On the Security of Holder-of-Key Single Sign-On

Andreas Mayer[1], Vladislav Mladenov[2], and Jörg Schwenk[2]

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

Abstract: Web Single Sign-On (SSO) is a valuable point of attack because it provides access to multiple resources once a user has initially authenticated. Therefore, the security of Web SSO is crucial. In this context, the SAML-based Holder-of-Key (HoK) SSO Profile is a cryptographically strong authentication protocol that is used in highly critical scenarios. We show that HoK is susceptible to a previously published attack by Armando et al. [ACC+11] that combines logical flaws with cross-site scripting. To fix this vulnerability, we propose to enhance HoK and call our novel approach HoK+. We have implemented HoK+ in the popular open source framework SimpleSAMLphp.

1 Introduction

Today's Internet users are forced to register to each website individually and have to manage a plethora of accounts and passwords as part of their daily job. This aspect is not only cumbersome but also seriously insecure, as users frequently choose weak (easy to remember) passwords and/or reuse them on several websites. Furthermore, each website has to reinvent the wheel by building and operating another stand-alone authentication solution and thus suffers from high user management costs.

Web Single Sign-On (SSO), as a subset of identity and access management, was proposed to tackle the described usability, security, and management issues. With SSO, a user authenticates *once* to a trusted third party, called Identity Provider (IdP), and subsequently gains access to all federated websites (i.e. Service Providers) he/she is entitled to – without being prompted with another login dialog.

Nowadays, Web SSO solutions are wide-spread and their importance still continues to grow. In this context, the Security Assertion Markup Language (SAML) [S. 05] is a flexible and open XML standard for exchanging authentication and authorization statements. Since its invention in 2001, SAML has become the dominant technology for enterprise Web SSO. SAML is also used in research, education, and e-Government scenarios.

In security critical use cases, SAML Holder-of-Key (HoK) Web SSO [KS10] is applied because it fulfills the highest "level of assurance" as defined by the National Institute of Standards (NIST) [The11]. HoK adds strong cryptographic guarantees to the authentication context and enhances the security of SAML assertion and message exchange by using mutual authenticated secure channels. It builds on the TLS protocol which is ubiquitously implemented in all major browsers (including mobile browsers) and web servers.

Therefore, maximum compatibility to existing infrastructure and deployments is given.

CONTRIBUTION. Although, HoK successfully defends against a wide range of attacks, like man-in-the middle (MITM) and man-in-the-browser (MITB), we show that it is still susceptible to a previously discovered attack by Armando et al. [ACC+11]. To mitigate this vulnerability, we propose to extend HoK and use the strong cryptographic binding in a more holistic approach. We call this countermeasure HoK+. In order to demonstrate the practical feasibility, we have implemented a proof-of-concept in the popular open source framework SimpleSAMLphp [Sim13].

OUTLINE. Related work is given in Section 2. The following section will introduce Web SSO, SAML, and cookie theft. HoK Web SSO is explained in Section 4. The RelayState Spoofing attack applied on HoK is presented in Section 5. Our countermeasure HoK+ along with the implementation is presented in Section 6. We conclude in Section 7.

2 Related Work

SINGLE SIGN-ON. The Security Assertion Markup Language (SAML) was developed for the secure exchange of XML-based messages and is mostly applied within federated identity management. The most widespread field of use of SAML is Web SSO. Unfortunately, a diversity of attacks have been discovered in the last years.

In 2003, Groß [Gro03] disclosed several adaptive attacks on the SAML Browser Artifact Profile resulting in the interception of the authentication token contained in the URL. Additionally, Groß analyzed the revisited version of SAML [S. 05], finding further logical flaws and security threats [GP06]. Related vulnerabilities have been analyzed and found in the Liberty Single Sign-On by Pfitzmann and Waidner [PW03]. In 2006 Y. Chan introduced a new parallel session attack to bypass all levels of authentication by exclusively breaking the weakest one among them [Cha06].

In 2008, Armando et al. [ACC+08] built a formal model of the SAML V2.0 Web Browser SSO protocol and analyzed it with the model checker SATMC. The practical evaluation revealed an existing security issue on the SAML interface of Google, allowing a malicious SP to impersonate any user at any Google application. Later on, same authors identified another attack on Google's SAML interface [ACC+11]. They manipulated the `RelayState` parameter in the query string of an HTTP request in order to exploit an existing cross-site scripting (XSS) vulnerability on Google. In this paper, we apply the RelayState Spoofing attack on HoK Web SSO.

Further researches on the security of Web SSO have shown serious flaws resulting in identity theft and compromising the security of the end-systems. In [SMS+12] the authors published an in-depth analysis of XML Signature. Additionally, they introduced novel techniques to manipulate digitally signed messages, despite the applied integrity protection mechanisms. As a result the authors of the study examined 14 major SAML frameworks and showed that 11 of them had critical XML Signature wrapping flaws allowing the impersonation of any user. Another study regarding the security of SSO systems has been published in 2012 by Wang et al. [WCW12]. The authors examined REST-based

authentication protocols like OpenID and found serious logic and implementation flaws resulting in identity theft.

TLS CHANNEL BINDINGS. In 2009 OASIS[1] standardized the *SAML Holder-of-Key Web Browser SSO Profile* [KS10] using TLS client certificates for strengthening the authentication process. Later on, RFC 5929 [AWZ10] has been published in 2010 and describes three different channel binding types for TLS without using any client certificates.

In 2012, Dietz et al. [DCBW12] have 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. However, their idea requires changes in the TLS protocol, thus all current TLS implementations must be modified.

Recently, Google introduced another TLS extension called *Channel ID* [BH12]. Again, fundamental changes to underlying TLS implementations are required. 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 freshly 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* [Can13] that allows the use of channel bindings in conjunction with SAML. In this manner, the channel bindings of RFC 5929 [AWZ10] can be integrated in all SAML related services (e.g. SSO).

3 Foundations

WEB SINGLE SIGN-ON. In an SSO flow (cf. Figure 1) user U navigates user agent UA (e.g. a browser) and tries to access a restricted resource on SP (1). Thus the user is not authenticated yet, SP generates a token request (2) and redirects UA with the token request to IdP (3,4). In the following step U authenticates himself to IdP (5) according to the supported authentication mechanisms. Subsequently, the security token is issued and sent through UA to SP, where the integrity and authenticity is verified and the content is evaluated (6,7).

SAML. The Security Assertion Markup Language (SAML) [S. 05] is a widely-used open XML standard for exchanging authentication and authorization statements about *subjects*. These statements are contained in security tokens called *assertions*. SAML consists of three other building blocks: (1) *protocols* – define how assertions are exchanged between the actors; (2) *bindings* – specify how to embed assertions into transport protocols (e.g. HTTP or SOAP), and (3) *profiles* define the interplay of assertions, protocols, and bindings that are necessary for the needs of a specific use case to be met. The investigated SAML HoK Web Browser SSO Profile [KS10] is such an application scenario.

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

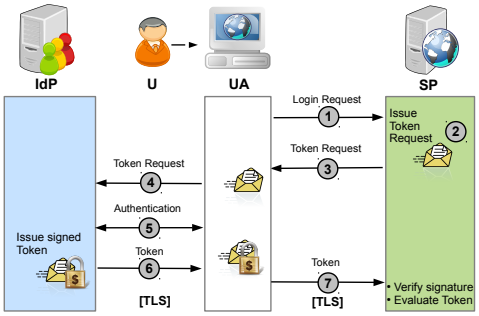

Figure 1: Single Sign-On Overview.

In order to request an assertion, SAML defines the `<AuthnRequest>` XML message (i.e. token request). Important elements and attributes of this message are: `ID` – a unique and randomly chosen request identifier; `AssertionConsumerServiceURL` (ACS_{URL}) – specifies the endpoint to which IdP must deliver the assertion; `<Issuer>` – the EntityID of SP.

The issued assertion (i.e. security token) contains the following elements and parameters relevant to this paper: `InResponseTo` – must match the `ID` of the `<AuthnRequest>`; `ID` – a unique and randomly chosen assertion identifier; `<Issuer>` – the EntityID of IdP; `<Audience>` – the EntityID of SP; `<SubjectConfirmation>` – the client certificate of UA; `<Subject>` – the EntityID of U;. To assure the integrity and authenticity of the security claims made, the whole assertion *must* be protected by a digital signature (`<Signature>`) which is compliant to the XML Signature standard [ERS+08].

COOKIE THEFT. The Hypertext Transfer Protocol (HTTP) [FGM+99] is a wide-spread web application protocol transmitting messages between user's browser and web server. Since HTTP is a stateless protocol, without additional mechanisms a user will be forced to re-enter his login information repeatedly, for every HTTP request. HTTP session cookies [Bar11] are applied to solve this problem by making HTTP stateful. In other words, the cookies are a mechanism to make authentication persistent. They are set by the web server, stored in the client's browser and transmitted with every HTTP request to the web server. In this manner a user has to authenticate himself only once in order to get a cookie which represents the session state of an authenticated user.

Unfortunately, HTTP cookies can be stolen by various attacks like eavesdropping the network communication, cross-site scripting (XSS), cross-site request forgery (CSRF), and UI redressing. The theft results in impersonation of the victim by the adversary. To impede cookie theft, two cookie flags are applied: `secure` – defines that cookies are only sent over a secure channel (i.e. TLS); `HTTPOnly` – makes a cookie inaccessible by client-side scripts (e.g. JavaScript). Unfortunately, all existing best-practice cookie theft countermeasures can be bypassed in several ways [Man03, Pal07, BBC11, Hei12].

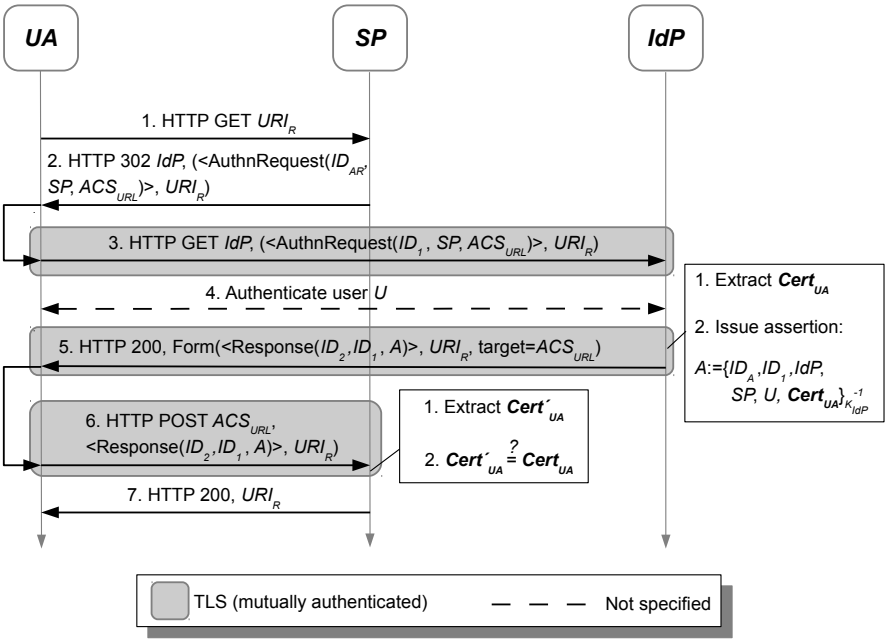

Figure 2: SAML 2.0 Holder-of-Key Web Browser SSO Profile.

4 Holder-of-Key Web SSO

The *SAML V2.0 Holder-of-Key (HoK) Web Browser SSO Profile* [KS10] is an OASIS standard based on the browser-based Kerberos scheme `BBKerberos` [GJMS08]. By using mutual authenticated secure channels, HoK adds strong cryptographic binding to the authentication context and enhances the security of the message exchange, i.e. the SAML assertion. HoK builds on the TLS protocol which is ubiquitously implemented in all major browsers (including mobile browsers) and web servers. Therefore, maximum compatibility to existing infrastructures and deployments is given.

In HoK the web server recognizes the browser on basis of a unique (self-signed) client certificate. The browser proves the possession of the client certificate's private key in a mutual authenticated TLS handshake. Any self-signed certificate is sufficient, as neither IdP nor SP are required to validate the trust chain of the certificate. Therefore, no complex and expensive public-key infrastructure (PKI) is needed. The issued assertion is cryptographically bound to the client certificate by including either the certificate itself or a hash of it in the signed assertion. In this manner the assertion can be used only in conjunction with the according private key stored in user's browser.

Figure 2 illustrates the detailed flow of the SAML Holder-of-Key Web Browser SSO Profile. Subsequently, we describe the individual steps:

1. $UA \rightarrow SP$**:** User U navigates its user agent UA to SP and requests a restricted resource R by accessing URI_R.[2] This starts a new SSO protocol run.
2. $SP \rightarrow UA$**:** SP determines that no valid security context (i.e. an active login session) exists. Accordingly, SP issues an authentication request <AuthnRequest(ID_1, SP, ACS_{URL})> and sends it Base64-encoded, along with the RelayState parameter URI_R, as an HTTP 302 (redirect to IdP) to UA. ID_1 is a fresh random string and SP the identifier of the Service Provider. ACS_{URL} specifies the endpoint to which the assertion must be delivered by IdP.
3. $UA \rightarrow IdP$**:** Triggered by the HTTP redirect, a mutually authenticated TLS connection is established between UA and IdP. Thereby, IdP and UA send to each other their certificates and each proves possession of the corresponding private key within the mutual TLS handshake. Afterwards, the built TLS channel is used to transport <AuthnRequest(ID_1, SP, ACS_{URL})>, along with URI_R, to IdP.
4. $UA \leftrightarrow IdP$**:** If the user is not yet authenticated, IdP identifies U by an arbitrary authentication mechanism.
5. $IdP \rightarrow UA$**:** IdP creates an assertion $A := (ID_A, ID_1, IdP, SP, U, Cert_{UA})$, including the unique identifier ID_A and ID_1 from the request, the entity IDs of IdP, SP, the user identity U, and the user certificate $Cert_{UA}$. Subsequently, A is signed with the IdP's private key K^{-1}_{IdP}. The signed assertion A is embedded into a <Response> message, together with ID_1 and the fresh response identifier ID_2, and is sent Base64-encoded in an HTML form, along with the RelayState=URI_R, to UA.
6. $UA \rightarrow SP$**:** A small JavaScript embedded in the HTML form triggers the forwarding of the assertion A to ACS_{URL} via HTTP POST. Simultaneously, a mutually authenticated TLS channel with SP is built, where UA presents a client certificate $Cert'_{UA}$.
7. $SP \rightarrow UA$**:** SP consumes A, and requires that ID_1 is included as InResponseTo attribute in the assertion. A is only valid if the contained $Cert_{UA}$ is equal to $Cert'_{UA}$ from the previous TLS handshake (step 6). SP verifies the XML signature, and authenticates user U resulting in a security context. Finally, SP grants U access to the protected resource R by redirecting U to URI_R.

HoK does not prevent assertion theft in any circumstance (e.g. via XSS). However, stolen assertions are *always* worthless for the adversary, since they are cryptographically bound to the legitimate browser. To successfully attack HoK, the adversary needs knowledge of the private key belonging to the used client certificate. Consequently, the private key is protected by the browser and/or by the underlying operating system. It is even possible to store the private key on a secure device (e.g. smart card) to protect against malware in untrusted environments (e.g. in kiosk scenarios, where computers are accessible to everyone at public places). Furthermore, HoK protects both TLS connections (between UA and SP as well as between UA and IdP) against MITM and MITB attacks. It is important to note that the presentation of a client certificate in step 1 and 2 (i.e. a mutually authenticated TLS handshake) is *strictly* optional [KS10, p.10].

[2] URI_R is called *RelayState* as it preserves and conveys the initial step of a SSO protocol run (i.e. the URI of the accessed resource R).

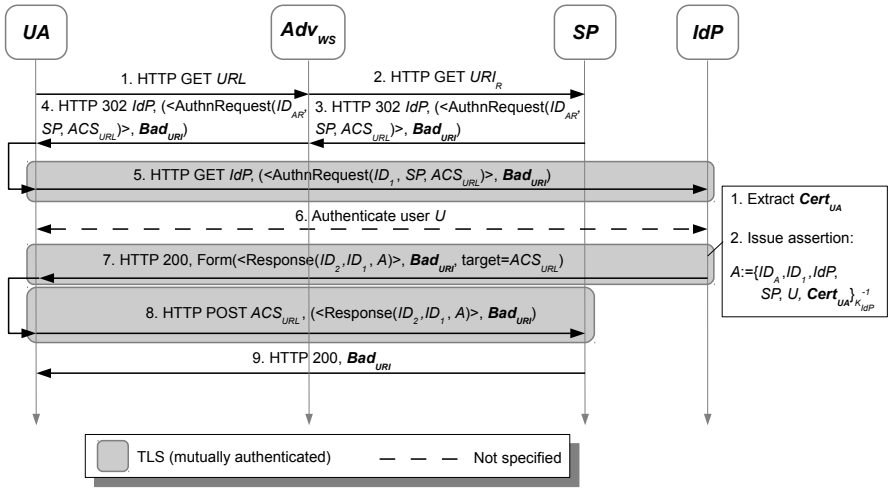

Figure 3: RelayState Spoofing attack on HoK Web SSO.

5 RelayState Spoofing Attack

Armando et al. [ACC+11] have discovered a serious authentication flaw (RelayState Spoofing) in standard Web SSO. In this section, we provide a review of this attack and show that this technique breaks the security of SAML HoK Web SSO.

5.1 Threat Model

We make two assumptions on adversary Adv to launch a successful RelayState Spoofing attack:

1. Adv is able to lure the victim to a malicious website (Adv_{ws}) controlled by him. Since there is no need to read the network traffic, we may assume that UA of the victim always communicates over encrypted TLS channels. Moreover, the victim may only accept communication partners with valid and trusted server certificates.
2. Adv requires an XSS vulnerability for each attacked SP that allows cookie theft.

5.2 Attack Description

The RelayState Spoofing attack combines a logical flaw in the SSO implementation (the `RelayState` parameter URI_R can be changed by Adv, and this parameter will be used in a final redirect triggered by SP) with implementation bugs at the Service Provider SP (an XSS attack can be launched through an HTTP redirect query string parameter). The attack flow is depicted in Figure 3.

In step 2 or 4, Adv injects an XSS attack vector into the parameters of the RelayState $URI_R = BAD_{URI}$. After successful authentication at the honest SP (i.e. after successful verification of the SAML assertion), the maliciously-crafted $URI_R = BAD_{URI}$ is loaded by a browser redirect (step 9), and the XSS attack is automatically executed in the browser resulting in a cookie theft.

Two preconditions must be met for this attack to be successful: (1) injectability of XSS code into URI_R and (2) XSS-vulnerable implementations of SPs. The first precondition normally holds in SAML-based SSO scenarios, because URI_R is not part of the XML-based data structures authentication request or assertion, and can thus not be integrity protected by an XML signature. Instead, the SAML standard recommends to protect the integrity of URI_R by a separate signature ([CHK+05], Section 3.4.3). However, [ACC+11] and our own investigations show that this is normally not the case in practice. The second precondition has been shown to be applicable by [WCW12] and [SB12], where numerous implementation bugs for SPs have been documented. Additionally, Armando et al. have presented two successful RelayState Spoofing attacks on Novel Access Manager 3.1 and Google Apps.

By misusing the SSO protocol flow, Adv can ensure that the victim has an authenticated session with the attacked SP, which is a precondition for cookie theft via XSS. Furthermore, Adv can use this attack as launching pad to automatically execute cross-site request forgery (CSRF) [The13] attacks.

An attack related to RelayState Spoofing is *login CSRF* [BJM08], whereby Adv forges a cross-site request to the honest website's login form, resulting in logging the victim into this website as the adversary. This CSRF variant is also applicable to SAML-based SSO.

6 HoK+ Web SSO

Although SAML HoK Web SSO protects against a variety of attacks, it is still susceptible to the RelayState Spoofing attack as shown in the previous section. This is due to the fact, that HoK does not protect the SP's `<AuthnRequest>` against a MITM attack.

To additionally mitigate this severe attack, we propose to enhance HoK and call our novel approach *HoK+*. In summary, HoK+ additionally binds the SP's `<AuthnRequest>` message to the client certificate. Therefore, the *whole* SSO protocol flow is cryptographically linked to the legitimate UA.

6.1 HoK+ Protocol

Figure 4 illustrates the detailed flow of HoK+, which consists of the following steps:

1. $UA \rightarrow SP$**:** User U navigates its user agent UA to SP and requests a restricted resource R by accessing URI_R. This starts a new SSO protocol run. A mutually authenticated TLS connection is established between UA and SP and thereby UA sends its client certificate $Cert_{UA}$ to SP.
2. $SP \rightarrow UA$**:** SP extracts $Cert_{UA}$ from the TLS handshake and issues an authentica-

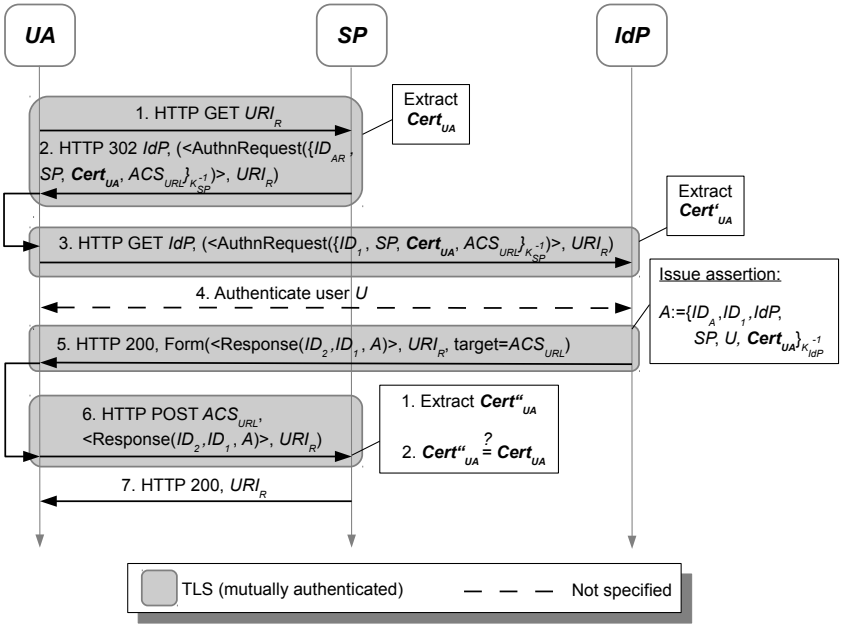

Figure 4: The novel HoK+ SSO Profile.

tion request <AuthnRequest (ID_1, SP, $Cert_{UA}$, ACS_{URL})>, which is then signed with the SP's private key K_{SP}^{-1}. The <AuthnRequest> is sent back to UA, along with URI_R, as HTTP redirect to IdP.

3. $UA \rightarrow IdP$**:** Triggered by the HTTP redirect, a mutual authenticated TLS connection between UA and IdP is established. UA uses this TLS connection to transport <AuthnRequest>, along with URI_R, to IdP.
4. $UA \leftrightarrow IdP$**:** IdP verifies the XML signature of the received <AuthnRequest> with SP's public key and then compares $Cert_{UA}$ from the authentication request with $Cert'_{UA}$ of the TLS connection. If they match, IdP authenticates U with an arbitrary method. Otherwise, the protocol is stopped.
5. $IdP \rightarrow UA$**:** IdP creates an assertion $A := (ID_A, ID_1, IdP, SP, U, Cert_{UA})$. Subsequently, A is signed with the IdP's private key K_{IdP}^{-1} and is embedded into a <Response> message, together with ID_1 and the fresh response identifier ID_2. Afterwards it is sent Base64-encoded in an HTML form, along with the RelayState=URI_R, to UA.
6. $UA \rightarrow SP$**:** A small JavaScript embedded in the HTML form triggers the forwarding of the assertion A to ACS_{URL} via HTTP POST. Simultaneously, a mutually authenticated TLS channel with SP is built, where UA presents a client certificate $Cert''_{UA}$.
7. $SP \rightarrow UA$**:** SP consumes A, and requires that ID_1 is included as InResponseTo attribute in the response message and in the assertion. Additionally, A is only valid if the contained $Cert_{UA}$ is equal to $Cert''_{UA}$ from the previous TLS handshake (step 6). SP verifies the XML signature, and authenticates U resulting in a security context. Finally, SP grants U access to the protected resource R by redirecting U to URI_R.

The reason why HoK+ mitigates the RelayState Spoofing attack is that no SAML assertion will be issued by IdP in case of an attack, since the authentication request is bound to the client certificate used by the adversary Adv. Thus we make it impossible for Adv to submit a valid SAML assertion to SP.

6.2 Implementation

In order to demonstrate the feasibility of HoK+, we have implemented it in the popular open source framework SimpleSAMLphp (SSP) [Sim13]. We have chosen SSP because it is known as a fairly secure framework [SMS+12], and because our own penetration tests and source code observations have shown that SSP did not reveal any XSS flaws. Moreover, SSP supports defense-in-depth techniques like `HTTPOnly` and `secure` flag cookies by default.

SSP already supports HoK [MS11]. We added code to create, process, and verify signed HoK+ authentication requests. The enhanced `<AuthnRequest>` is compliant with the SAML standard and conforming to the SAML V2.0 XML schema. The TLS client certificate is added in the same way as in the HoK SAML assertion: a `<SubjectConfirmation>` element, whose `Method` attribute is set to `holder-of-key:SSO:browser`, contains the Base64 encoded client certificate from the TLS channel. The `<SubjectConfirmation>` is inserted into the authentication request's `<Subject>` element. Due to the XML signature and the additional XML elements, the resulting HoK+ `<AuthnRequest>` messages are bigger than 2,048 bytes. Therefore, we had to change the HTTP redirect binding (i.e. transfer by HTTP GET parameter) to HTTP POST binding.

A total of 113 modified or added lines across 3 files in the SSP source code were required for these SSP modifications. Additionally, no further changes on UA were required.

7 Conclusion

Developing a secure Web SSO protocol is a nontrivial task. Due to the more complex trilateral communication flow between all participants, the attack surface is larger and more often leads to security issues compared to standard authentication. Despite the provided protection mechanisms to strengthen the authentication process, a plethora of attacks still exists.

The introduced HoK is a secure and robust Web SSO protocol that already protects against a variety of attacks (e.g. MITM). However, based on previously found authentication flaws, we showed that even the cryptographically strong security, provided by HoK, can be bypassed and thus cannot mitigate identity theft.

Our improved HoK+ approach *holistically* secures the whole SSO protocol flow and additionally mitigates RelayState Spoofing without the need of changing existing Web infrastructure (i.e. TLS, browser, and web server). Finally, the concept of HoK+ is generic and can be applied to other SSO protocols (e.g. OAuth and OpenID).

References

[ACC+08] Alessandro Armando, Roberto Carbone, Luca Compagna, Jorge Cuéllar, and M. Llanos Tobarra. Formal Analysis of SAML 2.0 Web Browser Single Sign-On: Breaking the SAML-based Single Sign-On for Google Apps. In *Proceedings of the 6th ACM Workshop on Formal Methods in Security Engineering, FMSE 2008*, pages 1–10, Alexandria and VA and USA, 2008. ACM.

[ACC+11] Alessandro Armando, Roberto Carbone, Luca Compagna, Jorge Cuéllar, Giancarlo Pellegrino, and Alessandro Sorniotti. From Multiple Credentials to Browser-Based Single Sign-On: Are We More Secure? In Jan Camenisch, Simone Fischer-Hübner, Yuko Murayama, Armand Portmann, and Carlos Rieder, editors, *SEC*, volume 354 of *IFIP Advances in Information and Communication Technology*, pages 68–79. Springer, 2011.

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

[Bar11] A. Barth. HTTP State Management Mechanism. RFC 6265 (Proposed Standard), April 2011.

[BBC11] Andrew Bortz, Adam Barth, and Alexei Czeskis. Origin Cookies: Session Integrity for Web Applications. In *W2SP: Web 2.0 Security and Privacy Workshop 2011*, May 2011.

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

[BJM08] Adam Barth, Collin Jackson, and John C. Mitchell. Robust Defenses for Cross-Site Request Forgery. In *Proceedings of the 15th ACM conference on Computer and communications security*, CCS '08, pages 75–88, New York and NY and USA, 2008. ACM.

[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`.

[Cha06] Yuen-Yan Chan. Weakest Link Attack on Single Sign-On and Its Case in SAML V2.0 Web SSO. In *Computational Science and Its Applications - ICCSA 2006*, volume 3982 of *Lecture Notes in Computer Science*, pages 507–516. Springer Berlin Heidelberg, 2006.

[CHK+05] Scott Cantor, Frederick Hirsch, John Kemp, Rob Philpott, and Eve Maler. Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0. OASIS Standard, 15.03.2005, 2005. `http://docs.oasis-open.org/security/saml/v2.0/saml-bindings-2.0-os.pdf`.

[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.

[ERS+08] Donald Eastlake, Joseph Reagle, David Solo, Frederick Hirsch, and Thomas Roessler. XML Signature Syntax and Processing (Second Edition), 2008. `http://www.w3.org/TR/xmldsig-core/`.

[FGM+99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. RFC 2616, Hypertext Transfer Protocol – HTTP/1.1, 1999.

[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.

[GP06] Thomas Groß and Birgit Pfitzmann. SAML artifact information flow revisited. Research Report RZ 3643 (99653), IBM Research, 2006. `http://www.zurich.ibm.com/security/publications/2006.html`.

[Gro03] T. Groß. Security analysis of the SAML Single Sign-on Browser/Artifact profile. In *Annual Computer Security Applications Conference*. IEEE Computer Society, 2003.

[Hei12] Mario Heiderich. *Towards Elimination of XSS Attacks with a Trusted and Capability Controlled DOM*. PhD thesis, Ruhr-University Bochum, Bochum, May 2012.

[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.

[Man03] Art Manion. Vulnerability Note VU#867593, 2003. `http://www.kb.cert.org/vuls/id/867593`.

[MS11] Andreas Mayer and Jörg Schwenk. Sicheres Single Sign-On mit dem SAML Holder-of-Key Web Browser SSO Profile und SimpleSAMLphp. In Bundesamt für Sicherheit in der Informationstechnik, editor, *Sicher in die digitale Welt von morgen*, pages 33–46, Gau-Algesheim, May 2011. SecuMedia Verlag.

[Pal07] Wladimir Palant. (CVE-2009-0357) XMLHttpRequest allows reading HTTPOnly cookies, 2007. `https://bugzilla.mozilla.org/show_bug.cgi?id=380418`.

[PW03] Birgit Pfitzmann and Michael Waidner. Analysis of Liberty Single-Sign-on with Enabled Clients. *IEEE Internet Computing*, 7(6):38–44, 2003.

[S. 05] S. Cantor et al. Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0. `http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf`, March 2005.

[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.

[Sim13] SimpleSAMLphp. SimpleSAMLphp Project. URL: `http://www.simplesamlphp.org`, 2007–2013.

[SMS+12] Juraj Somorovsky, Andreas Mayer, Jörg Schwenk, Marco Kampmann, and Meiko Jensen. On Breaking SAML: Be Whoever You Want to Be. In *21st USENIX Security Symposium*, Bellevue, WA, August 2012.

[The11] The National Institute of Technology. Special Publication 800-63-1: Electronic Authentication Guideline, December 2011. `http://csrc.nist.gov/publications/nistpubs/800-63-1/SP-800-63-1.pdf`.

[The13] The Open Web Application Security Project (OWASP). Cross-Site Request Forgery (CSRF), 2013. https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF).

[WCW12] Rui Wang, Shuo Chen, and XiaoFeng Wang. Signing Me onto Your Accounts through Facebook and Google: a Traffic-Guided Security Study of Commercially Deployed Single-Sign-On Web Services. In IEEE, editor, *Security & Privacy 2012*, 2012.