero-knowledge property of an authentication protocol does not, by itself, protect against MITM attacks—it only protects against information leakage related to the user's secret.[7]

Against this background, we make a case that most currently deployed user authentication mechanisms fail to provide effective protection against MITM attacks, even when they run on top of the SSL/TLS protocol. We see two main reasons for this.

[6] Recently, the term pharming has been coined to refer to DNS spoofing attacks, such as local DNS cache poisoning.

[7] Note in this regard that there is a construction due to Cramer and Damgård which can be used to immunize a zero-knowledge authentication protocol against MITM attacks [CD97].

1. SSL/TLS server authentication is usually done poorly by the naïve end user, if done at all.
2. SSL/TLS session establishment is usually decoupled from user authentication.

The first leads to a situation in which the user talks to the MITM, thereby revealing his credentials to the MITM. The second means that the credentials revealed by the user can be reused by the MITM to spoof the user to the origin server. Consequently, any effective countermeasure against MITM attacks in an SSL/TLS setting must address these problems by either enforcing proper server authentication or combining user authentication with SSL/TLS session establishment. As explained below, we think that the second approach is more appropriate and simpler to deploy.

3 Related Work

First of all, it is important to note that the SSL/TLS protocol is able to protect against MITM attacks. Hence, if one employed and took advantage of all of the protocol's features, then this would provide the necessary protection. This requires, however, (i) that all clients have personal public key certificates, and (ii) that server authentication is done properly. Both requirements are difficult.[8] This is particularly true for requirement (ii). Requirement (i) could be relaxed, if one extended the SSL/TLS protocol with some alternative client authentication methods (in addition to certificate-based authentication). For example, the adoption of password-based key exchange protocols is proposed in [S+01] and the use of the Secure Remote Password (SRP) protocol [Wu98] for TLS authentication is ongoing work within the IETF.[9] We think that this work is important and useful, and we regret that there is little other activity within the IETF to extend the TLS specification with other client authentication methods. In either case, we cannot count on the security properties of the SSL/TLS protocol alone. Instead, we have to assume a setting in which the SSL/TLS protocol is only used to authenticate the server and the user authenticates himself on top of an established SSL/TLS session using some additional authentication mechanism. Furthermore, we must assume that the certificate-based server authentication is done improperly by the ordinary user. In such a setting, it is very likely that MITM attacks will occur, typically sooner than later.

In spite of the fact that researchers have long been aware of MITM attacks, there are only a few protection mechanisms against them. Most mechanisms proposed are independent of SSL/TLS. For example, Rivest and Shamir proposed the Interlock protocol [RS84] that was later shown to be vulnerable when used for authentication [BM94]. Jakobsson and Myers proposed a technique called

[8] In some scenarios, such as Internet banking, the two requirements are not independent from each other.

[9] http://www.ietf.org/internet-drafts/draft-ietf-tls-srp-09.txt

delayed password disclosure (DPD) that can be used to complement a password-based authentication and key exchange protocol to protect against a special form of MITM attack—called a doppelganger window attack [JM05]. Unfortunately, DPD requires a password-based authentication and key exchange protocol and is not qualified to protect against the MITM attacks we have in mind.[10] Kaliski and Nyström proposed the notion of a password protection module (PPM) that provides some protection against MITM attacks [KN04]. PPMs are effective only if they take into account network-related information, such as IP addresses and URLs. In this case, however, PPMs are very difficult to deploy and manage. Finally, Asokan et al. proposed protection mechanisms to secure tunneled authentication protocols against MITM attacks [ANN03]. Their work is most closely related to our findings.

Instead of cryptographic techniques and protocols, some researchers have suggested employing multiple communication channels and channel hopping to thwart MITM attacks [ASS03]. This approach has its own disadvantages and is not practical for Internet-based applications. Similarly, there are several applications—especially in Europe—that authenticate users by sending out short messaging system (SMS) messages that contain TANs and require that users enter these TANs when they login. While it has been argued that this mechanism protects against MITM attacks, unfortunately, this is not the case. If a MITM is located between the user and the server, then he need not eavesdrop on the SMS messages; all he needs to do to spoof the user is to forward the TAN submitted by the user on the SSL/TLS session. If one wanted to work with TANs distributed via SMS messages, then one would have to work with transaction-based TANs.[11] For each transaction submitted by the user, a summary is returned to the user together with a TAN in an SMS message. To confirm the transaction, the user must enter the corresponding TAN. The downside of this proposal is that transaction-based TANs are expensive (perhaps prohibitively so) and not particularly user-friendly. Furthermore, they require the use of secondary networks, such as GSM.

It is a frequently quoted argument in the security industry that strong (possibly two-factor) user authentication mechanisms are needed to thwart MITM attacks.[12] This argument is wrong. Vulnerability to MITM attacks is not a user authentication problem; it is a server authentication problem. MITM attacks are possible mainly because SSL/TLS server authentication is usually done poorly, as explained above. In other words: if users properly authenticated the server with which they establish an SSL/TLS session, then they would also be protected against MITM attacks. Unfortunately, this is not the case and it is questionable whether this is possible at all. Note that a MITM can employ many tricks to give the user the impression of being connected to an origin server, for example using visual spoofing, which is becoming increasingly popular. In the most extreme case, one may think of a MITM that is able to control the graphical

[10] This point is, for example, further addressed in attack scenario three of [JM05].
[11] http://www.cryptomathic.com/pdf/The%20Future%20of%20Phishing.pdf
[12] www.antiphishing.org/sponsors_technical_papers/PHISH_WP_0904

user interface of the user's browser. In such a scenario, the user has almost no possibility to recognize that he is subject to a MITM attack.

4 SSL/TLS Session-Aware User Authentication

If the user is not able to properly enforce server authentication, then perhaps it is possible to technically enforce it. In theory, this seems to be straightforward: just make sure that the client digitally signs a CertificateVerify message during the execution of the SSL/TLS handshake protocol if and only if the server provides an appropriate public key certificate in a Certificate message. In practice, this is not as simple as it seems to be, mainly because it is not clear what public key certificate is appropriate. If one considers the use of a PKCS #11-compliant authentication token, which comes with a list of appropriate public keys and certificates, then this token normally has no access to the browser's SSL/TLS cache (which holds the server certificate for the SSL/TLS session). The only way out is to implement the SSL/TLS protocol, or at least parts of it, on the token itself. This approach, however, has its own disadvantages (e.g., performance) and is certainly not the preferred choice. The bottom line is that it is difficult and cumbersome to technically enforce proper server authentication without substantially modifying the client's or browser's SSL/TLS implementation.

In this paper, we propose another approach that is based on combining user authentication with SSL/TLS session establishment. We use the term *SSL/TLS session-aware user authentication* to refer to it. The main idea is to make the user authentication depend not only on the user's (secret) credentials, 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 the credentials and the SSL/TLS session (in particular, on 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.

There are a number of possibilities to implement SSL/TLS session-aware user authentication. Some of these possibilities are addressed next.

5 Implementation

We make a distinction between hardware-based and software-based implementations.

- In the first case, we are talking about hardware tokens (i.e., hard-tokens). Such a token may be (physically) connected or not. By connected, we mean there is a direct communication path in place between the token and the client system. This includes, for example, galvanic connections, as well as connections based on wireless technologies, such as Bluetooth or infrared.
- In the second case, we are talking about software tokens (i.e., soft-tokens).

Furthermore, an authentication token (be it hardware-based or software-based) can be personal or impersonal.

- If the token is personal, then it comprises a user-specific private key and this key can be used directly to authenticate the user. In this case, we are in the realm of full-fledged PKI solutions.
- If the token is impersonal, then it comprises a private key, but this key is not user-specific and hence cannot be used to authenticate the user. Instead, the user must authenticate himself by some other means on top of the SSL/TLS session. The use of impersonal token was originally proposed by Molva and Tsudik in [MT93]. Impersonal tokens are commercially interesting, mainly because they are inexpensive and do not require a user registration process.[13]

In either case, the token's private key must have signing capabilities, since it is used to digitally sign the client's CertificateVerify message of the SSL/TLS handshake protocol. This message, in turn, represents the hash value of all messages previously exchanged during the SSL/TLS handshake protocol execution. Part of this message is the server's Certificate message that comprises the server's public key certificate (the server's public key is included in this certificate), and hence the CertificateVerify message is indirectly bound to the server's public key.

Furthermore, an authentication token can be consistent with a cryptographic token interface standard, such as PKCS #11 [RSA04] or Microsoft's cryptographic application programming interface (CAPI). We assume that a standard CAPI driver is preinstalled on newer versions of the Windows operating system, and that this driver can be used by the Microsoft Internet Explorer to drive the token in some transparent way (from the user's viewpoint). On all other platforms, we assume that some PKCS #11 driver software must be installed to drive the token. This includes, for example, the case in which a Mozilla browser is employed on a Windows platform.

From a theoretical viewpoint, we prefer hardware tokens over software primarily because software-based approaches can be more easily attacked and manipulated. Trusted computing platforms may someday provide an alternative.

[13] In fact, we think that the user registration process is the cost driver for any PKI solution.

However, these platforms are not yet widely deployed, and it is questionable whether they will ever become available on a large scale [OR05]. Furthermore, we think that an authentication token should be compliant to the PKCS #11 standard. In either case, we prefer impersonal tokens because they do not depend on a user registration process, which simplifies matters considerably. More specifically, we envision impersonal tokens that can be plugged into a client system, and which can combine user authentication with SSL/TLS session establishment and thereby make user authentication SSL/TLS session-aware. Such a token-based implementation is addressed first. Afterwards we present extensions and modifications, provide an informal security analysis, and discuss possibilities for implementing SSL/TLS session-aware user authentication in software.

5.1 Token-based Implementation

In our token-based implementation, the following entities play a role:

- A user U;
- An impersonal PKCS #11-compliant authentication token T with a small display;
- A client (i.e., browser) C that is used by U to access an SSL/TLS-based application;
- An SSL/TLS-enabled server S that hosts the application.[14]

The entities are equipped with various parameter values and cryptographic keys. U is equipped with an identifier ID_U and a PIN PIN_U. PIN_U, in turn, is a secret that U shares with S, for example, a few decimal digits. Obviously, some mechanisms must be in place (not addressed in this paper) that allow U to manage his PINs. T is identified with a publicly known serial number SN_T.[15] Furthermore, T is equipped with both a public key pair (k, k^{-1})—of which the private key k^{-1} is used to digitally sign the CertificateVerify messages—and a secret token key K_T. The keys k and k^{-1} are the same for all tokens (which is why the tokens are impersonal in the first place), whereas K_T is unique and specific to T (it is not specific to the user). K_T can be generated randomly or pseudo-randomly using a master key MK:

$$K_T = E_{MK}(SN_T)\,.$$

Consequently, there is no need to centrally store all the token keys. Instead, K_T can be generated dynamically (by anybody who knows MK) from SN_T. MK is typically possessed and held exclusively by S.

When U wants to access S, he directs his browser C to S. C and S then try to establish an SSL/TLS session. As part of the SSL/TLS handshake protocol,

[14] We do not differentiate between S and the application it hosts. Conceptually, one may think of S as a dedicated server, i.e., a server that exclusively hosts only the application under consideration.

[15] The serial number may, for example, be imprinted on the token.

S authenticates itself using a public key certificate (we do not require the user to properly verify this certificate at this point). S is configured in a way that it always requires client authentication by sending out a CertificateRequest message to C. After C receives the message, it knows that it must authenticate itself by returning a properly signed CertificateVerify message to S. As mentioned above, this message comprises a digitally signed hash value of all previously exchanged messages (this value is further referred to as $Hash$). The signature is generated by T using its private key k^{-1}. Due to the fact that T is an impersonal token, the CertificateVerify message cannot really authenticate the client (as k^{-1} is the same for all tokens). Instead, the CertificateVerify message only ensures that C uses a token to establish an SSL/TLS session with S. In addition to providing a properly signed CertificateVerify message to S, the token also renders a shortened version of

$$N_T = E_{K_T}(Hash)$$

in decimal notation on its display.[16] N_T is shortened to the length of PIN_U (e.g., 8 digits) by truncating it. This value must then be combined with PIN_U to generate a user authentication code (UAC) that is valid for exactly one SSL/TLS session of U. If f is a function that combines N_T and PIN_U, then the UAC is computed as

$$UAC = f(N_T, PIN_U). \tag{1}$$

In general, there are many ways to define an appropriate function f. One possibility, which we adopt from [MT93], is to let f be the digit-wise addition modulo 10 of N_T and PIN_U.

The server must authenticate the user by asking him to enter ID_U, SN_T, and a valid UAC within the previously established SSL/TLS session.[17] On the server side, S can verify the UAC because it knows f and PIN_U and because it can construct N_T (since it knows $Hash$ and the master key MK that is used to dynamically generate the token key K_T that is shared with T). Obviously, the server authentication software must be designed in a way that it has access to the SSL/TLS cache (to retrieve $Hash$). Clearly this is much simpler to achieve on the server side.

Note that our token-based implementation does not require synchronized clocks. Instead, it employs nonces derived from the symmetrically encrypted hash values used by the SSL/TLS protocol.

[16] Alternatively, N_T could also represent a message authentication code (MAC) computed with a keyed one-way hash function (in this case, K_T represents the key). For example, the HMAC construction (e.g., [Opp05]) could be used to generate N_T: $N_T = HMAC_{K_T}(Hash)$

[17] Note that the user need not enter SN_T if this value is included in a public key certificate for T's private key k^{-1}. In this case, the server S can retrieve SN_T from the public key certificate. Similarly, one could also have the user register with a specific token. In this case, S can retrieve ID_U from its registration database and set it as a default value in the user authentication process. Both possibilities are not further addressed in this paper.

5.2 Extensions and Enhancements

The basic schema may be enhanced and extended in various ways. For example, instead of using the CertificateVerify message to make a user authentication be SSL/TLS session-aware, an implementation may also use the Finished message (that also includes a hash value of all previously exchanged messages). Also, instead of symmetrically generating N_T on the client and server side, N_T can be transmitted as part of the CertificateVerify message. In this case, however, N_T must be encrypted with a public key for which the private key is known only to the server. The big advantage we see is that the token does not have to store a secret token key K_T anymore (it only has to store a public key for the server). Taken this idea of step further, one may also think of having the CertificateVerify message include the UAC (in addition to the digitally signed hash value and the encrypted value of N_T). In this case, the user authentication is handled entirely by the token and the user does not have to enter anything in a Web form.

As briefly surveyed next, several other extensions and enhancements are possible.

Mutual authentication: The token can be extended to support server authentication. In this case, the token must be able to display both N_T and an authentication code

$$A_T = E_{K_T}(g(Hash)),$$

where g is a publicly known length-preserving function. N_T and A_T can be displayed on two separate displays, on one display (in concatenated form), or alternating on one display.

Complementing a one-time password or challenge-response system: The token can be used to complement a strong authentication system, such as a OTP system (e.g., Lamport-style OTPs or SecurID tokens) or a challenge-response system. In the first case, U employs the OTP as input for f (instead of PIN_U). In the second case, N_T (or a truncated version thereof, respectively) basically represents the challenge.

Relieving the user from computing f: The token can be extended to relieve the user of the burden of computing f. Instead of having U compute the UAC, one may imagine a token that contains a small keypad. In this case, the token can implement a pseudorandom function PRF and generate

$$UAC = PRF_{PIN_U}(N_T)$$

in a way that is transparent to the user. The main advantage here is that the user no longer has to manually compute an addition modulo 10. Alternatively, one may also consider the use of a software tool that asks the user to enter his PIN and provides the token with the user-entered value. Again, the UAC is then computed on the token. Of course, the client system should be free of malware in this case, as otherwise a Trojan horse may be used to request the user's PIN and make it available to the adversary.

Use of biometrics: An increasingly popular (but sometimes also overblown) option is to personalize computing devices with biometric authentication mechanisms. Authentication tokens, even impersonal ones, are good candidates for such personalization and can be extended with a biometric authentication step prior to activation. The use of biometrics is particularly interesting when combined with the previous extension of using a keypad to relieve the user from computing f.

Multi-institution tokens: The tokens proposed above can be used to authenticate a user to a single institution, using a single PIN. In a more general setting, one may employ a multi-institution token, which can authenticate a user to multiple institutions. Such a token is simple to design, but it may be considerably more involved to market it.

Some of these extensions and enhancements are further explored in a companion paper [OHB06].

5.3 Informal Security Analysis

We claim that our token-based approach protects users against MITM attacks in an SSL/TLS setting and we briefly sketch the reasons for this here.

Consider an adversary M that tries to mount a MITM attack against U. We assume that M can establish an SSL/TLS session to C and another SSL/TLS session to S. On top of the first sessions, M asks U to enter his UAC and forwards it on the second session to S (claiming to be U). In this situation, S will abort the second session, spotting that the UAC it obtains does not logically belong to the associated SSL/TLS session. This occurs because the hash value that T sees (when it generates N_T) and the hash value S sees (when it executes the SSL/TLS handshake protocol) are different, with a high probability (because the hash function in use is assumed to be collision-resistant). Consequently, the resulting UACs are different and hence S will refuse to accept the submitted UAC.

Another concern one may have is that M succeeds in retrieving PIN_U from a valid UAC. For this purpose, M may set up an SSL/TLS session to C (claiming to be S) and request a valid UAC. Because the UAC depends on N_T and N_T depends on $Hash$ (that is now known to M), it may appear feasible to retrieve PIN_U from this value. In this case, however, appearances are deceptive and this attack does not work. Note that N_T is encrypted with K_T—a key that is known only to T and S—and hence there is no easy way for M to construct N_T (from $Hash$) without knowing K_T. It goes without saying that the attack is successful if the adversary knows or can learn K_T or MK (in the second case, K_T can be computed by the adversary). The bottom line is that we must make the assumption that the symmetric encryption system used is resistant against known- and chosen-plaintext attacks. This is, however, a standard assumption in applied cryptography, and all practically relevant symmetric encryption systems, such as 3DES or AES, have modes of operation that satisfy this assumption (e.g., [B+97]).

In a similar line of argumentation, one may be concerned that U is not able to properly authenticate S (i.e., the server that requests the UAC). If this poses a problem, then the authentication code A_T may be employed. In this case, U authenticates himself to S with an appropriate UAC, and S authenticates itself to U with an appropriate authentication code. Due to the fact that the UAC is not particularly useful for an adversary, we recommend the use of authentication codes only in high-security environments.

5.4 Software-based Implementation

In a software-based implementation, the hard-token is replaced with a soft-token. This means that the token's functionality is simulated in software and that the token's display is emulated on the display of the client system. The soft-token may still be compliant to PKCS #11, CAPI, or any other cryptographic token interface standard. In this case, the basic functionality and interface of the soft-tokens remain essentially identical to the hard-tokens.

On the one hand, soft-tokens are more flexible and less expensive than hardware-based solutions. On the other hand, soft-tokens have additional security problems that must be dealt with.

1. Soft-tokens are inherently vulnerable to malware and keylogger attacks. Malware can, for example, read out cryptographic keys. Keylogger attacks can be used to retrieve the user credentials when they are typed in.
2. Soft-tokens are vulnerable to visual spoofing attacks.

Both problems are difficult to solve. Keylogger attacks can be partially solved by displaying a keyboard on the client' screen and having the user type in his credentials using this keyboard. The second problem is particularly tricky. One has to find means to have the soft-token's GUI display authentic information. As of this writing, there are only a few (visual) technologies that can be used for this purpose (e.g.,[YS02, DT05]).

In one way or another, a software-based implementation of SSL/TLS session-aware user authentication must have access to some SSL/TLS state information, in particular the $Hash$ value.

- If the soft-token is consistent with PKCS #11 or CAPI, then it has immediate access to this information (similar to the hard-token). In this case, the implementation of the soft-token is essentially the same. This includes, for example, the necessity to install driver software on the client system.
- If, however, the soft-token is not consistent with PKCS #11 or CAPI, then it has no immediate access to this information. In this case, the situation is slightly more involved and one must employ other means to access the $Hash$ value. One possibility is to modify the browser in a way that it is able to render and display the first digits of the $Hash$ (or $compress(Hash)$, respectively) value as it appears in the execution of the SSL/TLS handshake protocol. These digits can, for example, be displayed near the closed padlock icon that marks the SSL/TLS session (typically at the bottom right

of the bowser window). The character set and length of $compress(Hash)$ may be configurable, and thereby meet the requirements of different user authentication mechanisms and systems.

In the second case, the users can be equipped with a simple program that implements a UAC calculator. The UAC calculator must compute and display the currently valid UAC according to formula (1). This value must then be entered by the user in a Web form or transferred to the server S as part of the SSL/TLS CertificateVerify message. In this case, we have the possibility to work with nonces encrypted with a server public key instead of $Hash$. The advantage we see in this case is that there is no secret key that must be stored on the token. In either case, the user must have the assurance that the first digits of $Hash$ displayed by the soft-token are authentic and can somehow be verified. Otherwise, an attacker can fake the digits and use the UAC to launch a PIN-guessing attack. As noted previously, there are only a few technologies that can be used for to establish a trusted path between the browser and the user. We see this as one of the major challenges in a software-based implementation.

6 Conclusions and Outlook

MITM attacks pose a serious threat to many relevant SSL/TLS-based applications, such as Internet banking and remote Internet voting. In this paper, we argued that most deployed user authentication mechanisms fail to provide protection against this type of attack, even when they run on top of SSL/TLS. We introduced the notion of SSL/TLS session-aware user authentication, and elaborated on possibilities to implement it. More specifically, we started with a basic implementation that employs impersonal authentication tokens, addressed extensions and enhancements, gave an informal security analysis, and presented possibilities for implementing SSL/TLS session-aware user authentication in software. As of this writing, we are prototyping SSL/TLS session-aware user authentication for Internet banking, and we expect to have a proof-of-concept implementation ready for demonstration soon. The implementation is software-based and employs many ideas raised in Section 5. Most specifically, the implementation renders and displays the first digits of the $Hash$ (or $compress(Hash)$, respectively) value as it appears in the execution of the SSL/TLS handshake protocol. We hope that this extension makes it as a "trusted observer" extension into the default browsers.

We believe that the time to tackle this problem is running out and we expect that real-time variants of MITM attacks will take place soon. When this happens, it is important to have a full understanding of the problem as well as the space of possible solutions. Many mechanisms currently promoted by industry simply are not effective against this problem. This includes TAN lists (including the iTAN mechanism), TANs delivered with SMS messages, SecurID tokens, and many other two-factor authentication devices.[18] A mechanism we think is effective is the use of personal authentication tokens on the client side (to be used

[18] http://www.acm.org/technews/articles/2005-7/0316w.html#item3

in the SSL/TLS session establishment) and a very restrictive browser configuration. This mechanism, however, is very difficult to deploy and enforce in the real world; this is particularly true for the restrictive browser configuration. Consequently, we see SSL/TLS session-aware user authentication as a pragmatic and lightweight alternative to the deployment of a PKI-based solution. In fact, we do not require any redesign or change of the deployed SSL/TLS infrastructure. Any existing and deployed OTP system can be made SSL/TLS session-aware (e.g., [OHB06]). This protects the security investments made in the past. Furthermore, several institutions can work together to issue multi-institution tokens.

Finally, we hope that our work will stimulate further technical discussions about MITM attacks and their feasibility against applications protected by SSL/TLS and similar protocols. We believe that these applications are much less secure in practice than they appear to be in theory and that this misconception is very dangerous.

Acknowledgments

We thank Sebastian Gajek, Virgil Gligor, Christian Graber, and Michael Steiner for their valuable comments on previous drafts of this paper.

References

[ANN03] Asokan, N., Niemi, V., and K. Nyberg, "Man-in-the-Middle in Tunneled Authentication Protocols," *Proceedings of the International Workshop on Security Protocols*, 2003, pp. 15–24 (also available as IACR ePrint 2002/163).

[ASS03] Alkassar, A., Stüble, C., and A.-R. Sadeghi, "Secure Object Identification—or: Solving The Chess Grandmaster Problem," *Proceedings of the 2003 Workshop on New Security Paradigms*, Ascona, Switzerland, ACM Press, NY, 2003, pp. 77–85.

[And94] Anderson, R.J., "Why cryptosystems fail," *Communications of the ACM*, Vol. 37, No. 11, November 1994, pp. 32–40.

[B+97] Bellare, M., et al., "A Concrete Security Treatment of Symmetric Encryption: Analysis of the DES Modes of Operation," *Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS '97)*, 1997, pp. 394–403.

[Ble98] Bleichenbacher, D., "Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1," *Proceedings of CRYPTO '98*, Springer-Verlag, August 1998, pp. 1–12.

[BM94] Bellovin, S.M., and M. Merritt, "An Attack on the Interlock Protocol When Used for Authentication," *IEEE Transactions on Information Theory*, Vol. 40, No. 1, January 1994.

[BMV05] Basin, D., Mödersheim, S., and L. Viganò, "OFMC: A symbolic model checker for security protocols," *International Journal of Information Security*, Vol. 4, No. 3, June 2005, pp. 181–208.

[CD97] Cramer, R., and I. Damgård, "Fast and Secure Immunization Against Adaptive Man-in-the-Middle Impersonation," *Proceedings of EUROCRYPT '97*, Springer-Verlag, LNCS 1233, May 1997, pp. 75–87.

[DA99] Dierks, T., and C. Allen, "The TLS Protocol Version 1.0," Request for Comments 2246, January 1999.

[DT05] Dhamija, R., and J.D. Tygar, "The Battle Against Phishing: Dynamic Security Skins," *Proceedings of the 2005 ACM Symposium on Usable Security and Privacy (SOUPS 2005)*, ACM Press, July 2005, pp. 77–88.

[FS87] Fiat, A., and A. Shamir, "How To Prove Yourself: Practical Solutions to Identification and Signature Problems," *Proceedings of CRYPTO '86*, Springer, LNCS 263, 1987, pp. 186–194.

[GQ88] Guillou, L.C., and J.J. Quisquater, "A Practical Zero-Knowledge Protocol Fitted to Security Microprocessor Minimizing both Transmission and Memory," *Proceedings of EUROCRYPT '88*, Springer-Verlag, LNCS 330, 1988, pp. 123–128.

[JM05] Jakobsson, M., and S. Myers, "Stealth Attacks and Delayed Password Disclosure," 2005
`http://www.informatics.indiana.edu/markus/stealth-attacks.htm`.

[KN04] Kaliski, B., and M. Nyström, "Authentication: Risk vs. Readiness, Challenges & Solutions," presentation held at the BITS Protecting the Core Forum, October 6, 2004
`http://www.rsasecurity.com/rsalabs/staff/bios/bkaliski/publications/other/kaliski-authentication-risk-readiness-bits-2004.ppt`.

[Lam81] Lamport, L., "Password Authentication with Insecure Communication," *Communications of the ACM*, Vol. 24, 1981, pp. 770–772.

[LOP05] Javier, L., Oppliger, R., and G. Pernul, "Why Have Public Key Infrastructures Failed so far?" *Internet Research*, Vol. 15, No. 5, October 2005

[Man01] Manger, J., "A Chosen Ciphertext Attack on RSA Optimal Asymmetric Encryption Padding (OAEP) as Standardized in PKCS#1 v2.0," *Proceedings of CRYPTO '01*, Springer-Verlag, August 2001, pp. 230–238.

[MSS98] Mitchell, J., Shmatikov, V., and U. Stern, "Finite-state analysis of SSL 3.0," *Proceedings of the Seventh USENIX Security Symposium*, USENIX, 1998, pp. 201–216.

[MT93] Molva, R., and G. Tsudik, "Authentication Method with Impersonal Token Cards," *Proceedings of IEEE Symposium on Research in Security and Privacy*, IEEE Press, May 1993.

[OG05] Oppliger, R., and S. Gajek, "Effective Protection Against Phishing and Web Spoofing," *Proceedings of the 9th IFIP TC6 and TC11 Conference on Communications and Multimedia Security (CMS 2005)*, Springer-Verlag, LNCS 3677, September 2005, pp. 32–41.

[OHB06] Oppliger, R., Hauser, R., and D. Basin, "SSL/TLS Session-Aware User Authentication Revisited," work in progress.

[Opp03] Oppliger, R., *Security Technologies for the World Wide Web, Second Edition*, Artech House Publishers, Norwood, MA, 2003.

[Opp05] Oppliger, R., *Contemporary Cryptography*, Artech House Publishers, Norwood, MA, 2005.

[OR05] Oppliger, R., and R. Rytz, "Does Trusted Computing Remedy Computer Security Problems?" *IEEE Security & Privacy*, Vol. 3, No. 2, March/April 2005, pp. 16–19.

[Pau99] Paulson, L.C., "Inductive analysis of the Internet protocol TLS," *ACM Transactions on Computer and System Security*, Vol. 2, No. 3, 1999, pp. 332–351.

[RS84] Rivest, R.L., and A. Shamir, "How to Expose an Eavesdropper," *Communications of the ACM*, Vol. 27, No. 4, 1984, pp. 393–395.

[RSA04] RSA Laboratories, "PKCS #11 v2.20: Cryptographic Token Interface Standard," June 28, 2004.

[S+01] Steiner, M., et al., "Secure Password-Based Cipher Suite for TLS," *ACM Transactions on Information and System Security (TISSEC)*, Vol. 4, No. 2, May 2001, pp. 134–157.

[Shi00] Shirey, R., "Internet Security Glossary," Request for Comments 2828, May 2000.

[Vau02] Vaudenay, S., "Security Flaws Induced by CBC Padding—Applications to SSL, IPSEC, WTLS ... ," *Proceedings of EUROCRYPT '02*, Amsterdam, Netherland, Springer-Verlag, LNCS 2332, 2002, pp. 534–545.

[YS02] Ye, Z.E., and S. Smith, "Trusted Paths for Browsers," *Proceedings of the USENIX Security Symposium*, 2002, pp. 263–279.