Analysis of security vulnerabilities in Microsoft Office 365 in regard to SAML

Jennifer Jurreit, Patrik Fehrenbach, Prof. Dr. Friedbert Kaspar
Fakultät Informatik
Hochschule Furtwangen Universtiy
{jennifer.jurreit, patrik.fehrenbach, f.kaspar}@hs-furtwangen.de

***Abstract*—Microsoft Office 365 is a commonly used office application, distributed by Microsoft. Several well-known companies and education institutes, such as schools and colleges, registered their office subscriptions as federated. As it was demonstrated in April 2016, a security vulnerability in Microsoft's Office 365 Single Sign On (SSO) implementation exposed these accounts to an external threat.**

The implementation in question was an implementation of the Security Assertion Markup Language (SAML) 2.0 Web Browser SSO Profile, customized by Microsoft's developer team.

By analyzing known security vulnerabilities of SAML implementations and security risks in web-based SSO implementations, the probability of a general vulnerability in SAML has been assessed. Furthermore, the cause of the vulnerability in Microsoft's Office 365 SSO, as demonstrated by Kakavas and Bratec, has been examined.

With regard to the aforementioned analyses, it could be demonstrated that the security vulnerability in Microsoft's Office 365 SAML 2.0 Web Browser SSO Profile implementation did not emerge from a general security flaw in SAML, but from disregarding established security standards.

I. Introduction

The Security Assertion Markup Language, in short SAML, is a widely used authentication mechanism used by numerous companies and products around the world. Since it got first introduced in November 2002, it evolved across several versions to what it is right now. Over the last approximately twenty years, hundreds of functions have been implemented in SAML. Integrations of SAML are available in almost every programming language. As SAML and its integrations evolved, researches, focusing on its frameworks' security, commenced. An interesting example is "On Breaking SAML: Be Whoever You Want to be", which attempts to identify flaws in SAML implementations of commonly used frameworks [15]. One of the discussed topics in the paper is the commonly misinterpreted usage of cross domain authentication. The flaws associated with this mechanism were also discovered by Klemen Bratec and Ioannis Kakavas in December 2015, when they discovered a vulnerability in Microsoft Office 365, which allowed the bypassing of cross domain authentication. This vulnerability was caused by the implementation of the SAML Service Provider, and affected all federated domains.
In this paper the cause of the vulnerability will be assessed, answering the questions whether there is a security issue in SAML or whether Microsoft violated fundamental security mechanisms, causing a major security vulnerability.
Section II of this paper discusses related work of the topic, starting with the first published research regarding SAML security.
Section III and IV provide fundamental knowledge of WS-Trust and SAML.
In Section V known security issues of web-based Single Sign On are discussed, in order to assess the overall security of SAML in section VI.
The Microsoft Office 365 Single Sign On implementation is described and the aforementioned exploit discussed in section VII.
Section VIII concludes the paper.

II. Related Work

One of the earliest researches regarding SAML and security was published in October 2003 by Thomas Groß of IBM Zürich with the title "Security Analysis of the SAML Single Sign-on Browser/Artifact Profile" [13]. It provides a common attack-by-attack list, as well as countermeasures and security restrictions.
SAML and SOAP implementations were a common cause for devastating security issues over the last years. One famous example of a wrongly implemented SAML is the SSO (Single Sign On) plugin for the content management system and blogging platform WordPress. Due to a mistake in the implemented check of the <ds:Signature> tag, it was possible to bypass the authentication by stripping the Signature tag from the message. This example shows the importance of auditing the actual implementations of SAML endpoints. This was also demonstrated by an attack on the SOAP implementation of the e-Commerce platform Magento [20]. A privileged attacker was able to insert PHP classes through the autoloader function of PHP. This was possible due to the fact, that a SOAP message was used to insert products. This allowed the researcher to add a xsi:type="xsd:base64Binary" element into the body, which included a base64 encoded string. This string contained a ftp:// wrapper to maliciously include external PHP classes.
Lastly, the blog entry "The road to hell is paved with SAML Assertions" by Ioannis Kakavas [4] discusses the security vulnerability in Microsoft Office 365 SAML Service Provider regarding its cause and impact on Microsoft Office 365.

III. WS-Trust

WS-Trust was designed to enable applications to build and exchange trusted SOAP messages [9]. The trust system works by exchanging and brokering security tokens in a protocol agnostic way.
The WS-Trust specification provides a flexible set of procedures to support various security protocols, but does not directly describe any protocols [9].
The provided procedures include processes to request and obtain security tokens, and to establish, manage and assess trust relationships [9].
The core of each process is the Security Token Service (STS), which is a web service that declares security tokens [9]. To establish trust, a web service expects that incoming messages prove a series of claims, e.g. name, key, permissions. In case the expectation is not met, the web service rejects or ignores the message [9].
One way to present those claims is to associate security tokens with the messages. If the requestor does not have a security token to prove the required claims, it can request a token from the STS. This approach is called *brokered authentication*. Trust relationships are established between the client (also referred to as requestor) and the STS, and between the STS and the requested service (see Figure 1).

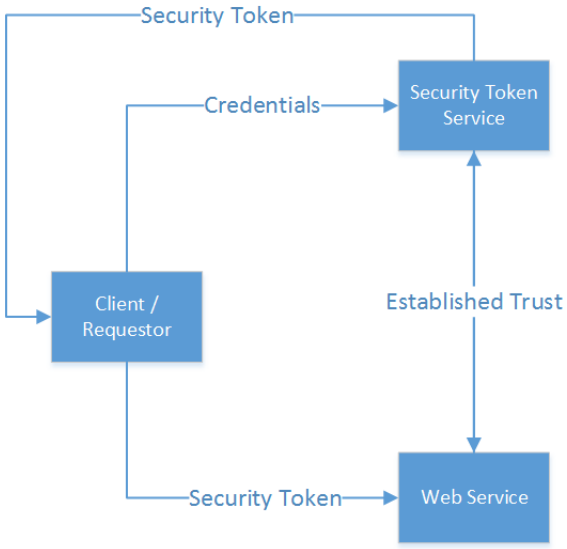

Figure 1. Brokered Authentication in WS-Trust

IV. Security Assertion Markup Language

The Security Assertion Markup Language (SAML), defined by the OASIS Security Technical Commitee (SSTC), is an XML-based, commonly used framework for authentication and authorization across platforms. It is used for sending and receiving attributes and user privileges, e.g. credentials and roles, between different platforms, without the necessity of sending critical data. Its well-known Web Browser SSO Profile was used in Microsoft Office 365 SSO implementation. In this section the fundamentals of SAML will be explained to assess its overall security.

A. Fundamentals

SAML is composed of three parties, the User Agent (user), the Identity Provider (IdP), and the Service Provider (SP).
The Service Provider corresponds to the application the user requests to use, while the Identity Provider saves all relevant user information, e.g. credentials and roles. Therefore, the IdP is also called the *SAML Asserting Party* [3].
In order to secure Single Sign On, SAML consists of four concepts, Assertions, Protocols, Bindings, and Profiles [7].
Assertions are XML structures that contain information about the user, requesting access. They are created by the IdP and consumed by the SP. To establish trust, the Assertions contain a Token the IdP created. To verify its validity, it also contains a digital signature.
There are three types of Assertions, the Authentication Statements, the Attribute Statements, and the Authorization Statements [3].
Authentication Statements inform the SP, that the user is authenticated by the IdP. They communicate when and how this authentication occurred.
The Attribute Statements are used by the SP to send information about user identifying attributes to the IdP.
Using the information received by both parties, the Authorization Statements serve to assess which resources the user is allowed to use.
Sending and receiving of Assertions between the IdP and SP typically follows a request/response pattern. This pattern is called a Protocol [7].
In SAML there are six different Protocols predefined [3].
The goal of the Authentication Request Protocol is to establish a security context between the IdP and the SP, that is needed to secure the SSO process. It is used to receive Assertions about the user from the IdP.
The Single Logout Protocol terminates all sessions of a user. It can be triggered by the user, the IdP, or the SP.
The Assertion Query and Request Protocol defines message types and rules for handling messages to request already known Assertions via reference, or to use defined search parameters to find a specific Assertion.
The Artifact Resolution Protocol is used to send references to SAML messages, and to use SAML artifacts via SAML binding. These are used to minimize the message's network load. The protocol can also be used to solve references.
To change the value of an identifier that is linked to the user, the Name Identifier Management Protocol is used.
The Name Identifier Mapping Protocol, on the other hand, enables the SP to request an identifier for the requesting user from the IdP to access another SP.
To determine how SAML messages are transported between

the IdP and SP within a protocol, Bindings are used [7]. Typical bindings are SOAP and HTTP.
Profiles combine Assertions, Protocols, and Bindings for specific use cases. The most common profile is the Web Browser SSO Profile, that is widely used for Web SSO [3].

B. SAML 2.0 Web Browser SSO

The Web Browser SSO Profile is a well-known SAML Profile for web-based Single Sign On. It combines the Authentication Request Protocol, as defined in [16] , an authentication Assertion, and Bindings. As defined by [6], the following steps are implemented by the Profile (see Figure 2).

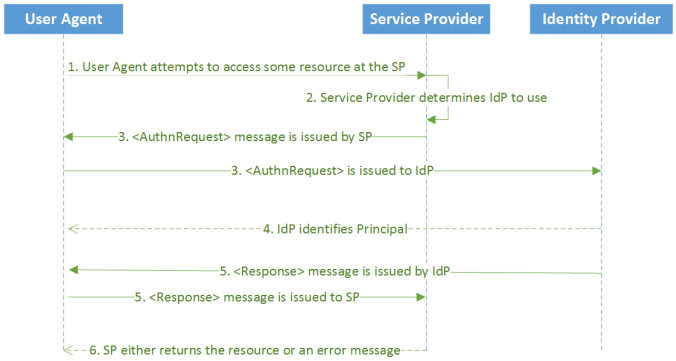

Figure 2. Web Browser SSO

First the principal makes an HTTP request to a protected resource at the SP without a security context, using an HTTP User Agent.
Then the SP determines which IdP to use for this request, based on the supported bindings.
In the third step the SP issues an <AuthnRequest> message to the IdP, which is delivered by the User Agent. For this, either an HTTP Redirect, HTTP POST or HTTP Artifact Binding is used to convey the message from the SP, through the User Agent, to the IdP.
Listing 1 shows a typical <AuthnRequest> message [10].

Listing 1. AuthnRequest message

```
<samlp:AuthnRequest
 xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
 xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
 ID="identifier_1" Version="2.0"
 IssueInstant="2004-12-05T09:21:59Z"
 AssertionConsumerServiceIndex="0">
    <saml:Issuer>
        https://sp.example.com/SAML2
    </saml:Issuer>
    <samlp:NameIDPolicy
     AllowCreate="true"
     Format="urn:oasis:names:tc:SAML:2.0:nameid-format:transient"/>
</samlp:AuthnRequest>
```

To define the SP issuing the request, an URI like string, uniquely identifying the SP, is included in the Issuer tag of the message. In addition to this, the IssueInstant value indicates when the request was issued. The given ID is an internal identifier to link the <Response> message to the request.
Examining the <AuthnRequest>, the IdP assesses whether it trusts the SP or not, identifies the principal by prompting the user for authentication (step four), and issues a <Response> message to the SP (step five). The <Response> message is issued by the IdP, but sent via the User Agent. In this case, the HTTP POST or HTTP Artifact Bindings can be used. The HTTP Redirect Binding can not be used in this context, because the response will most likely exceed the maximum URL length.
When the authentication was successful, the <Response> message looks like the example shown below (see Listing 2) [10].

Listing 2. Response message

```
<samlp:Response xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
 ID="s2cdc74f37f923e26fe1aeec42b70a93d24230334f"
 InResponseTo="90AA6073F01567BFB0DF194F596314E2"
 Version="2.0"
 IssueInstant="2010-04-29T23:21:51Z"
 Destination="https://dloomac.service-now.com/navpage.do"
>
 <saml:Issuer xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion">
    http://idp.ssocircle.com
 </saml:Issuer>
 <samlp:Status xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol">
   <samlp:StatusCode xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
    Value="urn:oasis:names:tc:SAML:2.0:status:Success">
   </samlp:StatusCode>
 </samlp:Status>
 <saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
  ID="s23e536bfc51b8487d4d3299dec162d9c2e338823b"
  IssueInstant="2010-04-29T23:21:51Z"
  Version="2.0"
 >
       <saml:Issuer>
            http://idp.ssocircle.com
       </saml:Issuer>
       <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
        ...
       </Signature>
       <saml:Subject>
         <saml:NameID Format="urn:oasis:names:tc:SAML:1.1:nameid-format:
                emailAddress"
          NameQualifier="http://idp.ssocircle.com"
          SPNameQualifier="https://dloomac.service-now.com/navpage.do">
                david.loo@service-now.com
         </saml:NameID>
         <saml:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
           <saml:SubjectConfirmationData
            InResponseTo="identifier_1"
            NotOnOrAfter="2010-04-29T23:31:51Z"
            Recipient="https://dloomac.service-now.com/navpage.do"
            />
         </saml:SubjectConfirmation>
     </saml:Subject>
     <saml:Conditions NotBefore="2010-04-29T23:11:51Z"
      NotOnOrAfter="2010-04-29T23:31:51Z"
     >
       <saml:AudienceRestriction>
          <saml:Audience>
              https://dloomac.service-now.com
          </saml:Audience>
       </saml:AudienceRestriction>
     </saml:Conditions>
     <saml:AuthnStatement AuthnInstant="2010-04-29T23:21:51Z"
      SessionIndex="s2dbf89ab99001e0e8cdaed67266d9d4b21b968a04"
     >
      <saml:AuthnContext>
        <saml:AuthnContextClassRef>
            urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport
        </saml:AuthnContextClassRef>
      </saml:AuthnContext>
     </saml:AuthnStatement>
 </saml:Assertion>
</samlp:Response>
```

To link the <Response> message to the issued request the InResponseTo value refers to the ID value shown above (see Listing 1).
IssueInstant, NotOnOrAfter and NotBefore define when the message is valid to protect against replay attacks.
To verify that the message is sent from the right IdP, the Issuer field is provided in the <Response> message.
To prevent the message to be used by a foreign SP, the AudienceRestriction element is provided.
The Subject element identifies the authtenticated principal, and the AttributeStatement contains user specific attributes

and their values.
Lastly, the SP either grants access to the principal or denies it, depending on the received response from the IdP.
In case the first request of the User Agent is not sent to the SP, but directly to the IdP, the process starts at the fifth step and ignores all prior steps.

V. Security Risks of Web-based Single Sign On via SAML

The SAML web-based SSO procedure, as described in Section IV, has been used for approximately twenty years. During this time, two essential security vulnerabilities have been discovered in many SAML implementations. These vulnerabilities are explained in this section.

A. XML Signature Wrapping

SAML and any other authentication based mechanisms, such as SOAP, are highly flexible, when it comes to deploying integrity features [17]. To establish a trust model between message deliveries, the integrity has to be validated. For that reason, several parts of SOAP and SAML messages use cryptographic signatures. In most cases, these signatures include a X509 Certificate with Block Encryption Algorithms of AES (Advanced Encrpytion Standard)-128, AES-256 or Triple DES (Data Encryption Standard), represented by ASN.1 (Abstract Syntax Notation One).
In a hypothetical scenario a web service receives a signed message by a client, validates the signature and processes the data. A malformed request should not be accepted by the web service in any case. However, the XML Signature Wrapping attack, also known as the XML Rewriting attack, allows an attacker to change the content of the signature part without invalidating the signature [17].

Simple Context. One instantiation of the XML Signature wrapping is the simple context attack methodology. In a signature process the data is embedded within the document body and is referenced via an *ID* attribute. The data is comprised in a so called Simple Ancestry Context. During the validation, the XML parser references the element and verifies the signature. As a result, it returns a boolean value. In case of a *true* statement the service processes the SOAP request, and in case of a *false* return value it rejects the message. If the validation logic works correctly, no manipulation is possible. However, it is possible to bypass the logic by changing the signed data without invalidating the signature. This is possible due to the fact, that the verification algorithm starts by looking for the Body element, which is stated in the <Reference> element. The referenced element may find the <soap:header> element within the <wrapper> element. The real <soap:Body> outside of the <soap:Header> gets ignored, since it has the wrong *ID* element [17]. Although the context of the message is malformed, the signature verification will still process the <soap:header> within the <wrapper> element, since the signature still contains the original SOAP Body, that was signed by the sender of the message. The SOAP message is then parsed to the application logic, that processes the SOAP body and ignores all other SOAP elements.

B. XXE Attacks

One major aspect of SAML-based authentication is the usage of XML (Extensible Markup Language) for request and response messages. XML is one of the oldest formats used in the world wide web. It was first introduced, almost twenty years ago, by the W3C Consortium. Back then, XML was seen as a simple text-based format for representing structured information. As the usage of XML and overlaying file formats increased, the W3C introduced a new functionality, called external entities. In theory, entities are used to assign names to reusable chunks of text, which can be declared by entities in XML data. One powerful aspect of the described entities is the ability to include foreign sources into XML. This functionality allows developers to outsource XML information, and, if needed, include them without having to store them on the same device. This functionality, although helpful in some cases, leads to the possibility of an attack, which is called XXE (External Entity Attack). In this kind of attack, an attacker creates a specially crafted XML document, containing a reference to an external entity, which is processed by an ill-configured XML parser. The fundamental problem with parsers is, that external entity processing is enabled by default. So whether it is used or not, the parser processes the entities. The security risk, however, is limited to requests, where attackers are able to interact with any kind of XML requests.
A typical attack scenario is demonstrated subsequently:

Listing 3. XXE Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE testingxxe [<! ENTITY name System "c:\boot.ini">]>
<Prod>
   <Prod>
      <Type>Hochschule Furtwangen
      </type>
      <name>XXE Test</name>
      <id>&name</id>
   </Prod>
</Prod>
```

As seen in the listing above, an additional doctype and entity element has been set to the original request. This request will be processed, if external entities are not disabled. The parser requests the external entity and processes the *SYSTEM* namespace, which requests the file *c :boot.ini*. Once the file is requested, it is referenced with the name attribute at the response context.
Figure 3 shows the attack flow of the XXE scenario. Using this method an attacker is capable to exfiltrate sensitive files from the server by abusing external entities. This attack, however, only works, if the XML output is reflected somewhere in the page content. If there is no direct reflection of XML, a verbose error message discloses the attack.

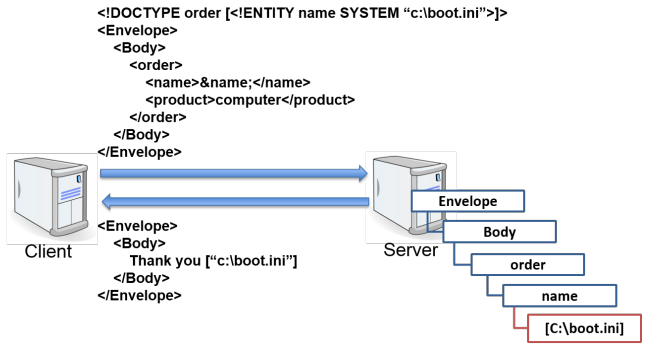

Figure 3. XXE Attack Example [14]

C. XXE OOB

A fairly new technique, called XXE OOB (External Entity Attack Out of Bounds), allows an attacker to exploit certain scenarios wherein there is no direct reflection of the XML output. This works by defining an external entity on a remote destination.

Listing 4. XXE OOB Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE xxeoob [
<!ENTITY % dtd SYSTEM "http://141.28.68.243:8080/payload.dtd" >
 %dtd;]>
<Prod>
   <Prod>
      <Type>Test
      </type>
      <name>Hochschule Furtwangen</name>
      <id>21</id>
   </Prod>
</Prod>
```

This attack works by defining a new document type and declaring an additional dtd (Document Type Definition) file on a remote server. A dtd documment represents a structure of legal elements and attributes of XML documents [19]. By adding the external entity on a remote server, the XML parser follows the link and processes the document type. In this attack scenario, the document type contains the following instructions:

```
<!ENTITY % data SYSTEM "file:///etc/passwd">
<!ENTITY % param1 "<!ENTITY exfil SYSTEM 'http://141.28.68.243:/?%data;'>">
```

The instructions execute *exfil* in the *SYSTEM* namespace to get the content of the well-known unix file etc/passwd. In a second step, the content of the file is appended as an URI to the server the attacker controls. The web server is receiving the content of the file. By using this technique an attacker is capable to exfiltrate files from the server.

VI. Security Assessment of SAML

The SAML 2.0 Web Browser SSO Profile is widely used in Single Sign On applications across the web. It has been implemented in various tools and open source libraries, such as OpenSAML [12].
Over the years, a few security vulnerabilities have been found in a series of SAML implementations (cf. Section V).
Whereas XXE is not directly affecting SAML, but XML, it does demonstrate the risks that must be considered in SAML implementations. XSW, on the other hand, is directly affecting SAML and has been found in many SAML tools and libraries. OpenSAML, for example, has been vulnerable to both attacks, XXE and XSW, in the past [11],[13].
The fact that some vulnerabilities have been found in example code, provided by OpenSAML, shows how difficult the implementation of SAML 2.0 Web Browser SSO can be [11]. However, while vulnerabilities have been detected in the past, solutions have also been presented [2], [8], [5].
Various considerations, concerning SAML security, can be found online, for example in [2] and [8].
While assessing an implementation of SAML several questions should be asked in order to ensure a correctly working SAML mechanism. These should include [18]:

- If an SAML message is signed, what happens if the content is changed?
- If it is accepted, what happens if the signature is removed?
- Does it accept re-signing with a different certificate?

Following the instructions in [2] and [8], and considering the aforementioned questions, leads to an overall secure implementation of SAML.
Therefore, SAML can be considered secure. To avoid vulnerabilities in implementations developers have to meticulously follow the described good practice.

VII. Single Sign On in Microsoft Office 365

In Microsoft Office 365 Microsoft used a combination of WS-Trust and SAML 2.0 Web Browser SSO to implement Single Sign On on the Office platform.
As discussed in [4], this implementation allowed malicious users to exploit the cross domain authentication (cf. Section I). This section discusses the implementation of WS-Trust and SAML as used for the platform's Single Sign On.

A. SAML Implementation

In order to use Single Sign On, the user must be registered in Azure AD for the specific tenant. The authentication process starts with a request to the SP by accessing the Office 365 portal and sending an authentication request to the portal. The SP then checks, whether the domain, provided by the user, is known and registered as federated in Azure AD, and sends an <AuthnRequest> to the IdP (see Listing 5).

Listing 5. AuthnRequest message from Microsoft Office 365 [4]

```
<samlp:AuthnRequest xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
ID="_f6daef39-fb54-407e-abb4-c75d261b75ae"
IssueInstant="2016-04-11T21:13:44Z"
Version="2.0"
AssertionConsumerServiceIndex="0"
>
```

```
  <saml:Issuer>urn:federation:MicrosoftOnline</saml:Issuer>
  <samlp:NameIDPolicy Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent
      " />
</samlp:AuthnRequest>
```

When the user has sent an authentication request to the IdP, the IdP identifies the user and sends a <Response> message back to the SP. To identify the user, the following attributes are released to the SP by the IdP within the <Response> message (see Listing 6):

- The IDPEmail, containing the domain of the user's IdP
- An ImmutableId to uniquely identify the user

Listing 6. Response message from Microsoft Office 365 [4]

```
<saml2p:Response Destination="https://login.microsoftonline.com/login.srf"
ID="_cefc5f992f2e455d7b3e52522fc479db" InResponseTo="_f6daef39-fb54-407e-abb4-
    c75d261b75ae"
IssueInstant="2016-04-11T21:14:35.365Z" Version="2.0" xmlns:saml2p="urn:oasis:
    names:tc:SAML:2.0:protocol"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
>
  <saml2:Issuer xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion">
    https://idp.admin.grnet.gr/idp/shibboleth
  </saml2:Issuer>
  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
      [...]
      <saml2:Subject>
          <saml2:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:
              persistent">
              This is where my ImmutableId is
          </saml2:NameID>
          <saml2:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer
              ">
              <saml2:SubjectConfirmationData Address="2a02:214d:811b:7200:dc15:
                  b7bc:a304:3738" InResponseTo="_f6daef39-fb54-407e-abb4-
                  c75d261b75ae"
               NotOnOrAfter="2016-04-11T21:19:35.384Z" Recipient="https://login.
                  microsoftonline.com/login.srf"
               />
          </saml2:SubjectConfirmation>
      </saml2:Subject>
      <saml2:Conditions NotBefore="2016-04-11T21:14:35.365Z" NotOnOrAfter="
          2016-04-11T21:19:35.365Z">
          <saml2:AudienceRestriction>
              <saml2:Audience>
                urn:federation:MicrosoftOnline
              </saml2:Audience>
          </saml2:AudienceRestriction>
      </saml2:Conditions>
      <saml2:AuthnStatement
       AuthnInstant="2016-04-11T21:14:35.027Z"
       SessionIndex="_91380d0480ac9af6bcb19f3b26f0ea81"
      >
          <saml2:SubjectLocality Address="2a02:214d:811b:7200:dc15:b7bc:a304:3738"
              />
          <saml2:AuthnContext>
              <saml2:AuthnContextClassRef>
                urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport
              </saml2:AuthnContextClassRef>
          </saml2:AuthnContext>
      </saml2:AuthnStatement>
      <saml2:AttributeStatement>
          <saml2:Attribute FriendlyName="IDPEmail"
           Name="IDPEmail"
           NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"
          >
              <saml2:AttributeValue xmlns:xsi="http://www.w3.org/2001/XMLSchema-
                  instance"
               xsi:type="xsd:string"
              >
                ikakavas@mymail.example.com
              </saml2:AttributeValue>
          </saml2:Attribute>
      </saml2:AttributeStatement>
  </saml2:Assertion>
</saml2p:Response>
```

B. WS-Trust Implementation

The implementation of WS-Trust is not isolated from the Web Browser SSO implementation. The IdP, as implemented for the Web Browser SSO, simultaneous works as a Security Token Service by internally converting SAML messages to WS-Trust messages [4]. Therefore, the WS-Trust token is not generated explicitly. Instead, the SAML message is merely translated into a WS-Trust token, which leads to a crucial flaw in this implementation (cf. Section VII-C).

C. Flaws in Microsoft Office 365 Implementation

The implementation, as it is described above, shows several deficits from a security perspective.

Even though the ImmutableId within the Subject element of the <Response> message uniquely identifies the user issuing the authentication request, the Service Provider disregards the Subject of the Assertion upon arrival.

Therefore, the user can only be identified via the IDPEmail, provided within the AttributeStatement element. Whereas the SP uses the Issuer element to verify the SAML Response by retrieving the corresponding certificate, it does not check the value of the IDPEmail attribute against the IdP, that issued the original request. Thus, neither the token content nor the IDPEmail is checked against the IdP, enabling the exploitation of the cross domain authentication [4].

Additionally, the obtained security token is forwarded to WS-Trust, in case the user chooses a different SSO method than SAML SSO. Therefore, the vulnerability's scope exceeds users federating their domains via SAML, including users federating their domains via Active Directory Federations Services (ADFS) [4].

D. Exploitation

The above mentioned flaws lead to a serious vulnerability that can be exploited by copying the victim's email to another Office 365 subscription [4] and changing the provided user credentials at the second login form.

The victim's email address must be known to the malicious Identity Provider to successfully log in to the account, when prompted with a login form. Thus, in a first step, the email must be copied to the directory of the attacker's Office 365 subscription. Figure 4 shows the process of the attack after the prerequisites are met.

First, the attacker requests access to the Office 365 portal through a web browser. Using the attacker's domain, something@attacker.com, to log in, he is redirected to the attacker's Identity Provider and prompted with another login request.

At this point, the attacker can use any user credentials that are known to the IdP to log in. Therefore, the attacker can change the previously set email (something@attacker.com) to something@victim.com and log in successfully.

Having logged in successfully, the attacker is redirected to the victim's Office 365 portal, where he has access to the victim's files, account information, and Office applications.

This attack is successful, because the provided email addresses used for the first and second login are not evaluated against each other (cf. Section VII-C).

E. Assessment of the vulnerability

Having analyzed the flaws in Microsoft's Office 365 SAML implementation and discussed an exploit, resulting from these, this section focuses on the assessment of the aforementioned vulnerabilities.

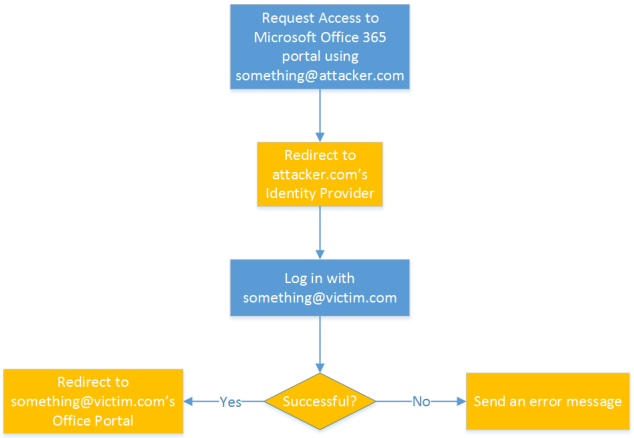

Figure 4. Flowchart of the attack (The attacker's actions are marked blue, and the application's processes are marked yellow)

Although a few security vulnerabilities in SAML have been discovered in the past (cf. Section V), the vulnerabilities in Microsoft's Office 365 implementation did not originate from any general SAML vulnerabilities.

Furthermore, the flaws in the implementation described above (cf. Section VII-C) could have been avoided by following well-known security standards, that are described in [8].

First Mile Considerations [2] were disregarded, because the IdP was not validated.

Furthermore Last Mile Considerations [2] were disregarded, because the user identity obtained from the SAML ticket assertions was not verified by the Service Provider.

VIII. Conclusion

SAML is used in various companies and products as a Single Sign On solution. When security vulnerabilities were found in the Microsoft Office 365 SAML implementation, it raised the question whether this vulnerability was caused by SAML or Microsoft's implementation.

Taking a closer look at SAML implementations and its history of known security vulnerabilities, it could be assessed that SAML can be considered secure, if established security standards are met.

In addition to this, it could be shown, that Microsoft's implementation of a custom SAML 2.0 Web Browser SSO has been disregarding some of these security standards, such as validation of the security token against the Identity Provider. Furthermore, the combination of SAML with WS-Trust caused an even more serious vulnerability by extending the scope of the attack. Because WS-Trust tokens were simply translated from SAML messages, the vulnerability in the SAML implementation did affect all federated accounts of Microsoft Office 365.

In conclusion, it could be assessed, that the cause of the security vulnerability in Microsoft Office 365 lay exclusively in a flawed implementation of SAML 2.0 Web Browser SSO, and was not related to any general flaws in the SAML specification.

References

[1] Benjamin Sanno *https://www.nds.rub.de/media/ei/arbeiten/2014/12/04/SAMLAttacker.pdf*, Ruhr Universität Bochum. Ruhr Unverstität Bochum 6, 2013.

[2] B. Broulik, P. Krawczyk, G. Peterson, and J. McGovern, "Saml security cheat sheet - owasp," 31.01.2017. [Online]. Available: *https://www.owasp.org/index.php/SAML_Security_Cheat_Sheet*

[3] D. Krafzig and M. Yunus, "Anwendungssicherheit saml – eine einführung - jaxenter," 2015. [Online]. Available: *https://jaxenter.de/anwendungssicherheit-saml-eine-einfuehrung-18066*

[4] I. Kakavas, "The road to hell is paved with saml assertions," 06.06.2016. [Online]. Available: *http://www.economyofmechanism.com/office365-authbypass.html*

[5] MCINTOSH, M., AND AUSTEL, P. *XML Signature Element Wrapping Attacks and Countermeasures.* In SWS '05: Proceedings of the 2005 workshop on Secure web services (New York, NY, USA, 2005) ACM Press, pp. 20–27

[6] OASIS, "Profiles for the oasis security profiles for the oasis security assertion markup language (saml) v2.0," 15.03.2005. [Online]. Available: *https://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf*

[7] OASIS, "Security Assertion Markup Language (SAML) V2.0 Technical Overview," 09.10.2006. [Online]. Available: *http://www.oasis-open.org/committees/documents.php?wg_abbrev=security*

[8] OASIS, "Security and Privacy Considerations for the OASIS Security Assertion Markup Language (SAML) V2.0," 15.03.2005. [Online]. Available: *http://docs.oasis-open.org/security/saml/v2.0/*

[9] OASIS, "Ws-trust 1.4," 25.04.2012. [Online]. Available: *http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/errata01/os/ws-trust-1.4-errata01-os-complete.html#_Toc325658922*

[10] ServiceNow, "Servicenow wiki:saml 2.0 web browser sso profile," 23.01.2017. [Online]. Available: *http://wiki.servicenow.com/index.php?title=SAML_2.0_Web_Browser_SSO_Profile#4._Validate_SAMLResponse&gsc.tab=0*

[11] SendSafely, "Web-based single sign-on and the dangers of saml xml parsing." [Online]. Available: *https://blog.sendsafely.com/web-based-single-sign-on-and-the-dangers-of-saml-xml-parsing*

[12] Shibboleth Consortium, "Opensaml-java," 06.01.2017. [Online]. Available: *https://shibboleth.net/products/opensaml-java.html*

[13] Somorovsky,Mayer; Kampmann,Jensen, *On Breaking SAML: Be Whoever You Want to Be*, IBM Zurich Research Laboratory. Usenix Security Symposium 12, 2003.

[14] Somorovsky, Juraj, "Webservices und Single Sign-On: XML-basierte Angriffe im Überblick" 29.06 2015 [Online]. Availabe: *https://www.informatik-aktuell.de/betrieb/sicherheit/webservices-und-single-sign-on-xml-basierte-angriffe-im-ueberblick.html*

[15] Thomas Groß, *Security Analysis of the SAML Single Sign-on Browser/Artifact Profile*, IBM Zurich Research Laboratory. Usenix Security Symposium 12, 2012.

[16] T. Hardjono, H. Lockhart, and S. Cantor, "Oasis security services (saml) tc — oasis." [Online]. Available: *https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security*

[17] "XML Signature Wrapping " 23.12.2015 [Online]. Available: *http://www.ws-attacks.org/XML_Signature_Wrapping*

[18] "Bypassing SAML 2.0 SSO with XML Signature Attacks"30.11.2016 [Online]. Available: *http://research.aurainfosec.io/bypassing-saml20-SSO/*

[19] "DTD Tutorial" [Online]. Available: *https://www.w3schools.com/xml/xml_dtd_intro.asp*

[20] "Autoloaded File Inclusion in Magento SOAP API" [Online]. Available: *http://blog.mindedsecurity.com/2015/09/autoloaded-file-inclusion-in-magento.html*