Strengthening Web Authentication through TLS - Beyond TLS Client Certificates

Andreas Mayer[1], Vladislav Mladenov[2], Jörg Schwenk[2], Florian Feldmann[2], and Christopher Meyer[2]

[1]Adolf Würth GmbH & Co. KG, Künzelsau-Gaisbach, Germany
[2]Horst Görtz Institute, Ruhr-University Bochum, Germany

Abstract: Even though novel identification techniques like Single Sign-On (SSO) are on the rise, stealing the credentials used for the authentication is still possible. This situation can only be changed if we make novel use of the single cryptographic functionality a web browser offers, namely TLS. Although the use of client certificates for initial login has a long history, only two approaches to integrate TLS in the session cookie mechanism have been proposed so far: Origin Bound Client Certificates in [DCBW12], and the Strong Locked Same Origin Policy (SLSOP) in [KSTW07].

In this paper, we propose a third method based on the TLS-unique API proposed in RFC 5929 [AWZ10]: A single TLS session is uniquely identified through each of the two Finished messages exchanged during the TLS handshake, and RFC 5929 proposes to make the *first* Finished message available to higher layer protocols through a novel browser API. We show how this API can be used to strengthen all commonly used types of authentication, ranging from simple password based authentication and SSO to session cookie binding.

Keywords: SSL, TLS, Session Cookies, Password based Authentication, Single-Sign-On, TLS-unique, Strong Locked Same Origin Policy

1 Introduction

Today, the most prevalent method for user authentication towards a server in a network environment consists of a username/password combination, where the username is usually a publicly known value and the password is a secret value known only to the corresponding user. By providing the correct password for a given username, the user proves his identity to the server.

After a correct initial authentication, the server issues a *cookie* containing a unique string identifying an authenticated session between user and server. Subsequent queries from the user to the server will then include this cookie and thus do not require another explicit authentication.

Nowadays, every user is working with various services on the Web, thus, a user has to create and manage multiple passwords. To mitigate the overhead of creating and remembering multiple secure passwords, SSO systems can be utilized. These systems allow a user to authenticate himself to an Identity Provider (IdP) which acts as a trusted third party towards both the user and the Service Provider (SP) (the server providing the desired service). After the user authenticated himself to IdP, the IdP authenticates the user towards an SP by issuing a cryptographically secured authentication token. The user can use this token to prove his identity to SP without requiring further explicit authentication by username/password.

Attacks on Authentication Methods In practice, many problems can arise in such authentication contexts. Passwords can be leaked, either via Phishing or during transport.

Cookies and authentication tokens can be eavesdropped or stolen via various attack techniques and used by an attacker to impersonate the user.

And even though the authentication data sent between the participants is almost always protected by secure channels, i.e. TLS sessions, it has been shown in the past (e.g., [Mar09] or [BDLF+14]) that TLS sessions are susceptible to Man-in-the-Middle (MitM) Attacks, potentially enabling an attacker to steal passwords, cookies and/or authentication tokens during transport.

Contribution Motivated by the existing threats and serious vulnerabilities found within today's authentication processes and the associated identity management, we propose a novel approach to strengthen authentication and trust between the participants.

Our main goal is to use certain features of the TLS session between participants to enhance the protection of the entire communication within the current HTTP-session. The distinct contributions of this paper are the following:

- Introduction of a novel binding of passwords to the applied TLS session during the login phase in order to prevent MitM attacks regarding the transmitted credentials.
- Reinforcement of the session cookies in conjunction with a TLS channel to mitigate the theft via XSS, CSRF and UI-Redressing attacks.
- Amplification of SSO protocols by cryptographically binding the issued security tokens to the applied TLS session.
- Only minor adjustments to the existing infrastructures are proposed, enabling an easy implementation and integration of the introduced mechanisms.

The security mechanisms introduced in this article are based on the sole assumption that the simple TLS-unique interface from RFC5929 [AWZ10] is implemented in the browser.

The paper is organized as follows: Section 2 discusses related work pointing other approaches and their respective differences to our approach. Section 3 explains the technical foundations our solution is based on. We define the threat model in Section 4 and list the technical preconditions in Section 5. Section 6 explains the theory behind our proposed solution and Section 7 provides information regarding possible implementations of the concept. We conclude our paper in Section 8.

2 Related Work

Several efforts have been proposed [PS00, FSSF01, LKHG05] to secure cookies by deploying modern public key-based authentication mechanisms. However, neither signing nor encrypting cookies prevents an adversary from transferring a cookie from one browser to another, thus rendering these efforts ineffective against cookie theft and similar attacks.

In 2007, Karlof et al. [KSTW07] proposed to strengthen the web browser's Same Origin Policy (SOP) by taking the server's TLS certificate into account. They recommended two variants, called *weak-* and *strong-locked Same Origin Policy*, respectively. The weak-locked SOP is easier to implement and enforces that web objects are sent only to servers with valid certificate chains (e.g. certificates included in the chain can neither be selfsigned nor expired). The strong-locked SOP tags each web object with the server's public key and solely returns them to a server if that public key matches the key from the TLS certificate

of the current TLS connection to the server.

A similar approach called *Web Server Key Enabled Cookies* (WSKECookies) was presented by Masone et al. [MBS07]. In this concept, the browser stores cookies along with the public key of the web server which initially set them. A WSKECookie is only returned to a web server which proved possession of the same key pair in following TLS sessions. In both concepts stolen web objects or WSKECookies are not cryptographically bound in any way – neither to the TLS channel nor to the client. Therefore, the attacker can potentially steal them and then use them to authenticate as the victim.

In 2008, Gajek et al. [GJMS08] proposed a variant of a browser-based Kerberos scheme using TLS client certificates. The concept has been standardized as *SAML Holder-of-Key Web Browser SSO Profile* by OASIS[1] [KS10]. In contrast to our proposed solution, the Holder-of-Key approach protects neither HTTP cookies nor password-based logins, but only the SAML assertion within the Single Sign-On based authentication.

RFC 5929 [AWZ10] was published in 2010 as proposed standard. The document describes three channel binding types for TLS, namely *tls-unique*, *tls-server-endpoint*, and *tls-unique-for-telnet*. TLS-unique specifies the API for our proposed binding solutions.

A solution specifically designed for channel bindings within SAML frameworks has been described in [KSJG10].

In 2012, Dietz et al. [DCBW12] proposed a TLS channel binding called *Origin-Bound Certificates* (OBC) by using a TLS extension. Their approach changes server authenticated TLS channels into mutually authenticated channels by using client certificates created on the fly by the browser. In consequence, their idea requires changes to the TLS protocol, which would affect all current TLS implementations. They propose to use the issued OBC by cryptographically binding them to HTTP cookies or SSO tokens.

Google introduced another TLS extension called *Channel ID* [BH12], which again requires fundamental changes to underlying TLS implementations. In summary, the browser creates an additional asymmetric key pair during the TLS handshake and uses the private key to sign all handshake messages up to the `ChangeCipherSpec` message. Subsequently, the signature, along with the public key, is sent encrypted through the TLS channel using the established TLS key material. This is done, before finishing the TLS handshake. The browser uses the public key as "Channel ID" that identifies the TLS connection.

In 2013, OASIS published the *SAML Channel Binding Extensions* allowing the use of channel bindings in conjunction with SAML [Can13]. This standard allows the proposed TLS channel bindings to be integrated in all SAML related services, e.g. SSO.

3 Technical Foundations

In order to exchange sensitive data between a client and a server, first a mutually authenticated secure channel has to be established between the two communication partners. Establishing such a channel requires multiple steps, where server side authentication and data protection during transport are usually handled via Transport Layer Security (TLS). Section 3.1 briefly discusses the details of a TLS handshake phase, resulting in establish-

[1] https://www.oasis-open.org/

ment of a server-side authenticated secure channel between the parties. Usually, the client then has to provide an authentication for himself. This client authentication can be done directly towards the server by using a mutual secret, i.e. a password, between client and server, as described in Section 3.2. Section 3.3 describes means to make this authentication persistent, so the user does not have to enter the password for every HTTP-request. Alternatively, authentication can be done indirectly with the help of a trusted third party in an SSO scenario, as shown in Section 3.4.

3.1 Transport Layer Security (TLS)

Transport Layer Security (TLS) [DR08] provides multiple security goals, such as

(1.) *Confidentiality* (2.) *Authenticity* (3.) *Integrity* and (4.) *Replay protection.*

These goals are achieved by different cryptographic primitives like encryption, Keyed-Hash Message Authentication Codes (HMACs), sequence numbers and nonces. However, TLS protects data only during transport and does not provide any end-to-end payload security. Thus, message level security requires additional mechanisms at a higher level in the protocol stack.

TLS consists of a two-phase architecture: The handshake phase and the application phase (see Figure 1).

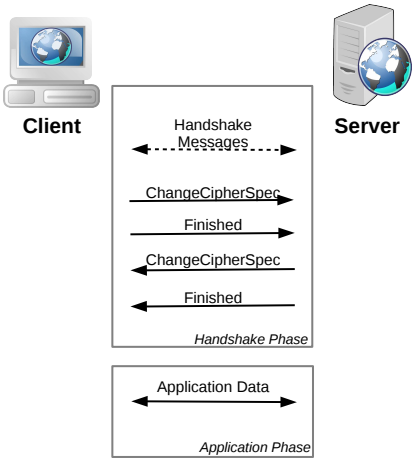

Figure 1: TLS Handshake Phase and Application Phase.

The handshake phase includes multiple messages sent between client and server and serves to establish algorithms, cryptographic primitives and an authenticated agreement on a shared secret between the participants. All key material required for a secure communication is derived from this secret.

Handshake messages are exchanged unencrypted until the `ChangeCipherSpec` protocol is invoked to activate cryptographic protection at the corresponding party (thus, after both parties sent their corresponding `ChangeCipherSpec` messages, all communication between the two parties will be encrypted). All handshake messages may only occur

in an exactly specified order and must follow the predefined workflow.

A parameter of interest for the concept of TLS Unique is the first Finished message (*Finished*) message. This *Finished* message is the first message encrypted with the negotiated keys and contains a hash value of all handshake messages previously exchanged between client and server. The hash especially includes all random values exchanged within the handshake, thus, the *Finished* message can be seen as a unique fingerprint of the TLS session in use.

After both parties have established the shared secret and confirmed this by sending the proper *Finished* message, the protocol continues with the application phase where the actual payload data is exchanged via the secure communication channel.

3.2 Initial Client Authentication

The initial client authentication describes authentication of a user against a server by proving the possession of a shared secret. The authentication process in most cases is visible to the user and requires interaction.

For our approach, we focus on *Form-based Authentication*, where the server provides an HTML form with one or more input fields into which the user enters his credentials. Then the form including the user's input is sent to the server for validation. On client side, e.g., JavaScript could be utilized to provide protection of integrity, authenticity and/or confidentiality, before sending the data. On server side, any application logic designed to validate authentication data, can read and process the received data. No specific client side enhancements have to be performed to support this authentication type.

Our approach could also be used in conjunction with either *HTTP Basic Authentication* or *HTTP Digest Authentication* (both described in [FHBH+99]), but this would require several client side modifications.

3.3 Persistency of Authentication

HTTP is a stateless protocol. Without additional mechanisms, a user would be forced to re-enter his login information for each HTTP request. Cookies [Bar11] are used to transform stateless HTTP requests into stateful user sessions by explicitly linking them together. They are sent with every HTTP request from the browser to the web server. Cookies consist of name-value pair containing session information. A web server can instruct a browser to store a cookie by sending an additional HTTP header, embedded in an HTTP response, as follows:

```
Set-Cookie: SessionID=280-9757248-2350101;
            domain=docs.foo.com; path=/accounts;
            expires=Sun, 30 Mar 2015 05:23:00 GMT
```

A cookie can have further attributes, such as `domain` and `path` that define the scope of a cookie (e.g. `docs.foo.com/accounts`). The `expires` attribute indicates when a cookie expires.

Stored HTTP cookies are automatically sent back to the server by adding the additional HTTP header `Cookie` containing the set cookie(s). This simple mechanism is supported by all browsers. HTTP cookies are often used to store the result of an initial authentication; we will call such cookies Session Cookies.

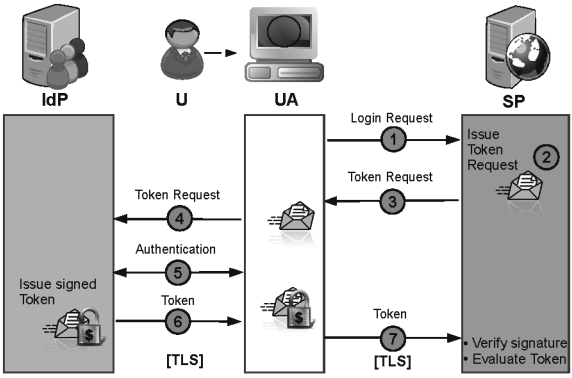

Figure 2: Single Sign-On overview.

3.4 Single Sign-On (SSO)

Single Sign-On (SSO) is a mechanism for identity and access management within a federation between related but independent parties. SSO allows a user to authenticate himself to a trusted party (called IdP) to get a security token. This token facilitates the access to a federated party (called SP) without any further login requests. Because an IdP can be federated with multiple SPs, SSO provides great usability and flexibility within the identity management and reduces the configuration and support costs.

Figure 2 shows an overview of an SSO login procedure: User **U** with a user agent **UA**, e.g. browser, tries to get restricted resources from a given SP (1). The SP generates a token request (2) and redirects the client to the IdP (3,4). In the following step, the user authenticates himself to the IdP (5) according to the supported authentication mechanisms (e.g. username/password, mutual authentication, two-factor authentication, and/or biometrics). As a result of the successful authentication, the security token is issued and sent through the user agent to the SP, where it can be verified (6,7). Please note the mandatory prerequisite of TLS secured communication channels between the participants.

4 Threat Model

For our security analysis, we assume an active attacker with full access to the communication network. Therefore, in addition to passively eavesdropping on the victim's communications, the attacker is capable of actively altering all messages sent or received by the victim and his communication partners (cf. [DY83]). We assume that the attacker is able to impersonate (e.g., by means of DNS and PKI spoofing) any server towards the victim.

For our *password-based login* approach, we assume that the victim shares a distinct secret (password) with each of his intended communication partners[2]. In the case of our *SSO* approach, we assume that the victim shares a secret (password) with the IdP, and the federation is set up such that every SP possesses the correct certificate and trusted public key of the IdP. Thus, the attacker is *not* able to impersonate the IdP towards any honest

[2]Please note that for security reasons instead of the password often only a salted hash value of the password is stored on server side.

SP.

We further assume, the attacker does *not* have direct access to the victim's device or actual servers (other than those possibly created by himself). This means the attacker cannot simply read out and steal the secrets shared between the victim and his communication partners, e.g., by utilizing a keylogger or manipulating the victim's browser's DOM. Similarly, we assume the victim is not prone to Phishing attacks intended to steal his password.

5 Technical Preconditions

In addition to the abilities and limitations of the attacker stated previously, several technical preconditions must be fulfilled for our approach to be feasible:

- The user's User Agent (UA) must provide an interface to access the first Finished message of each TLS connection.
- Client and server share a symmetric secret in form of a password. This password has been exchanged beforehand via a secure channel and is stored in a secure way.
- The optional TLS features *TLS session resumption* and *TLS renegotiation* must be disabled on client as well as server side. The usage of any of these features will automatically result in a Finished message differing from the one originally used for the binding. This will invalidate all previous channel bound authentication information.

6 TLS Unique

In this section we describe our solution by applying a secure cryptographic binding using TLS Unique to three practical use cases: (1) Password-Based Login, defined in Section 6.1. (2) HTTP cookies, described in Section 6.2. And finally (3) SSO, shown in Section 6.3. In this section, we also put these building blocks together and present a novel holistically secured browser-based SSO protocol.

6.1 Password-Based Login

As already discussed, authentication using passwords can be unsafe for many reasons. In order to strengthen the password-based login, we propose a novel approach binding the credentials to the TLS session. The protocol run is sketched in Figure 3.

(1.) The user requests a restricted resource on the Web server through his UA.
(2.) The server redirects the user to the authentication module via an HTTP Redirect. In addition, the HTTP Redirect message contains the HTTP-Header *tls-unique-auth: true*, which triggers the TLS Unique module in the UA.
(3.) The TLS channel between UA and server is established.
  3.1) The UA extracts the first Finished message fin via the RFC 5929 interface and stores it.
  3.2) The server acts accordingly on his end of the TLS connection and extracts and stores the first Finished message fin'.
  Note: If no MitM attack between UA and the server is active, it should hold that $fin == fin'$.
(4.) The user enters his *user_id* into the corresponding form field provided by the server.
(5.) The server provides the corresponding salt which was used to build the stored password hash on server side.
(6.) The user enters his password pw. Consequentially, the TLS Unique module in the

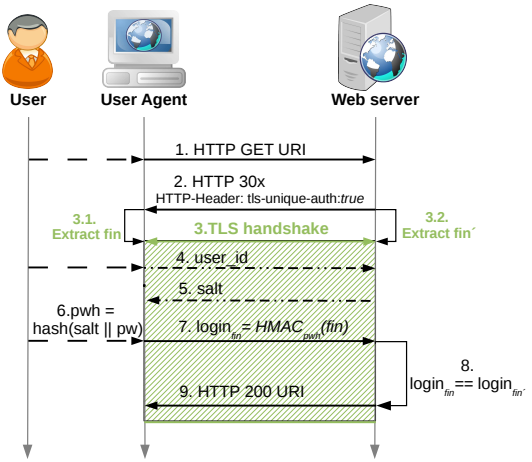

Figure 3: Password-Based Login bound to TLS Unique.

UA computes a hash value of the concatenation of the received salt and the user's provided password: $pwh = hash(salt||pw)$.

(7.) The TLS Unique module then computes an HMAC $login_{fin} = HMAC_{pwh}(fin)$, which is transmitted to the server.

(8.) After receiving this value the server computes $login_{fin'} = HMAC_{pwh}(fin')$. Only if $login_{fin} == login_{fin'}$, the server will accept the credentials and successfully authenticate the client.

(9.) The server grants access to the requested resource.

Note 1: Management of passwords on server side is out of scope of this paper. Thus, for the purpose of the protocol described here, usage of salts is optional (see steps 4 and 5). However, based on existing incidents, we highly recommend their usage within the authentication process.

Note 2: Using the TLS Unique binding during the login phase does not prevent the interception of the packages and their analysis by the given adversary controlling the network traffic. However, the $login_{fin}$ token contains only an HMAC over the related credentials and does not reveal the used password. Due to the TLS Unique binding, $login_{fin}$ can only be used in the TLS session uniquely identified by fin (resp. fin'); an adversary trying to use the stolen token in his own TLS session with the server will fail to authenticate himself as the victim because the Finished message fin'' on server side will not match the token.

Note 3: The security of $login_{fin}$ against brute-force attacks depends on the complexity of the chosen password. By this means, the password should have high entropy.

6.2 HTTP Session Cookies

Next, we extend the TLS Unique binding of the password based login protocol from Section 6.1 to HTTP cookies to maintain the authentication state in subsequent HTTP requests. Since session cookies are set and read by the server, a cookie binding to the TLS session has to take place at the server side of the communication.

We propose the following procedure to bind the HTTP session cookie to the TLS channel:

(1.) **Secure user authentication:** The user authenticates using the TLS Unique Password-Based Login protocol as described above.

(2.) **Set TLS Unique cookie:** After successful authentication, the server immediately sets an HTTP cookie $cky_{bound} = hash(fin')$ with the value of the Finished message of the TLS session used in step 1. The value of this cookie serves as a session identifier on server side[3].

(3.) **Verifying the Session cookie:** In all subsequent HTTP requests the browser automatically provides the cky_{bound} by adding it to the HTTP header of the request. The server accepts the cookie if the following comparison holds: $cky_{bound} == hash(fin')$. Thus, the server verifies that the session cookie belongs to the corresponding TLS channel.

Note: Even if the attacker is able to steal the session cookies of the user, he cannot establish a parallel TLS session to the server with the same Finished message $fin == fin'$. As a result, using the stolen cookies to authenticate through a different TLS channel is not possible.

6.3 Single Sign-On (SSO)

A major problem of SSO protocols are MitM attacks on involved TLS connections [SB12]. If TLS is badly configured, or if the adversary succeeds to mount a DNS/PKI spoofing attack, he may read authentication credentials from the network. The TLS Unique binding from RFC 5929 offers a way for the authenticating party to check if there is a malicious MitM on the network. A usage scenario for a TLS Unique binding within general SSO protocols is sketched in Figure 4.

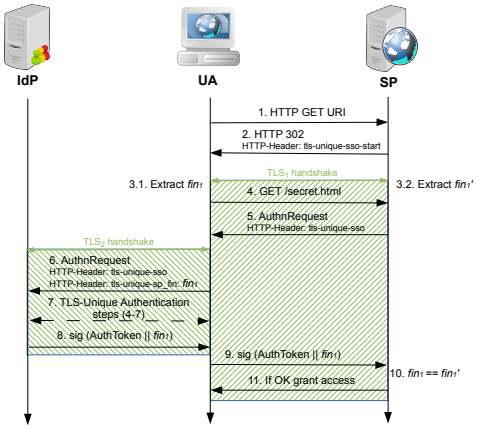

Figure 4: TLS-unique usage scenario in SSO.

In an SSO protocol, typically at least two TLS sessions are involved: TLS_1 between UA and SP, and TLS_2 between UA and IdP. Often the first request to SP is not protected by

[3] Additional cookie properties (e.g., confidentiality or integrity protection) can also be applied in the sense of [LKHG05].

TLS, but in order to be able to apply the TLS-unique binding, setting up a TLS channel immediately in response to Step 2 is essential: After receiving the authentication request from SP, the User Agent (UA) extracts the first Finished message fin_1 (Step 3.1) from TLS_1, using the API described in RFC 5929. This value is added as a parameter to the authentication request and forwarded to the IdP in Step 6.

Note: Technically, SP could immediately include the corresponding value fin'_1 into the authentication request in Step 5. However, it is important that this value is included on client side, as fin_1 is intended to be checked against fin'_1 in Step 10 by SP.

The link between UA and IdP can also be protected by TLS Unique: If fin_2 is included into the authentication process in Step 7 (e.g., like described in Section 6.1), IdP can check whether an MitM is present or not. Alternatively, authentication methods may range from password based authentication as described to arbitrary cryptographic protocols [JKSS10].

After successful authentication of the user, IdP will include fin_1 into the issued authentication token (Step 8). This token is cryptographically secured – either by a digital signature or by an HMAC. Thus, the content of the token cannot be altered by an MitM.

In Step 9, the UA forwards the token to the SP through the previously opened session TLS_1.

Note: TLS_1 has to be kept active between UA and SP all the time. This is important, so the Finished message of the current TLS connection is still the same as was extracted as fin'_1 in Step 3.2. The TLS connection can, e.g., be preserved by sending keep-alives on the TCP connection below.

When SP receives the authentication token in Step 10, he compares fin'_1 to the value fin_1 read from the token. Access is granted in Step 11 if and only if both the authentication token is valid and $fin_1 == fin'_1$.

Based on the modification, the protocol can resist network based attacks, e.g. MitM. In this case, an attacker can act undetected until after Step 9 in Figure 4, because SP only sees a TLS connection with an anonymous browser up until then, and the MitM has tricked the UA into accepting a TLS connection with him.

Nevertheless, the Finished messages for the TLS sessions between UA and MitM and between MitM and SP differ from each other. Even if the attacker is able to intercept the authentication token during Step 9, he is not able to redeem it, because the token will be issued for $fin_{UA,MitM}$, whereas the MitM attacker is connected to SP via a TLS session described by $fin_{MitM,SP}$.

7 TLS Unique Implementation

This section provides an overview of necessary changes to client software and required Application Programming Interfaces (API) for successful integration of the proposed concept. As the solution has to be implemented at client side, it is necessary to modify the user's client software, such as the browser. For convenience, the solution can be provided as a browser plugin, integrating seamlessly and without user interaction. The plugin performs all required steps and only becomes visible in case of failures.

Additional Header-Fields To simplify HTTP message processing, it is beneficial to introduce a new HTTP Headers:

- `tls-unique-auth:` {`boolean`} – initiates the password-based login via TLS Unique.
- `tls-unique-sso:` {`boolean`} – initiates SSO via TLS Unique.
- `tls-unique-sp_fin:` {`char[]`} – contains the value of the *Finished* message.

The integrated browser plugin only activates if this particular HTTP Header-Field is encountered, otherwise it remains inactive as a background process.

API Implementation Unfortunately, the plugin requires additional APIs for access to lower SSL/TLS functionality - *the SSL/TLS PreMasterSecret, MasterSecret and derived key material MUST be kept secret and unaccessible*. We propose an API that provides low-level access to all SSL/TLS handshake messages - *this MUST only include access to raw messages, as they are supposed to be sent over the network, explicitly excluding outputs of decryption processes and access to internal states*.

With direct access to the necessary TLS messages, it is possible to extract the information needed for channel binding (eg. encrypted Finished message).

8 Conclusion

Current authentication mechanisms are prone to a variety of attacks, e.g., cookie theft or MitM. This way, an attacker may steal credentials or authentication tokens and use them to impersonate the victim.

The introduced TLS Unique approach holistically secures the whole login and/or SSO protocol flow by binding the authentication information to specific TLS sessions. We use the first Finished message to uniquely identify a TLS session and then cryptographically bind this identifier to the authentication data. Our approach covers the transport of credentials (e.g. a username/password combination), session cookies, and SSO authentication tokens. By using this approach, the exploit of a large number of previously found authentication flaws can be prevented.

TLS Unique can be implemented as a browser plugin, provided that the corresponding API is available to allow access to the required parameters (especially the first Finished message) from the TLS handshake. A few new HTTP headers have to be provided by the server to activate and trigger the plugin. Our solution describes a generic approach which can be easily adapted to other SSO protocols (e.g. OAuth and OpenID).

References

[AWZ10] J. Altman, N. Williams, and L. Zhu. Channel Bindings for TLS. RFC 5929 (Proposed Standard), July 2010.

[Bar11] A. Barth. The Web Origin Concept. IETF, RFC 6454, December 2011. `http://tools.ietf.org/html/rfc6454`.

[BDLF+14] Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Alfredo Pironti, and Pierre-Yves Strub. Breaking and Fixing Authentication over TLS. 2014.

[BH12] D. Balfanz and R. Hamilton. Transport Layer Security (TLS) Channel IDs. Internet-Draft, November 2012.

[Can13] Scott Cantor. SAML V2.0 Channel Binding Extensions Version 1.0, 2013. `http://docs.oasis-open.org/security/saml/Post2.0/saml-`

channel-binding-ext/v1.0/cs01/samlchannel-binding-ext-v1.0-cs01.html.

[DCBW12] Michael Dietz, Alexei Czeskis, Dirk Balfanz, and Dan S. Wallach. Origin-bound certificates: a fresh approach to strong client authentication for the web. In *Proceedings of the 21st USENIX conference on Security symposium*, Security'12, pages 16–16, Berkeley, CA, USA, 2012. USENIX Association.

[DR08] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. Updated by RFC 5746.

[DY83] D. Dolev and A. Yao. On the security of public key protocols. *Information Theory, IEEE Transactions on*, 29(2):198–208, March 1983.

[FHBH+99] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. HTTP Authentication: Basic and Digest Access Authentication. RFC 2617 (Draft Standard), June 1999.

[FSSF01] Kevin Fu, Emil Sit, Kendra Smith, and Nick Feamster. Dos and Don'ts of Client Authentication on the Web. In *Proceedings of the 10th conference on USENIX Security Symposium - Volume 10*, SSYM'01, pages 19–19, Berkeley, CA, USA, 2001. USENIX Association.

[GJMS08] Sebastian Gajek, Tibor Jager, Mark Manulis, and Jörg Schwenk. A Browser-based Kerberos Authentication Scheme. In Sushil Jajodia and Javier López, editors, *Computer Security - ESORICS 2008, 13th European Symposium on Research in Computer Security, Málaga, Spain, October 6-8, 2008. Proceedings*, volume 5283 of *Lecture Notes in Computer Science*, pages 115–129. Springer, August 2008.

[JKSS10] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. Generic Compilers for Authenticated Key Exchange. In *ASIACRYPT*, 2010.

[KS10] Nate Klingenstein and Tom Scavo. SAML V2.0 Holder-of-Key Web Browser SSO Profile Version 1.0: Committee Specification 02. "http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-holder-of-key-browser-sso-cs-02.pdf", August 2010.

[KSJG10] Florian Kohlar, Jörg Schwenk, Meiko Jensen, and Sebastian Gajek. Secure Bindings of SAML Assertions to TLS Sessions. In *ARES*, pages 62–69, 2010.

[KSTW07] Chris K. Karlof, Umesh Shankar, Doug Tygar, and David Wagner. Dynamic pharming attacks and the locked same-origin policies for web browsers. Technical Report UCB/EECS-2007-52, EECS Department, University of California, Berkeley, May 2007.

[LKHG05] Alex X. Liu, Jason M. Kovacs, Chin-Tser Huang, and Mohamed G. Gouda. A Secure Cookie Protocol. In *Proceedings of the 14th IEEE International Conference on Computer Communications and Networks*, pages 333–338, San Diego, California, October 2005.

[Mar09] Moxie Marlinspike. More Tricks For Defeating SSL In Practice. *Black Hat USA*, 2009.

[MBS07] Chris Masone, Kwang-Hyun Baek, and Sean Smith. WSKE: web server key enabled cookies. In *Proceedings of the 11th International Conference on Financial cryptography and 1st International conference on Usable Security*, FC'07/USEC'07, pages 294–306, Berlin, Heidelberg, 2007. Springer-Verlag.

[PS00] Joon S. Park and Ravi S. Sandhu. Secure Cookies on the Web. *IEEE Internet Computing*, 4(4):36–44, 2000.

[SB12] San-Tsai Sun and Konstantin Beznosov. The devil is in the (implementation) details: an empirical analysis of OAuth SSO systems. In *Proceedings of the 2012 ACM conference on Computer and communications security*, CCS '12, pages 378–390, New York, NY, USA, 2012. ACM.