Your Software at my Service

Security Analysis of SaaS Single Sign-On Solutions in the Cloud

Christian Mainka*
Horst Görtz Institute for IT-Security
Bochum, Germany
christian.mainka@rub.de

Vladislav Mladenov†
Horst Görtz Institute for IT-Security
Bochum, Germany
vladislav.mladenov@rub.de

Florian Feldmann†
Horst Görtz Institute for IT-Security
Bochum, Germany
florian.feldmann@rub.de

Julian Krautwald†
Horst Görtz Institute for IT-Security
Bochum, Germany
julian.krautwald@rub.de

Jörg Schwenk
Horst Görtz Institute for IT-Security
Bochum, Germany
joerg.schwenk@rub.de

ABSTRACT

Software-as-a-Service (SaaS) is typically defined as a rental model for using a complex software product, running on a centralized computing platform, using a thin client (most frequently a web browser). As such, it is one of the major categories of Cloud Computing, besides IaaS and PaaS.

While there are many economic benefits in using SaaS, each company must nevertheless enforce control over its own data processed in the Cloud. One of the most important building blocks of such an enforcement scheme is Identity Management (IdM), whereat the industry standard for IdM is SAML, the Security Assertion Markup Language.

In this paper, we study the security of the SAML implementations of 22 SaaS Cloud Providers (SaaS-CPs) and show that 90% of them can be broken, resulting in company data exposure to attackers on the Internet. The detected vulnerabilities are exploited by a wide variety of attack techniques, ranging from classical web attacks to problems specific to XML processing.

1. INTRODUCTION

`Software-as-a-Service (SaaS).` It is difficult to define what SaaS exactly is. Even "The NIST Definition of Cloud Computing" [25] does not help: Any application running on a "Cloud Platform" is classified as SaaS there. Informally, many people think of SaaS as an *off-premise* replacement of a complex software system that previously was running *on-premise*. For instance, a customer relationship management (CRM) system that previously was hosted in the company's own computing center and was accessed through some additional client software, is now hosted in some Cloud Computing center and accessed via a web browser.

In any case it should be clear that company data, which was previously stored on-premise and protected by network perimeter security mechanisms like firewalls and VLANs, is now hosted in a Public Cloud system accessible from the Internet. Thus, the data is no longer protected by the company's firewalls, but accessible to anyone who can circumvent the corresponding SaaS security mechanisms.

`Authentication in SaaS.` Authentication and authorization are the most important SaaS security mechanisms. For this, there are commonly two possibilities:
(1.) Passwords.
The SaaS Cloud Provider (SaaS-CP) authenticates users by comparing their Username/Password pairs to some stored values. Each Username/Password pair is associated with an account. Access rights are directly linked to the user account. For example, a resource could be available only for a specific user or for a list of specific users.
(2.) Federated Identity Management (IdM)/Single Sign-On (SSO).
The SaaS system accepts SSO authentication tokens from a trusted Identity Provider (IdP). These tokens may contain identity information and/or additional authorization information.

This paper focuses on the security of the SSO authentication related mechanisms.

`SAML.` The most important industry standard for IdM is the Security Assertion Markup Language (SAML) [31]. SAML is based on the eXtensible Markup Language (XML) and enables the secure exchange of XML-based authentication and authorization messages. In conjunction with SSO systems, SAML especially offers a standardized format for authentication tokens.

`Differentiation from Web Applications.` A web application is commonly defined as an application that uses a web browser as client software. By this definition, every SaaS-CP is also a web application, in case it can be accessed via a web browser. In contrast to most standard web applications, SaaS-CPs (1.) store a great amount of valuable data and (2.) offer rich authentication and authorization interfaces. Property (1.) makes them a very attractive target for attacks, whereas property (2.) can be used to strengthen security.

`Security Analysis of online SaaS-CPs.` This paper identi-

* This work has been funded by the European Union within the European Regional Development Fund program.
† The authors were supported by the SkIDentity project of the German Federal Ministry of Economics and Technology (BMWi, FKZ: 01MD11030).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright 2014 ACM 978-1-4503-3239-2/14/11 ...$15.00.

fies components that are responsible for the SSO authentication. We therefore identified the message flow of the token processing, the components that are passed and their interaction with each other. Based on this, we found eight different attacks and categorized them within three different attacker models (AM). Model AM1 is the weakest one and models an attacker that only uses publicly available information (URL of the Server, SAML specification, ...). AM2 is stronger than AM1 – the attacker has access to a valid authentication token, for example, because he has a valid account on the target SaaS-CP. Attacker model AM3 uses Cross-Site-Request-Forgery (CSRF) [29] to lure a victim into clicking on a link. The goal of the attacker in each AM is to get access to a protected resource on the target SaaS-CP and thus break the authentication.

`Results.` We performed a large-scale security study which evaluates 22 SaaS-CPs. Applying the eight identified attacks, we were able to circumvent security mechanisms on 90% of them. This is an alarming result, given the value of the data stored there. The majority of the corresponding SaaS-CPs responded quickly and positively, when we communicated our findings to them. Most SaaS-CPs have already taken measures to mitigate the reported attacks and some of them acknowledged our work [12, 11, 10, 13, 14].

`Contribution.` The contribution of this paper can be summarized as follows:

- Analysis of SaaS-CP authentication mechanisms and identification of security related components on a logical level.
- Identification of eight SAML-related attacks and their corresponding component.
- Security study of 22 real-life SaaS-CPs by applying the identified attack classes and analyzing the results.

`Outline.` Section 2 describes our findings about the logical architecture of security related components of SaaS-CPs. In Section 3, we describe our attacker model by differentiating between three different categories of required attacker capabilities. Section 4 explains our methodology for selecting the SaaS-CPs we analyzed and for the security analysis we performed on the selected SaaS-CPs. Sections 5 through 7 offer detailed descriptions of all attack classes we identified, sorted by required attacker capabilities. Examples of our experiences while performing the security study are provided in Section 8. Section 9 discusses security insights to be gained from this study. Section 10 lists past and current related research work and finally, we conclude our study in Section 11.

2. SAAS PROVIDER MODEL

The architecture of an SaaS-CP regarding authentication is a very complex system consisting of different modules and entities interacting with each other. In this section, the role of these components and the communication between them is explained.

2.1 Single Sign-On (SSO)

Using authentication via SSO has many advantages over simple Username/Password mechanisms. Whereas for the former, the user has to remember multiple different Username/Password combinations, this overhead can be significantly reduced with SSO. Also, the security of Username/Password relies solely on the strength of the password provided by the user, but SSO allows for the adoption of several technical measures to further enhance the security of the login procedure.

In general, authentication using SSO is done utilizing a trusted third party called Identity Provider (IdP). When a user uses his user agent (UA), e.g., a web browser, to request a login to the service provider (*SaaS-CP*), instead of asking for Username/Password, the service issues a *Token Request* and redirects the client to the *IdP*. After proper authentication, the IdP issues a signed authentication token (*Token*) and redirects the client back to the SaaS-CP, where the token will be validated and the user logged in.

2.2 Components

Figure 1 shows a generic block diagram of the different security related components of an SaaS-CP:

The **Client** is a user communicating with an SaaS-CP via his UA, for example, a web browser. The communication takes place via the HTTP Protocol.

On the server side of this communication, the SaaS-CP provides a **Web Frontend**, for example, a web server listening on a specific port and forwarding the traffic to the according handlers (PHP, Java, ...). Additionally, the Web Frontend implements RFC 6265 [5] in order to make HTTP stateful and allow the deployment of different web applications.

The **Username/Password** module manages the corresponding authentication and the password-recovery mechanism. For example, it compares the Username/Password combination sent to the Web Frontend with the information stored in the User Database.

The **Session Management (SM)** module resolves the received Session Cookies by the Web Frontend to a user identity and forwards this information to Authorization & Access Management (AAM).

The **SSO** module carries out the verification of the received authentication tokens (c.f. Section 2.4). Additionally, it forwards the information about the authenticated user to AAM. During the verification, the SSO module fetches the configured IdP certificate from the User Database and uses it for verifying the provided digital signature within the authentication token. The SSO module contains three internal modules:

- **The Parser** is a module that converts an input string into data objects, which can then be further processed by following components. The structure of the parsed messages can differ according to the IdM protocol, for example, this could be a JSON or an XML parser. For our analysis, the SAML case is most important, in which a XML parser is applied.
- The verification of the token is provided by the **Verificator**. This is a module responsible for the validation of all security relevant parameters within the authentication token, c.f. Section 2.3.
- After the verification, the information regarding the authenticated user will be extracted from the token and forwarded to the business logic. This process is provided by the **Processor**. The information regarding the user is usually his name, which is then looked up in the AAM to get the according access rights. At the end of the processing, the authentication token will be deleted, since it is not needed any more.

The **Authorization & Access Management (AAM)** component controls the access to the restricted resources. It gets information regarding the authenticated user from the previous components (Username/Password, Session Management and Single Sign-On), fetches information from the User Database and enables or restricts access to the resources.

The **User Database** stores information about users and their credentials, for example, Username/Password combinations with their

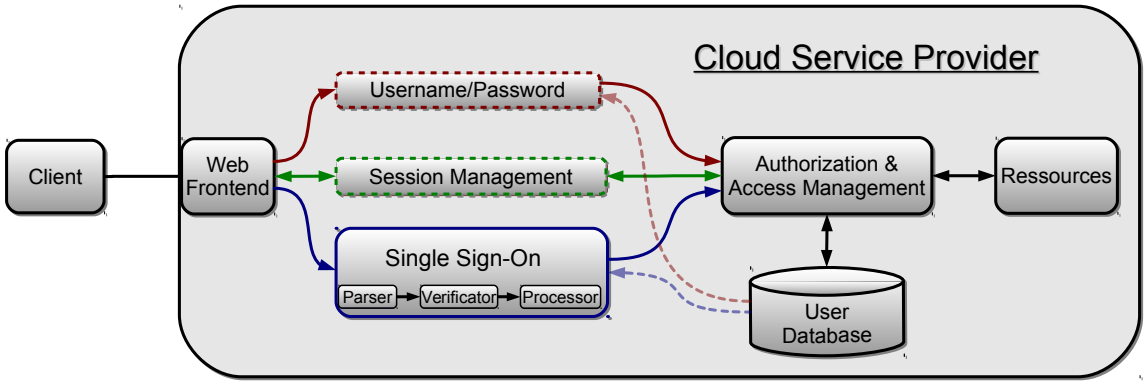


Figure 1: Overview over the different modules related to the authentication process on the Cloud Service Provider.

corresponding access rights. Additionally, it stores the SSO configuration data, for example, the endpoint and certificate of the federated IdP. This data is essential to establish a trust relationship between the SaaS-CP and the IdP.

The **Resources** include the entirety of data accessible to registered users.

2.3 SaaS Authentication flow

Whenever an HTTP Request arrives at the SaaS-CP, the `Web Frontend` module checks if a corresponding Session Cookie is provided, indicating that the user has already been authenticated.

`Initial Authentication`. Initial Authentication states that the user does not have a current session with the SaaS-CP. He cannot provide a corresponding Session Cookie and therefore must authenticate first. In this case, the user can choose to authenticate via (1.) Username/Password or to initiate (2.) SSO.

(1.) During the Username/Password authentication, the module verifies the correctness of the user supplied credentials. For that purpose the `Username/Password` module fetches the data stored within User Database and uses it for verification. In case that the authentication is successful, the information about the user is forwarded to AAM and a Session Cookie is set to make the authentication persistent.

(2.) The SSO procedure on Cloud Service Provider side consists of two phases: The redirection of the user to the IdP and the verification of the received authentication token. In the first of these phases, the SSO module fetches information about the federated IdP from the User Database. It then generates the token request and redirects the user to a specified IdP. In the subsequent phase (after the Client provides the token), the SSO module verifies the received token. For the verification of the digital signature, it fetches the IdP's certificate from the User Database. In case of successful token verification, the SSO module extracts information about the user's identity from the token and forwards it to AAM. Finally, a Session Cookie is set by the Web Frontend and the authentication is made persistent.

`Repeated Authentication`. Repeated Authentication indicates that the user has already been authenticated, either by Username/-Password or by SSO authentication. In this case, the received Session Cookie is forwarded to the `Session Management` module that resolves the identity of the user via the value of the Session Cookie. Consequentially, the identity of the user is forwarded to AAM.

2.4 SSO Authentication Token

On a logical level, an SSO authentication token, called in SAML *Assertion*, always contains certain information, regardless of the actual SSO mechanism. These contents are defined as follows:

- The `Identity` (I) is the value representing the user on the SaaS-CP. Typically, this is an email address to uniquely identify the user. In SAML context, I is represented by the element *Subject* [31, Section 2.4.1].

- `Freshness` ($\mathcal{N}$) is used to limit the usage of the token. This can be in the form of a nonce so that the token can only be used once, a timestamp which restricts the usage to a specific time slot, or a combination of both nonce and timestamp. The SAML tokens analyzed for this study usually bear timestamp elements *NotOnOrAfter* and *NotBefore* [31, Section 2.5.1] within the *Conditions* element as a freshness value $\mathcal{N}$. As a further freshness parameter, the unique *ID* [31, Section 1.3.4] of the Assertion is used as a nonce.

- The `Destination` ($\mathcal{D}$) is contained in the token to restrict the usage to a specific SaaS-CP. The value $\mathcal{D}$ is typically represented by the *AudienceRestricton* [31, Section 2.5.1] and *Recipient* [31, Section 2.4.1] elements.

- To protect the token information, a `Signature` or `HMAC` (σ) is used. The IdP is the one to chose which information are protected by σ. SAML tokens use the XML Signature standard [20] to realize this protection: A *Signature* element is added to the *Response*. To determine the *Assertion*, σ belongs to, a *Reference* to the *ID* attribute of said *Assertion* is used in SAML context. The IdP uses its secret key k to generate σ. The corresponding public key is then included in a certificate which is stored on the Service Provider (SP). Note that there is no PKI connected to this certificate. The certificate contains just a public key plus some meta information.

Summarized, an SSO token can be defined as $t = (I, \mathcal{N}, \mathcal{D})$ together with a corresponding signature σ. During our study, we found that most SaaS-CPs expect multiple/redundant information within their SAML tokens, so that we can define $t_{\text{real}} = (I, \mathcal{N}_0, \ldots, \mathcal{N}_i, \mathcal{D}_0, \ldots \mathcal{D}_j)$. For instance, we found tokens with up to five freshness parameters and up to three destination parameters. An interesting observation at this is that σ does not always protect all data specified in the token. For instance, $\mathcal{N}_0$ might be signed but $\mathcal{N}_1$ might not.

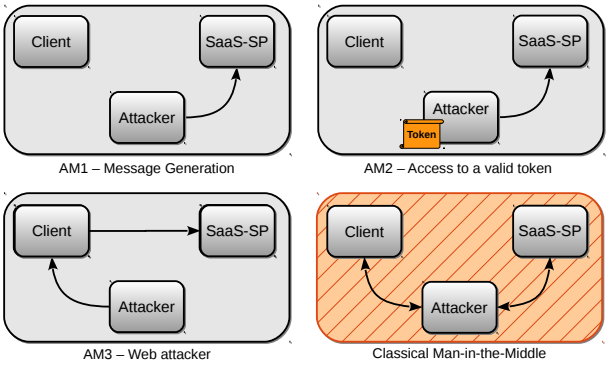


Figure 2: Overview of the relevant attacker models: AM1, AM2, AM3. The "Classical Man-in-the-middle" attacker controlling the network traffic between Client and SaaS-CP is not relevant for the attacks introduced in the paper and thus not considered further.

3. ATTACKER MODEL

For the security analysis explained in the next section, we distinguish three different attacker models as shown in Figure 2. Each attacker type has different capabilities.

All of our attacker models do not require the attacker to control the network communication. Thus, we do not require the attacker to be able to eavesdrop or modify messages sent between client and servers, as it is the case in classical Man-in-the-Middle (MitM) attacks.

Finally, we do not assume the attacker having knowledge of any secrets (passwords, secret keys etc.) other than those established by himself.

`AM1 - Message Generation.`

For the first type of attacks, we assume that the attacker is able to generate valid XML messages of the SaaS-CP's specified format. These messages can be of any publicly available XML format (plain XML, SAML, XSLT, DTD etc.) and can contain any data which is publicly available or chosen arbitrarily by the attacker.

Note 1: No secret information, such as secret keys or passwords known either to the SaaS-CP or to the victim must be used to generate these messages. For example, *AM1* includes the capability to create arbitrary tokens $t = (I, \mathcal{N}, \mathcal{D})$, but no corresponding valid signatures $\sigma = SIG_{IdP}(t)$ (as the attacker does not have access to the secret key of IdP).

Note 2: An exception to *Note 1* occurs for tokens generated by an attacker who uses his own key material. In this case, the secret key used to generate the digital signature σ was chosen by the attacker, thus, he can create tokens with arbitrary content and then correctly sign them with a corresponding signature σ.

`AM2 - Access to Valid Token.`

In *AM2*, we assume the attacker having access to a valid token $t = (I, \mathcal{N}, \mathcal{D})$ of the victim, including the corresponding signature $\sigma = SIG_{IdP}(t)$. Multiple possibilities to obtain such a token exist: They can, for example, be stolen via Cross-Site-Scripting (XSS) [46], or they can simply be found in public support forums by using an online search engine like Google. Please note that token theft via MitM attacks or eavesdropping on the communication channel between the Client and SaaS-CP is not part of *AM2*.

A special case of *AM2* occurs, when the attacker himself is a valid user of the system. He can then use his credentials on his IdP to let it issue tokens for his own account on demand (but not for the victim account).

Note: In general, an attacker will not be able to chose the contents of such a token arbitrarily – even as a regular user he will be restricted to his given identity and the according permissions. Thus, the main goal of attacks related to *AM2* is to expand the rights provided by the given token.

`AM3 - Web attacker.`

For attacker model *AM3*, we assume that the attacker is able to influence the victim to click on an attacker provided link. This includes both, a technical method to provide an attacker generated link to the user (e.g., by e-mail or through user forums), and the technical and/or social means to convince the user to actually activate (i.e. click on) the link.

AM3 further assumes that the victim is logged in to his account on the targeted SaaS-CP at the time of clicking on the malicious link (i.e., the victim has an active authenticated session with the SaaS-CP – see *Repeated Authentication* in Section 2.3).

4. METHODOLOGY

`Selection criteria.` For this study, we intended to analyze the security of the most important SaaS-CPs. Naturally, the first approach was to analyze all SaaS-CPs from the Alexa Top 100,000. Unfortunately, this proved to be very inefficient: The main problem of this approach was to decide whether or not an analyzed domain is an SaaS-CP, as there is no strict labeling for this kind of site (c.f. Section 1). This is why it is not possible to simply do an automatic keyword search for, for example, "SaaS-CP", as this would lead to (1.) a substantial amount of false positives, in case websites just advertised working in conjunction with certain SaaS-CPs, and (2.) an at least equally considerable amount of false negatives for those SaaS-CPs, which called themselves, for example, "Cloud Service Provider" instead of "SaaS-CP". Thus, this approach would have meant manually analyzing all 100,000 domains including their corresponding subdomains, and possibly also studying the provided documentation and external descriptions.

In order to find a more efficient way of identifying the most important SaaS-CPs, our second approach was to use precompiled lists of existing Cloud Service Providers. Such lists could be found at major Identity Providers (e.g., OneLogin [28] and Bitium [6]), who maintain up to date lists of their respective supported Cloud Service Providers. Wikipedia also offers a list of known Cloud Service Providers [41]. Further internet research yielded additional independent lists [16, 38]. After consolidating all the lists and eliminating overlaps, we used the resulting consolidated list as a basis for our analysis. From this list we selected all SaaS-CPs, which satisfied certain selection criteria:

- Cloud Service Provider Type: As we intended to analyze SaaS-CPs, we concentrated only on Software-as-a-Service Providers. For this study, this was a sensible approach, as other Cloud Provider Types, such as Platform-as-a-Service (PaaS) or Infrastructure-as-a-Service (IaaS), differ substantially in terms of attack surface and handling. Thus, these other types would require a mostly different approach in terms of attack types, attacker models, etc.
- Free Trial Accounts: We did not have any specific funding for opening and maintaining accounts with SaaS-CPs for this study. Thus, we were limited to analyze only those SaaS-CPs which offer a free trial account for their services.
- SAML-based Single Sign-On: Our goal was to evaluate the security of SAML-based SSO authentication modules. Thus, we included all SaaS-CPs which offer SSO functionality with support for SAML in their authentication procedures.

`Security analysis.` After we selected all relevant SaaS-CPs for our study, we specifically analyzed the security architecture of the SSO authentication mechanisms of these SaaS-CPs. On a logical level, we identified the components related to the authentication process and the relevant information flows between them. An abstract overview of our findings is given in Section 2.2.

Based on our understanding of this logical architecture, we identified possible attack classes against the corresponding component. Sections 5, 6 and 7 give detailed descriptions of all identified attacks. Testing has been performed in a `semi-automated blackbox analysis`. Due to the fact that we analyzed real applications for which we did not have access to the source code, we were only able to perform blackbox testing on the SaaS-CPs. For the analysis itself, we used a self-developed tool to dynamically generate different messages and observed the reaction of the targeted systems. The analysis of each SaaS-CP was split into three phases:

- `Learning phase` – This phase introduced the calibration of our tool as a preparation for the next phase. We configured the tool to generate valid messages that were accepted by the target SaaS-CP. Analyzing the system's responses allowed us to estimate the "normal" (i.e., intended) behavior of the system.
- `Security phase` – This phase was split into several steps - one for each attack. During this phase's steps, we used our tool to generate different messages deviating from those accepted by the target SaaS-CP in the previous phase. The actual deviations of the generated messages in each step were depending on the corresponding attack class. All attack classes were applied to each analyzed SaaS-CP. We then observed the reaction of the systems and analyzed the results.
- `Verification phase` – To verify the results obtained from the previous phase and to mitigate possible false-positives, we used a completely independent platform, for example, another PC, to manually execute the exploits and verify the correctness of the results.

5. AM1 RELATED ATTACKS

This section describes attacks related to AM1: (1.) Signature Exclusion (∅Sig), (2.) Certificate Faking (CF), (3.) XML External Entity Attack (XXEA) and (4.) XSLT Attack (XSLTA). For all these attacks, only publicly available information is required. Especially, no knowledge of any foreign secret keys is needed.

5.1 Signature Exclusion (∅Sig)

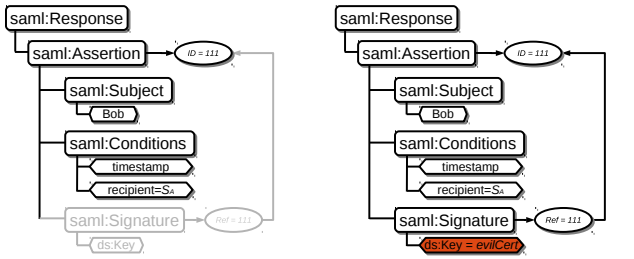
The integrity of all authentication tokens should be protected. In case of SAML, this is realized by a digital signature $\sigma = SIG_{IdP}(t)$. Signature Exclusion (∅Sig) exploits a vulnerability in the verification logic allowing the usage of unsigned tokens, see Figure 3 (a).

Since no digital signature for the token is required, an attacker can generate tokens containing arbitrary identities I of other users.

`Exploit.` The attacker creates authentication tokens containing statements about other users, $t = (\ldots, I_{Alice}/\, I_{Bob}/\, I_{Admin}, \ldots)$. He then sends the token to an SaaS-CP (S_{target}) and is logged in with the corresponding identity.

`Impact.` The attacker gains access to arbitrary accounts and their resources.

`SaaS-CP component.` The attack is targeted at the *SSO Verificator*, which should require that the authentication token is signed and verify the applied signature. By this means, the integrity of the authentication token is guaranteed.



(a) The SAML token does not contain any signature. This means, no protection of integrity or authenticity is provided.

(b) The SAML token is signed with an untrusted key. If the key stored in the token is used for the verification without validating the trust relationship to it, CF is applicable.

Figure 3: Signature Exclusion (a) and Certificate Faking (b)

5.2 Certificate Faking (CF)

The cryptographic verification of the digital signature guarantees the integrity of the token. Additionally, it is essential to verify the token's authenticity, too (c.f. [32, Section 4.4.2]). In other words, the SaaS-CP should check whether the token was signed by a trusted IdP. The CF attack utilizes possible flaws in the selection logic of the key used for the verification of tokens, by providing an attacker generated token signed by an attacker generated key.

In order to run the attack, the attacker must be able to create SAML tokens and sign them with his own self-created key.

`Exploit.` The attacker creates a token $t = (I, \mathcal{N}, \mathcal{D})$. Then, he creates a secret key *evilKey* and a corresponding public key. The secret key is used to compute the digital signature $\sigma = SIG_{evilKey}(t)$. The attacker then uses his key pair to create a certificate *evilCert* containing the corresponding public key to verify σ. SAML uses the XML Signature standard that allows to store *evilCert* directly within the XML Signature as shown in Figure 3 (b). If the target SaaS-CP uses *evilCert* to verify the signature σ (without prior check of the trust relationship for the corresponding key), the token will be accepted as valid.

`Impact.` The attacker gains access to arbitrary accounts, since he can generate and sign valid tokens containing the identities of other users, for example, $t = (I_{admin}/\, I_{bob}, \mathcal{N}, \mathcal{D})$ with $\sigma = SIG_{evilKey}(t)$.

`SaaS-CP component.` The attack targets the *SSO Verificator*, which should verify that the authentication token is signed by a trusted third party instead of accepting any key provided with the token (although the XML Signature standard allows to include certificates, it is essential to verify whether it is a trusted certificate). This attack can be mitigated by manually deploying the trusted certificates to the corresponding SaaS-CP and not using any certificates provided with the token.

5.3 XML External Entity Attack (XXEA)

XML offers the possibility to describe the document's structure by using a Document Type Definition (DTD). Unfortunately, the usage of these features can lead to security vulnerabilities enabling very efficient Denial-of-Service attacks [17] or allowing unauthorized access to files stored on the target SaaS-CP, for example, */etc/passwd* or key files.

The idea of the XML External Entity Attack (XXEA) is shown in Figure 4. The attacker sends an XML document containing the attack vector as shown in Listing 1. The vulnerable application parses the XML document and processes the defined DTD. The DTD con-

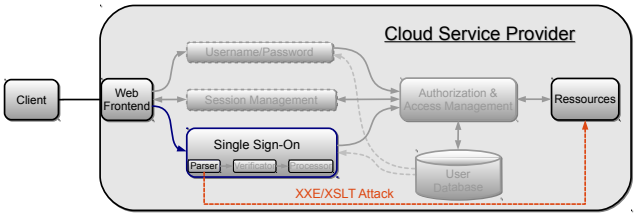


Figure 4: The attacker sends an XML document containing an Entity, which points to a file stored on the local filesystem. As a result the attacker breaks out of the usual processing schema and bypasses the security verification provided by the SSO-Verificator plus AAM and reads locally stored files.

tains an External Entity reading a resource from the filesystem, in this case the `/etc/passwd` file, and sends the content to the attacker.

In order to start XXEA, the attacker only has to create a valid XML message containing a DTD. Note that the message does not has to be a SAML token (c.f. Listing 1).

`Exploit.` An example exploit is shown in Listing 1. The XML message contains two External Entities. The first Entity (`file`) will read the content of the protected resource. The second Entity (`send`) is used to send this content to a web server controlled by the attacker via a GET parameter. If the SaaS-CP reflects the content of the `file` Entity in the HTML response, which will be automatically displayed in the attacker's browser, the `send` Entity is unnecessary. However, this is rarely the case for SAML token verification.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE Response [
<!ENTITY file SYSTEM "/etc/passwd">
<!ENTITY send SYSTEM "http://attacker.com/?read=&file;">
]>
<samlp:Response>
  <attack>&send;</attack>
</samlp:Response>
```

Listing 1: XML message containing the XXE attack vector.

The listing only sketches the concept of the attack. The code as shown will not work on most XML parsers, because the usage of the External Entity `file` within the External Entity `send` is not allowed. However, this technical restriction can be avoided as shown in [26].

`Impact.` XXEA allows an attacker to read arbitrary files within the context of the used web server. Particularly, it is possible to read configuration and SSL keystore files.

`SaaS-CP component.` The attack targets the *SSO Parser*. To prohibit XXEA, the processing of DTDs should be disabled. XML Schema [36] can be used to verify the structure of XML messages.

A more detailed explanation of XXEA on SAML interfaces can be found in our Blog Post [9].

5.4 XSLT Attack (XSLTA)

Extensible Stylesheet Language Tranformation (XSLT) is a language for transforming XML documents into other documents, for example, XML, HTML, JSON or even PDF [23]. The XML Signature standard allows the usage of XSLT by definition, and thus, XSLT can be used in SAML.

XSLT is a turing complete language [27]. By this means, it is possible to use XSLT, for example, to read/write files on the local filesystem and send them over the Internet. Furthermore, the XSLT transformation will be executed before the digital signature is verified. Thus, an attacker can send a SAML token including a digital signature containing the XSLT Attack (XSLTA) vector, but it is not required that the signature is valid.

The attacker needs the same resources as for XXEA. In comparison to XXEA, the message has to be a SAML token. However, this token does not have to be signed with a valid key nor the signature needs to be valid.

`Exploit.` The attacker prepares a SAML token t and creates an XML Signature for it. Note that it is not important to have a correctly computed signature value – the XSLTA only requires a well-formed XML document. The attacker adds a `Transform` element to the XML Signature and places the XSLT payload in it as shown in Figure 5.

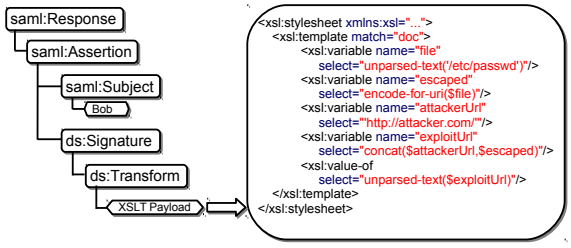


Figure 5: XSLTA payload that reads the `/etc/passwd` file and forwards its content to an attacker controlled server.

The basic idea of the attack is similar to XXEA: First, the attacker reads an arbitrary file using XSLT (in this example by using the `unparsed-text()` function). Afterwards, he forwards the contents of the file to his own server via a GET parameter.

`Impact.` XSLTA allows accessing files within the context of the used web server.

`SaaS-CP component.` The attack targets the *SSO Verificator*. The *SSO Verificator* should mitigate the usage of XSLT within the token.

6. AM2 RELATED ATTACKS

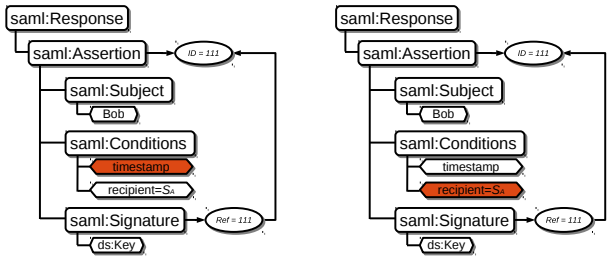
In this section, attacks related to AM2 are described: (1.) Replay Attack (RA), (2.) XML Signature Wrapping (XSW), (3.) Token Recipient Confusion (TRC). Compared to AM1, AM2 describes a stronger attacker. He is able to do everything an attacker according to AM1 can do (because AM1 only uses publicy available information). Additionally, the AM2 attacker has access to a valid token.

6.1 Replay Attack (RA)

Every SSO protocol provides parameters $\mathcal{N}$ to limit the reuse and lifetime of the authentication tokens. Taking into account that the reuse of tokens is optional [32, Section 6.4.4], the validation of the attributes providing freshness is not considered as critical.

On the other hand, the time restriction regarding the usage of authentication tokens is more critical and should be evaluated [32, Section 6.4.1]. Otherwise, tokens issued once might be valid for an extended time period or even an infinite amount of time [31, Section 2.5.1.2].

The attacker needs access to a valid token (*AM2*). More specifically, the token in question is required to be valid for the SaaS-CP at any time in the past. This can be achieved if the attacker had legitimate access (for a limited period of time) to the SaaS-CP via SSO and used this access to generate and store a token for himself. Alternatively, searching for published tokens in forums or in techni-



(a) SAML token with expired timestamps is sent to the SaaS-CP.

(b) SAML token addressed for service S_A will be sent to S_{target}.

Figure 6: Replay Attack (a) and Token Recipient Confusion (b)

cal documentations could also provide valid, though most possibly outdated, tokens.

`Exploit.` The attacker sends an expired authentication token to the target SaaS-CP, see Figure 6 (a). In case that the unlimited reuse of authentication tokens is applicable and the token is successfully verified, the attack is classified as successful.

`Impact.` The attack's impact is average, since the attacker has limited attack surface – he can only spend authentication tokens he possesses. However, the potential impact drastically rises in case the attacker gains hold of an authentication token granting him extended access rights (e.g., as an administrator of the system).

`SaaS-CP component.` The attack specifically targets the *SSO Verificator*. This component should validate attributes providing the corresponding restrictions, i.e., the freshness parameter $\mathcal{N}$. In the SAML context relevant to this study, this parameter is represented by `NotOnOrAfter` and `NotBefore` [31, Section 2.5.1]. Failing to properly verify these parameters will enable this attack type. Another possibility to enable this attack type would be via additional freshness attributes, which are not part of the digital signature σ.

6.2 XML Signature Wrapping (XSW)

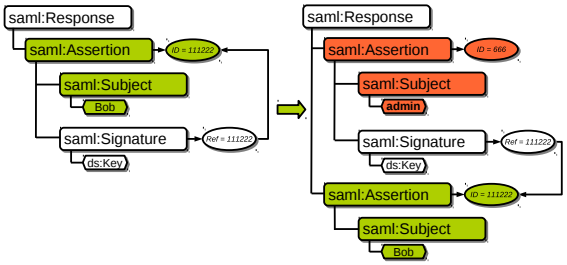


Figure 7: The authentication token is signed for a user Bob. Via XSW the attacker can inject a second Assertion containing another identity (e.g. *admin*). The verification logic will verify the Assertion pointed by the *Ref*, which is valid. The business logic (*SSO Processor*) will process the injected (malicious) Assertion.

The idea of XSW [24] is to exploit the separation between *SSO Verificator* and *SSO Processor* (see Figure 1). In case both logics have different "views" of the same document, XSW can be applicable.

For this attack to work, the attacker modifies the contents of the token by injecting malicious data without invalidating the signature. One example for this is shown in Figure 7. The goal is to force the *SSO Verificator* to use different elements than the *SSO Processor*. For the given example, the *SSO Verificator* will verify the signature based on the contents of the original `Assertion`, which is selected by ID. However, if the *SSO Processor*'s program logic automatically process the first `Assertion` found within the token, an attacker can bypass the integrity protection and enforce the processing of unverified data on the SaaS-CP.

The attacker needs the same resources as for Replay Attack (RA), in order to provide the attack – access to a valid token.

`Exploit.` The attacker manipulates his token by injecting malicious contents, for example, the identity I of other users, see Figure 7. Multiple possibilities to apply XSW exist and a detailed study regarding this attack type was already published by Somorovsky et.al [34, 35].

`Impact.` XSW allows an attacker to log into arbitrary user accounts and gain unauthorized access to their data.

`SaaS-CP component.` The attack targets the discrepancy in the program logic of *SSO Verificator* and *SSO Processor*. The latter should extract and forward only exactly the data verified by the former to AAM.

6.3 Token Recipient Confusion (TRC)

In real-life SSO there exist multiple SaaS-CPs federating with the same IdP. In order to distinguish the authentication tokens generated for different SaaS-CPs, each token contains information $\mathcal{D}$ about its recipient. In most cases this is the URL of the SaaS-CP for which the token was generated.

The goal of Token Recipient Confusion (TRC) is to use an authentication token t_A generated for a service S_A (depicted in Figure 6 (b) with highlighted `Recipient` element) on a second service S_{target}. The attack is considered successful, if S_{target} becomes "confused" by the recipient of the token and accepts t_A as valid.

As in the previous two attacks, the attacker has access to a valid token. An additional requirement is that both services (S_A and S_{target}) have to be federating with the same IdP. This is a realistic assumption, since an IdP usually offers authentication services for multiple SaaS-CPs.

`Exploit.` There are two different approaches for a TRC exploit:

Exploit 1: Suppose that SaaS-CPs S_A and S_{target} are accepting tokens from the same IdP, and the attacker does not have access to S_{target}. The attacker does, however, have legitimate account on S_A, thus he can request a token $t_A = (\ldots, \mathcal{D}_A, \ldots)$ from the IdP. By sending t_A to S_{target} (instead of S_A), the attack is performed. It is considered successful if t_A is accepted by S_{target}; the attacker is thus logged in with the same account name as he has for S_A and gets access to S_{target}'s corresponding resources.

Exploit 2: Alternatively, the attacker can set up his own SaaS-CP (S_{bad}) offering some service for registered users (e.g., a weather forecast). To authenticate to S_{bad}, SSO is used and the attacker specifically federates it with the same IdP used by S_{target}. After that, the attacker lures his victim (a legitimate user of S_{target}) to register with and authenticate to S_{bad}. Instead of or in addition to its usual service (weather forecast), S_{bad} stores all tokens in a database so that the attacker can access them. The attacker can then try to use the tokens to log in on S_{target} as the victim. The attack is considered successful, if an authentication token t_{bad} issued for the victim for service S_{bad} is successfully verified on S_{target}.

`Impact.` According to [3], this type of attacks is classified as critical, since information disclosure or privilege escalation is possible. A dishonest user can redeem his tokens on different services and get unauthorized access to restricted resources. Furthermore, a malicious SaaS-CP could collect authentication tokens and forward

them to other SPs in order to get login in arbitrary accounts.

`SaaS-CP component.` This attack also targets the *SSO Verificator*, which is responsible for checking the restrictions regarding the destination of the token, $\mathcal{D}$. In the SAML context, specifically the `AudienceRestricton` [31, Section 2.5.1] and `Recipient` [31, Section 2.4.1] elements are relevant for this attack type.

7. AM3 RELATED ATTACKS

The only attack related to AM3 is Certificate Injection (CInj), because it makes use of CSRF.

7.1 Certificate Injection (CInj)

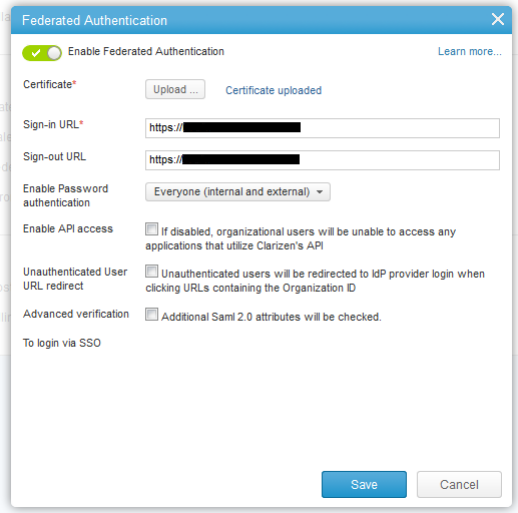


Figure 8: The SAML module is configured via the Web interface of the SaaS-CP. The certificate of the trusted IdP and the according endpoints can be uploaded and stored in the AAM module.

All tested SaaS-CPs provide a Web interface allowing to activate and configure the SSO module. See Figure 8 for an example. This is a critical area and thus has to be well protected.

During our research we recognized that the security of the entire SSO module (Figure 1) depends on the security of the applied Web interface and the stored information in the AAM module (which includes, e.g., the IdP certificate used for verification). This observation led to the Certificate Injection (CInj) attack. The basic idea of CInj is to inject a malicious certificate and store it in the AAM module. Since the SSO module uses this certificate for the token verification, tokens signed by the attacker, who possesses the private key to the injected certificate, will be successfully verified. By this means, even a correctly implemented SSO module, which mitigates all attacks directly related to this module, can still be bypassed.

In order to apply CInj we used CSRF attacks to inject a malicious configuration regarding the SAML interface. In case that the target SaaS-CP does not provide any CSRF protection, the attacker can enable and configure the SAML interface by injecting the malicious contents via a CSRF attack. As a result, he is able to remotely establish a trust relationship between the target SaaS-CP and a malicious certificate, which does not belong to any trusted IdP.

The attacker uses CSRF attack techniques in order to inject the malicious contents, thus, he must be able to lure the victim to click on a link or to visit a web page.

`Exploit.` An actual exploit for this attack type can be separated into three consecutive phases:

Phase 1 – Preparation. The attacker creates his private key *evilKey* and a corresponding public key and uses these to create a certificate *evilCert*.

Phase 2 – Configuration Injection. The attacker creates a malicious link containing the CSRF attack vector, i.e., the injection of *evilCert*. Luring the victim to click on that link, he will exchange the originally stored certificate in the User Database with the one provided by the attacker. This is possible because the HTTP-request that changes the certificate is sent via the victim's browser (using the victim's session cookies).

Phase 3 – Access to resources. The attacker can then generate a token t for an arbitrary user and sign it with the key belonging to *evilCert* generating σ. Then, he sends (t, σ) to the target SaaS-CP for verification. The target SaaS-CP will use the certificate stored in the AAM module and use this for the verification of σ. Since the stored certificate is *evilCert*, the verification is successful and the attacker can log in with the chosen identity.

`Impact.` If the attacker is able to inject his own SSO configuration, the SaaS-CP and the according SSO module will trust the attacker just as a regular trusted IdP. By this means, the attacker can generate valid tokens for any user on the SaaS-CP and log into his account.

`SaaS-CP component.` The attack uses the CSRF technique to enforce the victim to change changing configuration data without explicit user interaction. Therefore it targets the *Session Management*, which should include a protection against CSRF to mitigate the attack.

8. CLOUD STUDY

According to the responsible disclosure model, we promptly reported all vulnerabilities found to the liable security teams as well as to the Computer Emergency Response Team (CERT)[1]. In case we got a response from the developers, the time to fix the reported issues ranged from between a few days and several months. Furthermore, we supported the developer teams during fixing the reported issues.

8.1 Summary

In this survey we provided a security analysis of 22 SaaS-CPs, see Table 1. We discovered vulnerabilities, resulting in unauthorized access to restricted resources of a victim, in 20 of them. As shown in Table 1, one single SaaS-CP was susceptible to ∅Sig and none of them against CF. Almost 50% of the SaaS-CPs (10/22) were vulnerable against XXEA but only one was vulnerable against XSLTA. Six of the evaluated SaaS-CPs did not correctly evaluate the timestamps within the token, making RA attacks applicable. Some of these six SaaS-CPs did verify the freshness parameters only partly and some of them did not verify them at all, allowing the unlimited usage of the tokens for an infinite amount of time. 11 SaaS-CPs were vulnerable against XSW. This is a surprisingly huge amount with respect to the large scale study of SAML framework implementations in 2012 [35]. 17 out of the 22 SaaS-CP were vulnerable against TRC. During our evaluation, we found some

[1]https://cert.org

Service Provider	AM1				AM2			AM3	
	∅Sig	**CF**	**XXEA**	**XSLTA**	**RA**	**XSW**	**TRC**	**CInj**	**Summary**
Salesforce	×	×	×	×	×	×	×	×	×
Google Apps	×	×	×	×	×	×	×	×	×
Zoho	×	×	×	×	×	✓	×	×	✓
Zendesk	×	×	×	×	×	×	✓	×	✓
Clarizen	✓	×	✓	×	✓	✓	✓	×	✓
SAManage	×	×	✓	×	×	✓	✓	✓	✓
Shiftplanning	×	×	✓	×	×	×	✓	✓	✓
Panorama9	×	×	×	×	×	×	✓	×	✓
UserVoice (Marketing)	×	×	×	×	×	×	✓	×	✓
Instructure	×	×	×	✓	✓	✓	✓	×	✓
The Resumator	×	×	✓	×	×	×	✓	×	✓
BambooHR	×	×	×	×	×	×	✓	✓	✓
AppDynamics	×	×	✓	×	✓	✓	✓	×	✓
IdeaScale	×	×	✓	×	×	×	×	✓	✓
Panopto	×	×	×	×	×	✓	✓	×	✓
TimeOffManager	×	×	✓	×	✓	✓	✓	×	✓
HappyFox	×	×	×	×	×	✓	✓	×	✓
SpringCM	×	×	×	×	×	✓	×	×	✓
ScreenSteps Live	×	×	✓	×	×	✓	✓	×	✓
LiveHive	×	×	✓	×	✓	✓	✓	×	✓
Howlr	×	×	×	×	×	×	✓	✓	✓
CA Service Management	×	×	✓	×	✓	×	✓	✓	✓
Total	1	0	10	1	6	11	17	6	20/ 22

Table 1: Results of our practical evaluation. We have evaluated 22 SaaS-CPs against 8 different attacks: (1.) Signature Exclusion (∅Sig) (2.) Certificate Injection (CInj) (3.) XML External Entity Attack (XXEA) (4.) XSLT Attack (XSLTA) (5.) Replay Attack (RA) (6.) XML Signature Wrapping (XSW) (7.) Token Recipient Confusion (TRC) (8.) Certificate Injection (CInj). 20 of them were vulnerable to at least one attack so that we could successfully access unauthorized resources.

very interesting aspects about the impact of the TRC attack: Some vulnerable SaaS-CPs accepted tokens from other SaaS-CPs even if there was no account associated with the identity I contained in the token. The vulnerable SaaS-CP instantly creates a new account for the corresponding identity, even if the SaaS-CP is a payed service. Finally, about 30% of the SaaS-CPs were vulnerable against CInj and thus enabled a backdoor, which bypassed the protection mechanisms verificating the authentication tokens.

8.2 Details of Exploit

In the following, we want to highlight some of our findings and picked out some of the evaluated SaaS-CPs.

`Zendesk.`

Zendesk [44] (Alexa Rank Global 304 / US: 174) is a web-based ticketing software. Zendesk was not vulnerable to 7 out of 8 attacks. However, by using the TRC attack, we were able to prove that Zendesk did not perform the recipient verification ($\mathcal{D}$) on a received token. Due to this single missing verification, an attacker is able to redeem tokens generated for other SaaS-CPs for logging in at Zendesk and breaking the SSO authentication.

`Zoho.`

Zoho [45] (Alexa Rank Global 605 / US: 423) is offering a web-based office suite alongside project management, CRM and other services. Although Zoho offered a comparatively good overall-security concerning their SAML token verification (7 out of 8 attacks failed), we discovered a discrepancy within the program logic of their SSO Verificator and their SSO Processor. Due to this discrepancy, we were able to log in as arbitrary users and thus escalate our privileges making the XSW attack scenario applicable. This was very surprising to us, because Zoho did a very good job in preventing other SSO attacks (e.g., RA, TRC, ...), but were vulnerable only to XSW although a lot of research in this area is already published [35]

`Clarizen.`

Clarizen [15] (Alexa Rank Global 30.396 / US: 16.044) is a project management SaaS-CP. During our evaluation, we discovered that the provider did not verify whether a digital signature for the token is present – Clarizen was vulnerable to the ∅Sig attack and it was the only SaaS-CP where this attack worked. Interestingly, we could use this vulnerability to take a deeper look into Clarizen's token verification logic, because we could modify every parameter that normally would have been signed. We determined which of the provided timestamps $\mathcal{N}_0, \ldots, \mathcal{N}_4$ were validated and found that Clarizen verified only the timestamp value of $\mathcal{N}_0$ and ignored the values of $\mathcal{N}_1, \ldots, \mathcal{N}_4$. Unfortunately, this value $\mathcal{N}_0$ was not included in the digital signature, and thus could be altered by an attacker even if no ∅Sig attack was performed. This enabled RAs.

`IdeaScale.`

IdeaScale [21] (Alexa Rank Global 62,332 / US: 21,785) is a web-based innovation management platform. IdeaScale did a very good job implementing their SAML token verification and thus passed all tests targeting their SSO Verificator and SSO Processor. Unfortunately, they did not as well in securing the incoming HTTP requests changing the configuration of the web application. Thus, an attacker was able to bypass all security mechanisms of the SSO module by injecting an attacker-controlled certificate directly into the User Database. We thus categorized the Certificate Injection (CInj) attack scenario as applicable. CInj is a good example to show how important the relations between the different modules are. Even if the SSO module resists all known attacks, it can be still bypassed by injecting wrong keys into the database via a vulnerable

Session Management module.

Additionally, IdeaScale was vulnerable to the above-described XML External Entity Attack (XXEA), providing a debug-functionality of received SAML tokens displaying the contents of the locally pointed-to file.

`TimeOffManager.`

TimeOffManager [39] (Alexa Rank Global 162,258 / US: 121,438) is a web-based leave management solution.

TimeOffManager gives a good example for an XXEA attack. At first, we wanted to detect if the SaaS-CP is vulnerable to XXEA. To verify this, we simply created a token containing the External Entity `<!ENTITY send SYSTEM 'http://attacker.com'>`. Because our `attacker.com` server received a GET request, we knew that External Entities were resolved. Nevertheless, when we tried to read the file `/etc/hostname`, it did not work. We wondered why it was impossible and after some tries, we detected that TimeOffManager uses a load balancer and delegates our requests to different systems. On some of them, the `/etc/hostname` file did not exist, and therefore we got an error, on others, it worked and we could read it. The high percentage of affected SaaS-CPs can be explained by the fact, that the External Entities are turned on by default in the most XML parsers (even if they are rarely needed).

`Instructure.`

Instructure [22] (Alexa Rank Global 5,015 / US: 1,237) is an educational technology based company. While investigating their SSO authentication flow, we discovered that Instructure processes arbitrary XSLT instructions if they are contained in the XML Signature within the SAML token. This for itself is very dangerous, because XSLT could be used to access files on the machine or execute system commands. Interestingly we found out, that if we use XSLT to load an XML file from our own server (`http://attacker.com/a.xml`) that includes an XXEA attack, this XXEA is processed by Instructure. If an XXEA is contained directly within the token sent to Instructure, it is ignored. This indicates, that there are multiple XML parsers involved while processing one single SAML token, and each parser must be protected. Instructure was the only SaaS-CP vulnerable against XSLT. However, it shows how complex the different modules are. Even if the XML Parser forbids the usage of External Entities and provides protection against XXEA, the SSO Verificator can by exploited in the next step in order to start XXEA.

9. LESSONS LEARNED

The results of our evaluation revealed a multitude of existing security flaws in real-life SaaS-CPs. In this section we try to investigate the reasons why so many systems could be compromised.

`Verification of security critical parameters.`

In Section 2.4 we presented the information contained in an authentication token: `Identity` (I), `Freshness` ($\mathcal{N}$), `Destination` ($\mathcal{D}$) and `Signature` or `HMAC` (σ). Considering Table 1, one can say that the importance of the correct verification of σ is well understood and correctly implemented. Therefore, the most SaaS-CPs resist both $\emptyset$Sig and CF attacks.

Surprisingly large is the number of implementations that provide a faulty validation of $\mathcal{N}$ (6/22) and $\mathcal{D}$ (17/22). One reason for this could be the redundancy of these parameters. Developers seem to be confused about choosing the correct parameters for the verification. In some of these cases they validated parameters that are not protected by the digital signature and thus can be altered by an attacker. It has to be taken into account that a guideline describing how to correctly verify an authentication token and its relevant parameters already exists [32, Section 7.1]. However, our evaluation shows that it seems to be unknown to most developers, or it should be enlarged with example attacks for a better common understanding.

`XML Signature Wrapping.`

50% of the evaluated implementations were vulnerable to XSW. This result was not expected by the authors of this paper because a large scale study was published in 2012 [35]. Due to the good contact to some of the development teams, we learned that countermeasures were taken. However, not all existing attack vectors were considered. Based on this knowledge, we recognized the need of automated analysis.

`XML External Entity and XSLT Attack.`

Most implementations do not implement their own XML parser and XML Transformation, but use available free libraries. Unfortunately, in case the used libraries allow the usage of External Entities or XSLT within the token, XXEA and XSLTA are applicable. Since both features are rarely needed, their usage should be disabled by default. In case this functionality is needed, it can be still configured accordingly and then re-enabled.

`The security of the entire system.`

We want to stress the fact that it is important to consider the security of the entire system and not only the security of one module. Instead, it is essential to study the relations between the different modules and evaluate possible cross-module attacks – attacks affecting the security of more than one module. By this means, the high impact of attacks like CInj can be mitigated.

10. RELATED WORK

`SAML.` The Security Assertion Markup Language (SAML) was developed for the secure exchange of XML-based authentication and authorization messages. With respect to SSO, SAML can be applied within federated identity management. Unfortunately, a diversity of attacks have been discovered in the last years.

In 2003, Groß [18] analyzed the security of the SAML Browser Artifact Profile and found several adaptive attacks. Additionally, Groß analyzed the revisited version of SAML [31] and found further logical flaws [19]. Nonetheless, the attacks described by Groß (with the exception of Replay attacks) require a MitM attacker, which controls the communication channel between the client and the SaaS-CP. Such a MitM attacker is deliberately not included in our attacker capabilities and is not needed for the attacks described in this paper. Related vulnerabilities have been analyzed and found in the Liberty SSO by Pfitzmann and Waidner [30]. In 2006 Y. Chan introduced a new parallel session attack to bypass all levels of authentication by exclusively breaking the weakest one among them [8].

In 2008, Armando et al. [2] 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 in the SAML interface of Google, allowing a malicious SP to impersonate any user at any Google application. The discovered attack was used as basis for our TRC attack. Later on, the same authors identified another attack on Google's SAML interface [1]. They manipulated the `RelayState` parameter in the query string of an HTTP request in order to exploit an existing XSS vulnerability on Google.

In 2011 Somorovsky et al. reported critical security flaws regarding the XML Signature validation in the Amazon Cloud [34] allowing the impersonation of any user. In 2012 an in depth security analysis of XML Signature regarding the critical XSW attack was published [35]. As a result, the authors of the study examined 14 major SAML frameworks and showed that 11 of them had XSW

and ØSig flaws allowing the impersonation of any user.

In 2014 Morgan et al. published a whitepaper regarding the security of XML parser and the exploitation of XXE vulnerabilities [26].

`Single Sign-On.` Another study regarding the security of SSO systems has been published in 2012 by Wang et al. [40]. The authors examined REST-based authentication protocols like OpenID and found serious logic and implementation flaws resulting in identity theft. Later on, the authors developed a tool *InteGuard* detecting invariances in the communication and preventing logical flaws in SSO systems [42].

In 2012 Sun and Beznosov [37] have examined three major OAuth SSO systems (Facebook, Microsoft, and Google) and 96 OAuth SPs by analyzing the HTTP traffic going through the browser. Their empirical studies uncovered several vulnerabilities caused by implementation flaws in SPs. In 2013, Bai et al. [4] have proposed *AuthScan*, a framework to automatically extract the authentication protocol specifications from implementations. They have found multiple security flaws in several important SSO protocols (e.g. Facebook Connect, BrowserID, and Windows Live Messenger Connect). However, they have not investigated SAML implementations. In 2014 Evans et al. [43] developed a fully automated tool named *SSOScan* for analyzing the security of OAuth implementations and described five attacks, which can be automatically tested by the tool. However, none of these researches examined the security of SAML based SSO systems and none of the tools support the analysis of SAML related messages. Additionally, none of the previous papers considered security in regard to the different relevant authentication components within the target system and the relations between these components.

`Authentication in the Cloud.` In 2012 a research study considering existing attacks regarding the authentication in the Cloud [33] was published. However, this research does not provide in depth security analysis of the authentication modules within the Cloud and the relation between these modules. Additionally, attacks like TRC, CInj and XXEA are not part of this research.

11. DISCUSSION

`Authentication Components.` In this paper, we pointed out the different components responsible for authentication in the cloud. Some attacks target exactly one component, other attacks abuse the interaction between two of them. The presented attacks nevertheless targeted each component at least once. Keeping this in mind, it is important to know that SSO authentication is not just one component but many and that interaction between these components must be understood and protected thoroughly. Google and Salesforce showed that this is possible and SSO authentication can be secured by a correct technical implementation. For classical Username/Password authentication, this might not be the case, because the prohibition of weak passwords is a difficult task.

`SAML Security Survey.` While most of the presented SAML attacks are already known [29, 35, 2, 26], it is very surprising that the majority of SaaS related attacks is also already covered by previous literature: The SAML standard itself [32] even discusses a lot of the attacks in this paper, but obviously this is not clear enough. One reason for this might be that the SAML standard is very large and includes a lot of edge use-cases. We recommend to extend its security considerations to highlight the issues for different attacker types and the impact in case that a verification is not provided. As an example, the XXEA attacks require AM1, and can therefore be performed with the least knowledge of the SaaS-CP, but those attacks are not mentioned in the standard. Attacks regarding the Session Management are not considered either.

`Attack Automatism.` The evaluation results show that the described attacks are not very well understood in the wild. Only Salesforce and Google were not vulnerable. As a future work, we plan to develop an online service that automatically performs the attacks and is able to report them with sufficient details. Our current tools allow semi-automatic tests only. For a fully-automatic test, it is necessary to configure the SaaS-CP correctly so that our tools can be applied. This is similar to the SSOScan approach [43], but for SAML it is much more complicated due to the great flexibility allowed by the specification. By this means, many different token permutations are possible and should be considered during testing. Additionally, it is not just enough, to find the SSO login button, because there exist several different SAML profiles [7] and each SaaS-CP needs different parameters within the SAML token.

12. REFERENCES

[1] 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 *SEC*, pages 68–79, 2011.

[2] 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.

[3] Armando, Alessandro and Carbone, Roberto and Compagna, Luca and Cuéllar, Jorge and Tobarra, M. Llanos. SAML: CVE-2008-3891. http://www.cvedetails.com, September 2008.

[4] Guangdong Bai, Jike Lei, Guozhu Meng, Sai Sathyanarayan Venkatraman, Prateek Saxena, Jun Sun, Yang Liu, and Jin Song Dong. AUTHSCAN: Automatic extraction of web authentication protocols from implementations. *NDSS, February*, 2013.

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

[6] Bitium. Bitium Partners, 2014. [online] `https://www.bitium.com/site/apps/`.

[7] Scott Cantor, John Kemp, Rob Philpott, and Eve Maler. Profiles 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-profiles-2.0-os.pdf`.

[8] 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.

[9] Christian Mainka. Detecting and exploiting XXE in SAML Interfaces, 2014. [online] `http://web-in-security.blogspot.de/2014/11/detecting-and-exploiting-xxe-in-saml.html`.

[10] Christian Mainka, Vladislav Mladenov, Florian Feldmann, Julian Krautwald. Instructure Advisory IAC00722 - SAML Ruby gem vulnerability. https://help.instructure.com/entries/46981014-Instructure-Advisory-IAC00722-SAML-Ruby-gem-vulnerability, Feb 2014.

[11] Christian Mainka, Vladislav Mladenov, Florian Feldmann, Julian Krautwald. Multiple CVEs: VU 190556, VRF HXR9YUNY,VRF HXRAH4O0,VU 774084,VRF HXRAND04. Not published yet, 2014.

[12] Christian Mainka, Vladislav Mladenov, Florian Feldmann, Julian Krautwald. Responsible Disclosure Policy, Contributors. http://www.zendesk.com/company/responsible-disclosure-policy, Feb 2014.

[13] Christian Mainka, Vladislav Mladenov, Florian Feldmann, Julian Krautwald. SAML attacks on Canvas interface. https://help.instructure.com/entries/26920510-Instructure-Advisory-IAC44584-SAML-Signature-Wrapping, Feb 2014.

[14] Christian Mainka, Vladislav Mladenov, Florian Feldmann, Julian Krautwald. SAML attacks on Clarizen interface. http://www.clarizen.com/security-log.html, Feb 2014.

[15] Clarizen. Clarizen - The way to work, 2014. [online] http://www.clarizen.com/.

[16] CloudReviews. CloudReviews, 2014. [online] http://www.cloudreviews.com/cat/apps.html.

[17] Andreas Falkenberg, Christian Mainka, Juraj Somorovsky, and Jorg Schwenk. A New Approach towards DoS Penetration Testing on Web Services. *2013 IEEE 20th International Conference on Web Services*, 0:491–498, 2013.

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

[19] 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.

[20] Frederick Hirsch, David Solo, Joseph Reagle, Donald Eastlake, and Thomas Roessler. XML Signature Syntax and Processing (Second Edition). W3C recommendation, W3C, June 2008.

[21] ideascale. ideascale, 2014. [online] http://ideascale.com/.

[22] Instructure. Canvas, 2014. [online] http://www.instructure.com/.

[23] Michael Kay. XSL Transformations (XSLT) Version 2.0 (Second Edition). W3C proposed edited recommendation, W3C, April 2009. http://www.w3.org/TR/2009/PER-xslt20-20090421/.

[24] Michael McIntosh and Paula Austel. XML signature element wrapping attacks and countermeasures. In *SWS '05: Proceedings of the 2005 Workshop on Secure Web Services*, pages 20–27, New York, NY, USA, 2005. ACM Press.

[25] Peter M. Mell and Timothy Grance. SP 800-145. The NIST Definition of Cloud Computing. Technical report, Gaithersburg, MD, United States, 2011.

[26] Timothy D. Morgan and Omar Al Ibrahim. XML Schema, DTD, and Entity Attacks - A Compendium of Known Techniques. 2014.

[27] Ruhsan Onder and Zeki Bayram. XSLT version 2.0 is turing-complete: A purely transformation based proof. In *Implementation and Application of Automata*, pages 275–276. Springer, 2006.

[28] OneLogin. OneLogin Partners, 2014. [online] http://www.onelogin.com/partners/app-partners/.

[29] OWASP Foundation. Cross-Site Request Forgery (CSRF). https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF), 2013. [online] https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF).

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

[31] S. Cantor et al. Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0, March 2005.

[32] S. Cantor et al. Security and Privacy Considerations for the OASIS Security Assertion Markup. Language (SAML) V2.0, March 2005.

[33] Kalayan Sudia Santosh Bulusu. AStudy on Cloud Computing Security Challenges. Master's thesis, School of Computing Blekinge Institute of Technology SE-371 79 Karlskrona Sweden, January 2012.

[34] Juraj Somorovsky, Mario Heiderich, Meiko Jensen, Jörg Schwenk, Nils Gruschka, and Luigi Lo Iacono. All Your Clouds are Belong to us – Security Analysis of Cloud Management Interfaces. In *The ACM Cloud Computing Security Workshop (CCSW)*, October 2011.

[35] 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.

[36] C. M. Sperberg-McQueen, Henry S. Thompson, Murray Maloney, Henry S. Thompson, David Beech, Noah Mendelsohn, and Shudi (Sandy) Gao. W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures. Last call WD, W3C, December 2009.

[37] 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.

[38] Talkin' Cloud. Top 100 Cloud Services Providers (CSPs) List And Research, 2014. [online] http://talkincloud.com/tc100.

[39] TimeOffManager. TimeOffManager, 2014.

[40] 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.

[41] Wikipedia. Cloud computing providers, 2014. [online] http://en.wikipedia.org/wiki/Category:Cloud_computing_providers.

[42] Luyi Xing, Yangyi Chen, XiaoFeng Wang, and Shuo Chen. InteGuard: Toward Automatic Protection of Third-Party Web Service Integrations. In *NDSS*, 2013.

[43] David Evans Yuchen Zhou. Automated Testing of Web Applications for Single Sign-On Vulnerabilities. In *23rd USENIX Security Symposium (USENIX Security 14)*, San Diego, CA, August 2014. USENIX Association.

[44] Zendesk. Zendesk, 2014. [online] http://zendesk.com/.

[45] Zoho. Zoho, 2014. [online] http://www.zoho.com/.

[46] Gavin Zuchlinski. The Anatomy of Cross Site Scripting. *Hitchhiker's World*, 8, 2003.