title: 'On Breaking SAML Be Whoever You Want to Be ' date: 2017-05-18 23:54:43

tags: ‘web’

原文连接：https://www.owasp.org/images/2/28/Breaking*SAML*Be*Whoever*You*Want*to_Be_-_Juraj_Somorovsky+Christian_Mainka.pdf

打破SAML：成为任何你想成为的人

0 摘要

安全声明标记语言是一个被采用的语言，用于制作相关安全声明。它是开发联合身份部署和单点登录场景的关键组件。为了保护交换的SAML脚本工程的完整性和真实性，应用了XML签名标准。然而，签名验证算法比PKCS＃7这样的传统签名格式复杂得多。因此，在弱对抗模式下，通过应用不同的XML签名特定攻击，可以成功地规避完整性保护。

在本文中，我们描述了对14个主要SAML框架的深入分析，并显示其中包括Salesforce，Shibboleth和IBM XS40 在内的其中11个具有关键的XML签名包装（XSW）漏洞。基于我们的分析，我们开发了一种用于XSW在SAML框架中的自动渗透测试工具。通过额外发现新的XSW变体证明了其可行性。我们提出了第一个分析这种攻击的框架，这是基于依赖方的两个组成部分之间的信息流。令人惊讶的是，这种分析还产生了有效和实用的对策。

1 介绍

安全声明标记语言（SAML）是一种基于XML的语言，用于制作关于主题的安全声明。SAML声明在WS-Security和基于REST的单点登录（SSO）场景中被用作安全令牌。主要软件供应商和开源项目都支持SAML，并被广泛部署。 由于其灵活性和广泛的支持，不断定义新的应用场景。

SAML协议 由于SAML断言包含关于主题的安全关键声明，因此这些声明的有效性必须经过认证。 根据标准，这应该通过使用XML签名来实现，该签名应该覆盖完整的SAML断言或包含它的XML文档（例如，SAML验证响应）。

然而，我们评估的大约80％的SAML框架可以通过用新的XML签名包装（XSW）攻击规避完整性保护来破坏。 这个令人惊讶的结果主要是由于两个事实：

- 复杂签名算法：PKCS＃7和OpenPGP之前的数字签名数据格式计算整个文档的单个散列值，签名简单地附加到文档中。XML签名标准要复杂得多。 特别是签名和签名内容的位置是可变的。因此，存在同一XML文档的多种排列。
- 未指定的内部接口：大多数SAML框架将依赖方（即，Web服务或网站使用SAML声明）视为单个块，假

设所有任务都具有共同的共同状态。然而，逻辑上，该块必须被细分为执行加密操作的签名验证模块（后面称为RPsig），以及处理SAML中包含的声明的SAML处理模块（稍后称为RPclaims）。 两个模块对声明有不同的看法，它们通常只交换一个关于签名有效性的布尔值。

贡献

在本文中，我们对14个SAML框架和系统进行了深入的分析。在此分析中，我们在其中的11个框架中发现了XSW漏洞鉴于SAML在实践中的重要性，这一结果令人震惊，特别是由于SSO框架可能会成为一个单一的攻击点。 它清楚地表明，SAML和XML签名背后的安全隐患尚未被理解。

第二，这些漏洞可以被攻击者所利用，方法远远少于基于传统网络加密技术的攻击：即使攻击者不控制网络也可能会成功。他不需要实时窃听，但可以利用那些SAML声明已过期的。单一签名的SAML足以完全危及SAML发行者/身份提供者。使用SSL/TLS加密SAML声明，从而防止对手通过拦截网络流量来学习声明，这也不利于：对手可能会在SAML发行人注册为普通客户，并可以使用自己的声明来冒充其他客户。

第三，我们给出了考虑到RPsig和RPclaim之间的接口的SAML框架的第一个模型。 这个模型给出了对SAML的成功攻击的明确定义。除了它的理论兴趣之外，它也使我们能够证明几个积极的结果。这些结果是新的，有助于解释为什么一些框架不容易受到我们的攻击，并就如何提高其他11个框架的安全性提供建议。

最后，我们展示XSW漏洞构成了一个重要而广泛的攻击向量。防范XSW攻击并不容易：因为与普遍的看法相反，即使是签署整个文件也不一定能够保护他们。要建立工作防御，需要更好地了解这个多功能的攻击类。在研究过程中开发的一种专门用来测试XSW的工具将作为开放源码发布，以帮助您了解这一点。 通过在Salesforce SAML界面上发现一个新的攻击向量来证明其实用性，尽管已经应用了特定的应对措施。

责任公开

我们分析过程中发现的所有漏洞均报告给负责的安全小组。因此，在许多情况下，我们与他们密切合作，以补救发现的问题。

大纲

本文的其余部分安排如下。第2节给出了SAML的高级概述，第3节增加了细节。 调查的方法在第4节中进行了说明，详细的结果在第5节中进行了说明。在第6节中，我们介绍了第一个对SAML进行全面自动化的XSW渗透测试工具。 第7节进行正式分析，并得出两个对策。在第8节，我们讨论他们的实际可行性。 第9节概述了相关工作。在最后一节，我们总结并提出未来的研究方向。

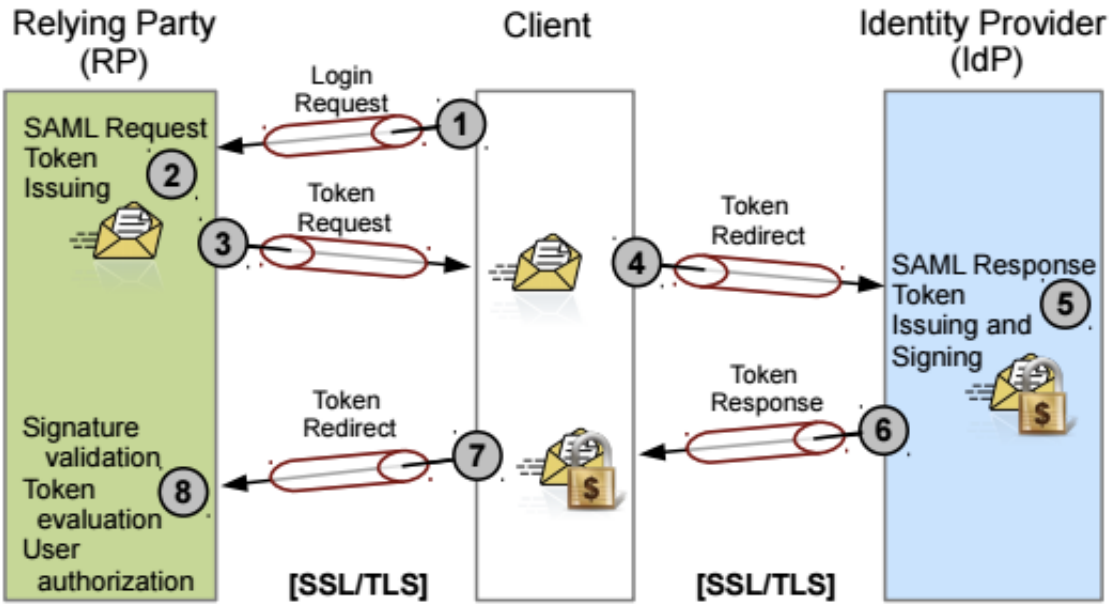


图1：典型的单点登录方案：用户访问生成请求令牌的RP。 他将这个令牌重定向到IdP。 发出的令牌发送给用户并转发给RP。 即使通道由SSL / TLS保护，用户仍然可以看到令牌。

2 动机

在本节中，我们介绍两种典型的SAML使用场景和一些广泛使用的SAML框架。

基于SAML的单点登录

典型的互联网用户必须管理不同Web应用程序的许多身份。 为了克服这个问题，单点登录被开发了。 在这种方法中，用户仅向可信赖的身份提供者（IdP）认证一次。 用户成功登录后，IdP根据需要分发安全令牌。 这些令牌用于向依赖方（RP）进行身份验证。

简单的单点登录方案如图1所示。 在此设置中，IdP登录的用户首先访问所需的RP（1）。 RP发出令牌请求（2）。 该令牌发送给将其转发给IdP（4）的用户（3）。 IdP向用户发出令牌响应，包括几个声明（例如他的访问权限或到期时间）。 为了保护要求的真实性和完整性，令牌被签名（5）。 随后，将令牌发送给将其转发到RP（7）的用户（6）。 如果用户被授权（8），RP验证签名然后授权访问受保护的服务或资源。该访问控制决定基于验证令牌中的权利要求。

用SAML保护网络服务

另一个典型的应用场景是在SOAP [21]中使用SAML与WS-Security [29]一起向Web服务提供认证和授权机制。 SAML断言作为安全性标记包含在安全性头文件中。

SAML提供者和框架。

本文提出的评估是在过去18个月内进行的，包括突出而且使用较为有效的SAML框架，总结在表1中。我们的

分析包括应用于XML安全网关的IBM硬件设备XS40。

Framework/Provider	Type	Language	Reference	Application
Apache Axis 2	WS	Java	http://axis.apache.org	WSO2 Web Services
Guanxi	Web SSO	Java	http://guanxi.sourceforge.net	Sakai Project (www.sakaiproject.org)
Higgins 1.x	Web SSO	Java	www.eclipse.org/higgins	Identity project
IBM XS40	WS	XSLT	www.ibm.com	Enterprise XML Security Gateway
JOSSO	Web SSO	Java	www.josso.org	Motorola, NEC, Redhat
WIF	Web SSO	.NET	http://msdn.microsoft.com	Microsoft Sharepoint 2010
OIOSAML	Web SSO	Java, .NET	http://www.oiosaml.info	Danish eGovernment (e.g. www.virk.dk)
OpenAM	Web SSO	Java	http://forgerock.com/openam.html	Enterprise-Class Open Source SSO
OneLogin	Web SSO	Java, PHP, Ruby, Python	www.onelogin.com	Joomla, Wordpress, SugarCRM, Drupal
OpenAthens	Web SSO	Java, C++	www.openathens.net	UK Federation (www.eduserv.org.uk)
OpenSAML	Web SSO	Java, C++	http://opensaml.org	Shibboleth, SuisseID
Salesforce	Web SSO	—	www.salesforce.com	Cloud Computing and CRM
SimpleSAMLphp	Web SSO	PHP	http://simplesamlphp.org	Danish e-ID Federation (www.wayf.dk)
WSO2	Web SSO	Java	www.wso2.com	eBay, Deutsche Bank, HP

另一个封闭源代码框架的例子是Microsoft Sharepoint中使用的Windows Identity Foundation（WIF）和Salesforce云平台。重要的开源框架包括OpenSAML，OpenAM，OIOSAML，OneLogin和Apache Axis 2. OpenSAML例如用于Shibboleth和来自瑞士（SuisseID）的电子身份证的SDK。 OpenAM，以前称为SUN OpenSSO，是一种身份和访问管理中间件，用于大型企业。 OIOSAML框架是例如。用于丹麦公共部门联合会（如电子商务和公民门户网站）。 OneLogin Toolkit将SAML集成到各种受欢迎的开源Web应用程序，如Wordpress，Joomla，Drupal和SugarCRM。此外，这些工具包由许多OneLogin使用 客户（例如Zendesk，Riskonnect，Zoho，KnowledgeTree和Yammer）来启用基于SAML的SSO。 Apache Axis2是用于生成和部署Web Service应用程序的标准框架。

3 技术基础

在本节中，我们简要介绍SAML标准和XML签名包装攻击。 此外，对于不熟悉相关W3C标准的读者，我们提供XML签名[14]和XML模式[36]。

3.1 XML签名

XML签名标准[14]定义了用于创建，表示和验证基于XML的数字签名的语法和处理规则。 可以签署整个XML树或只有特定的元素。 一个XML签名可以覆盖几个本地或全球资源。 签名内容中的签名称为包络签名。 如果签名包围签名的部分，它是一个包络签名。 分离的签名既不在签名数据的内部也不在父进程。

```
<Signature ID?>
    <SignedInfo>
        <CanonicalizationMethod/>
        <SignatureMethod/>
        (<Reference URI? >
            (<Transforms>)?
            <DigestMethod>
            <DigestValue>
        </Reference>)+
    </SignedInfo>
    <SignatureValue>
    (<KeyInfo>)?
    (<Object ID?>)*
</Signature>
```

XML签名由 `Signature` 元素表示。图2提供了其基本结构。XML签名是双程签名：资源的哈希值 （`DigestValue`） 以及所使用的哈希算法 （`DigestMethod`） 和对资源的URI引用都存储在参考元素中。另外，Transforms元素指定了在消化资源之前应用的处理步骤。每个有符号资源由 `SignedInfo` 元素中的一个 `Reference` 元素表示。因此， `SignedInfo` 是哈希值和URI的集合。 `SignedInfo` 本身由签名保护。 `CanonicalizationMethod` 和 `SignatureMethod` 元素指定用于规范化和签名创建的算法，并且还嵌入在 `SignedInfo` 中。 计算签名的Base64编码值存放在 `SignatureValue` 元素中。此外， `KeyInfo` 元素有助于签名相关密钥管理信息的传输。对象是可选的元素，可以包含任何数据。

```
<saml:Assertion Version ID IssueInstant>
    <saml:Issuer>
    <ds:Signature>?
    <saml:Subject>?
    <saml:Conditions>?
    <saml:Advice>?
    <saml:AuthnStatement>*
    <saml:AuthzDecisionStatement>*
    <saml:AttributeStatement>*
</saml:Assertion>
```

3.2 XML模式

W3C推荐的XML Schema [36]是描述XML文档的布局，语义和内容的语言。 当符合特定模式时，文档被视为有效。模式由内容模型，词汇表和已使用的数据类型组成。内容模型描述文档结构和项目的关系。该标准提供了19个基本数据类型来定义元素和属性的允许内容。

根据我们对基于SAML的XML签名包装攻击的评估，XML Schema中有一个重要的元素定义。任何元素允许在声明的内容类型中使用任何格式正确的XML文档。当XML处理器验证由任何元素定义的元素时， `processContents` 属性指定了灵活性级别。值lax指示模式验证器检查给定的命名空间。 如果没有模式信息可用，内容被认为是有效的。 在 `processContents="skip"` 的情况下，XML处理器根本不会验证该元素。

3.3 SAML

SAML是用于交换关于主题的认证和授权语句的XML标准[11]。[10]中定义了几个配置文件。最重要的配置文件是浏览器SSO配置文件，它定义了如何将SAML与Web浏览器配合使用。

SAML断言具有图3所示的结构。断言的发布时间在saml中是特定的： `IssueInstant` 。所有属性都是必需的。saml：Issuer元素指定在断言中提出声明的SAML权限（IdP）。主张的主旨：主体定义了关于所有声明内的所有声明的主体。saml：*Statement元素用于指定与SAML断言的上下文相关的用户定义语句。

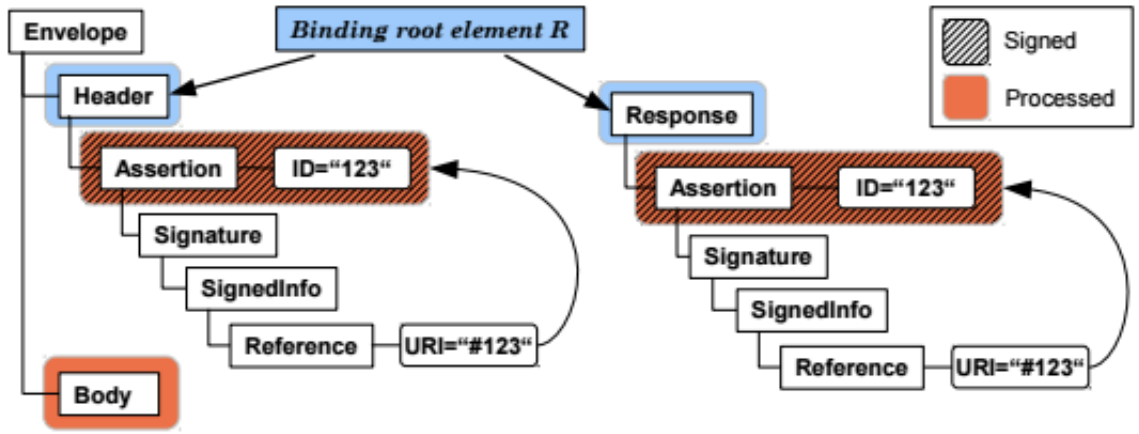


Figure 4: SAML message examples (SOAP and REST): The SAML assertion is put into a root element *R* and signed using an enveloped signature. When signing the SOAP body, an additional detached signature is used.

为了保护发行人提供的安全声明的完整性，整个saml： `Assertion` 元素必须使用XML签名标准之后的数字签名进行保护。 因此，SAML规范[11]要求saml： `Assertion` 元素或祖先元素必须由 `Signature` 元素引用，并带有包络的XML签名（[11]，第5.4.1节））。 此外，必须使用基于Id的引用（[11]，第5.4.2节），这为XSW攻击打开了道路.

在基于REST的框架中，SAML断言通常放在一个包络的Response元素中。 应用SOAP的框架将SAML断言插入SOAP标头（或SOAP标头内的Security元素）。 为了说明目的，请考虑使用包络XML签名对SAML断言进行签名，并将其放入一些绑定根元素R（参见图4）。

3.4 XML签名包装攻击

包含XML签名的XML文档通常在两个独立的步骤中进行处理：签名验证和函数调用（业务逻辑）。如果两个模块对数据都有不同的观点，则存在一类名为XML签名包装攻击（XSW）[27,23]的漏洞。如果两个模块对数据都有不同的观点，则存在一类名为XML签名包装攻击（XSW）[27,23]的漏洞。这种改变的目的是改变消息，使得应用程序逻辑和签名验证模块使用消息的不同部分。因此，接收者验证XML签名成功，但应用程序逻辑处理虚假元素。攻击者因此规避了XML签名的完整性保护和源认证，并可以注入任意内容。图5显示了对SOAP消息的简单XSW攻击。

XSW攻击类似于其他类型的注入攻击（如XSS或SQLi）：在所有情况下，攻击者会尝试强制对安全模块（例如Web应用程序防火墙）和数据处理模块（HTML解析器，SQL引擎）中的数据的不同视图。

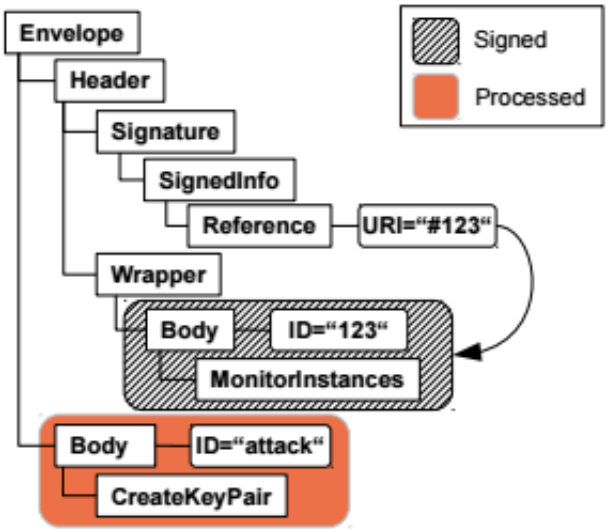


Figure 5: A simple XML Signature wrapping attack: The attacker moves the original signed content to a newly created `Wrapper` element. Afterwards, he creates an arbitrary content with a different `Id`, which is invoked by the business logic.

4 基于SAML的XSW攻击

4.1 威胁模型

作为先决条件，攻击者需要任意签名的SAML消息。这可能是单个断言A或具有嵌入断言的整个文档D，其生命周期可以过期。获得这样的消息后，攻击者通过注入恶意内容来修改它,例如一个恶意的断言EA。在我们的模型中，我们假设两种不同类型的对手，它们都比基于经典网络的攻击者弱：

1. Advacc。 为了获得断言，此攻击者注册为身份提供商IdP的用户。 Advacc然后通过与IdP的正常交互接收有效的签名SAML断言A（可能作为较大文档D的一部分）对Advacc进行声明。 攻击者现在还添加关于任何其他主题S的附加声明EA，并将修改后的文档D 0（A 0）提交给RP。
2. Advintc。 该对手从互联网检索SAML断言，但他无法读取加密的网络流量。 这可以通过直接从不受保护的网络（嗅探）访问传输的数据，或者通过分析代理或浏览器缓存以“脱机”的方式来完成。 SAML的断言生命周期一旦终止那便毫无价值的，甚至可能会被发布在技术讨论委员会（Advancecm）中。

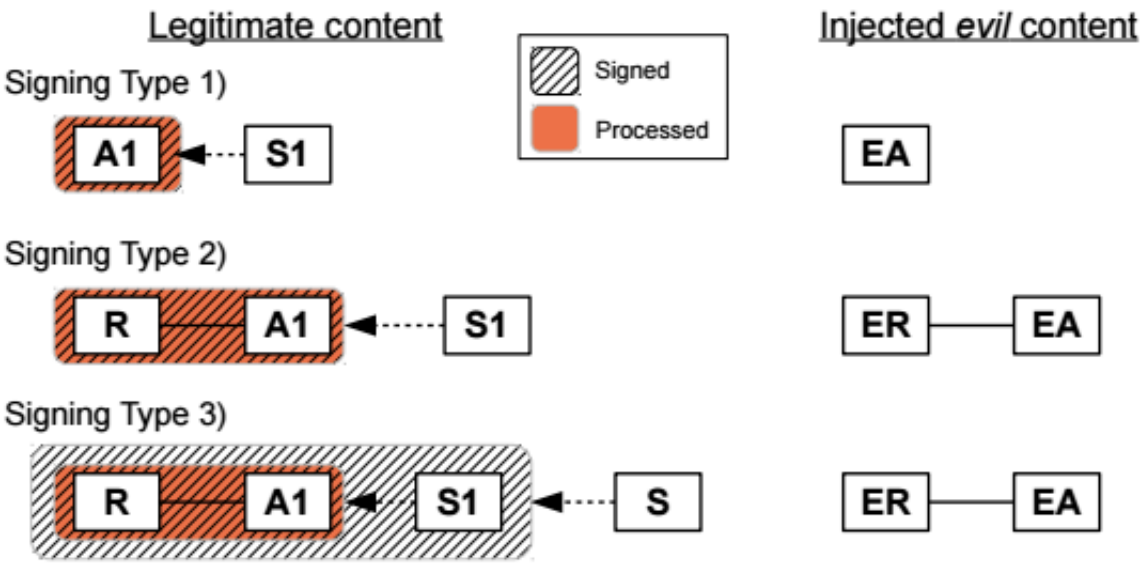


Figure 6: Types of signature applications on SAML assertions on the left. The new malicious content needed to execute the attacks depicted on the right, accordingly.

4.2 攻击原理

如上一节所述，XML签名可以以不同的方式应用于SAML断言，并放置在不同的元素中。 唯一的先决条件是使用基于Id的引用的包络签名对Assertion元素或协议绑定元素（Assertion的祖先）进行签名。 在本节中，我们分析了不同框架中SAML断言的用法以及插入恶意内容的可能性。 通常，SAML断言及其签名的实现如图6所示：

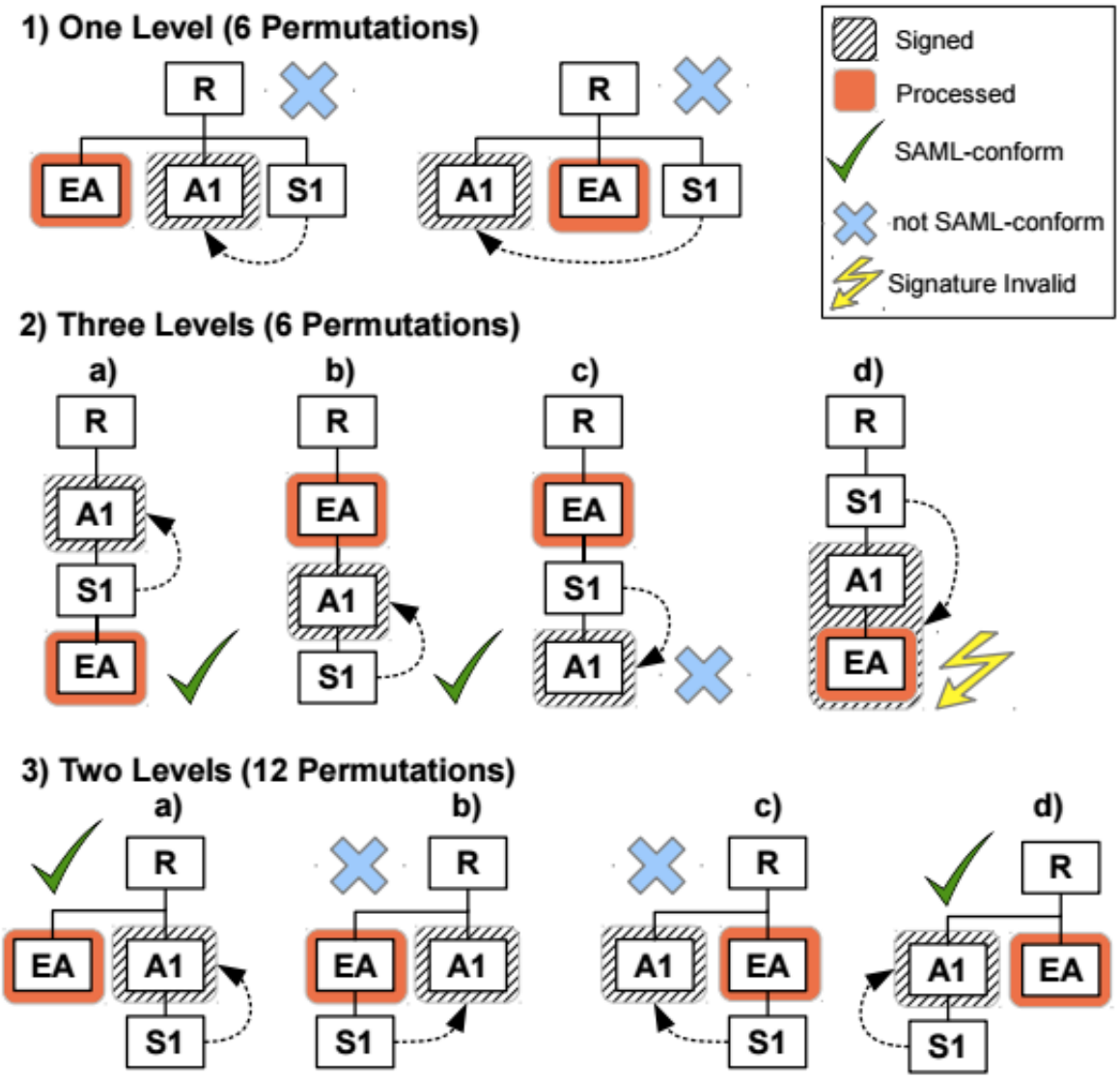
1. SAML签名的第一个可能用法是将XML签名S1作为SAML断言A1的子代插入，并且仅对Assert元素A1进行签名。这种类型可以独立于底层协议（SOAP或REST）使用。
2. SAML中的第二种签名应用程序签署了整个协议绑定元素R.可以将XML签名放入SAML断言A1或直接放入协议绑定根元素R.XML签名可以放入SAML断言A1或直接放入协议绑定根元素R.这种签名应用程序用于不同的SAML HTTP绑定，其中整个Response元素都被签名。
3. 也可以使用多个XML签名。第三个示例显示了这种签名应用：内部签名S1保护SAML断言，外部签名S另外保护整个协议消息。这种签名应用例如由SimpleSAMLphp框架使用。

为了将XSW攻击应用于SAML断言，基本的攻击理念保持不变：攻击者必须创建新的恶意元素，并强制使用断言逻辑进行处理，而签名验证逻辑可以验证原始内容的完整性和真实性。在第一个签名类型的应用程序中，攻击者只需创建一个新的恶意断言EA。在第二和第三个签名类型中，他还必须创建包括恶意断言在内的整个恶意根ER元素。

4.3 攻击排列

攻击者插入恶意和原始内容的的位置有很多可能性。为此，他必须处理以下问题： - XML消息树中的哪个级别应该包含恶意内容和原始签名数据？ - 哪个断言元素由断言逻辑处理？ - 哪个元素用于签名验证？

通过回答这些问题，我们可以定义不同的攻击模式，原始和恶意元素可以被重新排列（图7）。因此，我们获得了完整的攻击媒介清单，作为我们调查的指南。对于以下说明，我们仅考虑图6中定义的签名类型1）。在此签名类型中，仅引用了Assertion元素。



攻击排列如图7所示。此外，我们分析其SAML标准一致性和签名有效性：

1. 恶意断言，原始断言和签名留在相同的邮件级别上：这种XML邮件可以有六个排列。 它们都不符合SAML标准，因为XML签名没有签署其父元素。 所有消息中有符号元素的摘要值可以正确验证。 如果服务器不检查SAML一致性，我们可以使用这种类型的攻击消息。
2. 所有三个元素都以不同的消息级别插入，作为彼此的子元素，这再次导致六个排列：消息2-a和2-b示出了SAML标准符合和加密有效消息的示例。 在这两种情况下，签名元素引用其父代 - 原始断言A1。 消息2-c示出了不是SAML标准符合签名的子消息。 然而，消息在加密方面是有效的。最后，消息2-d示出了无效消息的示例，因为将在两个断言上验证签名。 通常，如果签名作为根元素的子代插入，则该消息也

将无效或不符合SAML标准。

3. 为了插入这三个元素，我们使用两个消息级别：消息3-a显示了一个有效和符合SAML的文档的示例。通过构建消息3-b，签名元素被移动到新的恶意断言。 由于它引用原始元素，它仍然有效，但不符合SAML标准。

上述分析可以类似地应用于具有不同签名类型的消息（参见图6）。

实际评估

我们评估了上述对第2节中介绍的常用ch系统和框架的攻击。在本节中，我们将介绍结果。

5.1 签名排除攻击

我们以最简单的攻击类型开始介绍我们的结果，称为签名排除攻击。这种攻击依赖于服务器的安全逻辑的执行不力，只有签名被包含在内才会检查签名有效性。如果安全逻辑没有找到Signature元素，它只是跳过验证步骤。

评估表明，三个基于SAML的框架容易受到这些攻击：Apache Axis2 Web服务框架，JOSSO以及在Eduserv中的SAML 2.0的基于Java的实现（SAML的其他版本和Eduserv中的C实现不受影响）。

通过对JOSSO和Eduserv应用此攻击，攻击者必须从消息中删除Signature元素，因为如果找到该元素，则该框架尝试验证它。另一方面，即使Apache Axis2框架包含在消息中，Apache Axis2框架也没有通过SAML断言来验证Signature元素。 Apache Axis2仅验证SOAP身体和Timestamp元素上的签名。保护在断言元素中单独包含的SAML断言的签名被完全忽略。

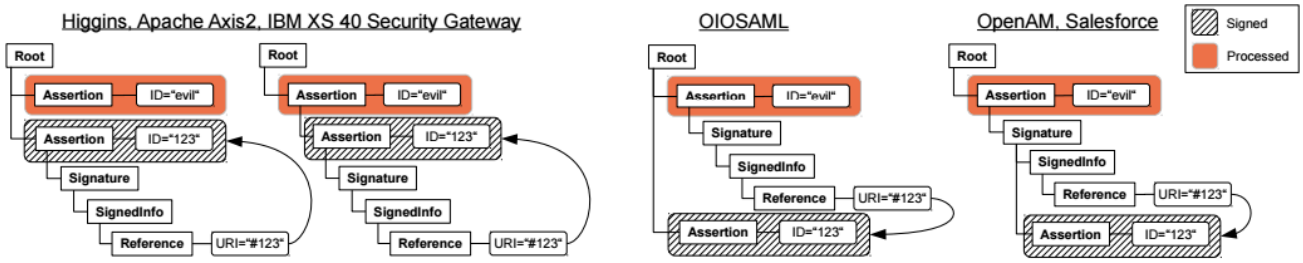


Figure 8: XML tree-based illustration of refined XSW attacks found in Type 1 signature applications.

5.2 精简签名包装

14个系统中有10个容易出现精细的XSW攻击。

根据图6给出的三种不同的签名应用程序类型，五个基于SAML的系统在验证类型1消息时失败，其中只有断言被XML签名保护。 图6中，五个基于SAML的系统在验证类型1时失败图8描述了找到的XSW变体的基于XML树的图示。从左到右，Higgins，Apache Axis2和IBM XS 40安全网关都被两个描述的排列所胜过在第一个变体中，在原始断言前面注入一个具有不同Id属性的邪恶断言就足够了。由于SAML标准允许在一个协议元素中具有多个断言，因此XML模式验证仍然成功。第二种攻击类型将原始断言作为子元素嵌入到邪恶的断言

EA中。在这两种情况下，XML签名仍然符合标准，因为包络签名被应用。在OIOSAML的情况下，使用分离的签名被打破了。在该变体中，原始的Signature元素被移动到EA中，该EA被插入到合法断言之前。最后显示的排列适用于Salesforce和OpenAM框架的云服务。在这一点上，真正的断言被放置在原始的Signature元素中。 由于两个实现都应用XMLSchema来验证SAML消息的模式一致性，所以通过将它们注入到允许任意内容的Object元素中来实现。再次，这不符合SAML标准，因为这个突变将被包围变换为包络签名。

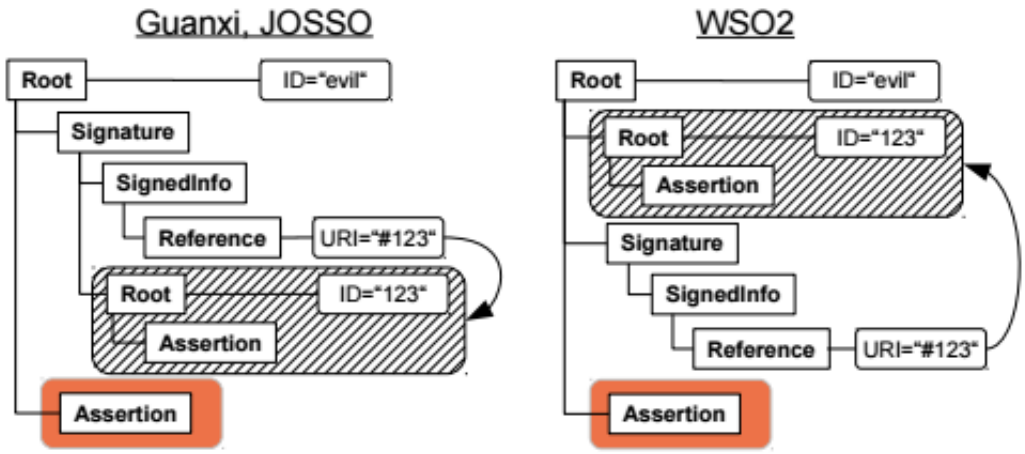


Figure 9: XML tree-based illustration of refined XSW attacks found in Type 2 signature applications.

我们发现三个敏感的实现，它们应用了类型2消息，其中整个消息被XML签名保护。 我们描述了图9中对这些实现的攻击。在Guanxi和JOSSO实现中，合法的根元素被插入到原始签名中的Object元素中。签名节点被移动到ER元素中，还包括新的恶意断言。在WSO2的情况下，将原始根元素放入ER对象就足够了。当然，有人会期望执行全面的文件签名将彻底消除XSW。 这两个例子表明，这在实践中并不适用。再次，这突出了实施复杂标准（如SAML）时所需的警惕。

最后，我们没有发现其根和标识都由不同的签名保护的应用类型3消息的弱点框架最后，我们没有发现应用类型3消息的弱点框架，其中根和标识都由不同的签名保护。实际上，一个合理的原因是，大多数SAML实现不使用Type 3消息。在我们的实际评估中，默认情况下只应用SimpleSAMLphp。 然而，这并不意味着XSW在实践中不适用于此消息类型。

5.3 OpenSAML 漏洞

上述攻击媒介对于普遍使用的OpenSAML库无效。原因是OpenSAML将签名验证使用的Id与被处理声明的Id进行了比较。如果这些标识符不同（基于字符串比较），签名验证失败。另外，包含多个具有相同ID的元素的XML消息也被拒绝。 这两种机制都通过使用Apache Xerces库及其XML Schema验证方法在OpenSAML中处理[34]。然而，可以用更复杂的XSW攻击来克服这些对策。

如前所述，在OpenSAML中，Apache Xerces库对每个传入的XML消息执行模式验证。因此，可以通过使用适当的XML Schema文件来定义每个元素的Id。这允许Xerces库识别所有包含的ID，并拒绝具有不唯一（例如重复）的Id值的消息。但是，此库中的一个错误导致使用xsd敌营XML元素：任何内容未正确处理。更具体

地说，使用定义的XML模式未检查定义为 `<xsd: any processContents =“lax”>` 的元素的内容。 因此，可以在XML消息中插入具有任意 - 也是重复的Id的元素。 这为我们的包装内容创造了一个很好的位置。

仍然是可扩展元素可用于执行我们的攻击的问题。这取决于两种加工性能：

1. 哪个元素用于断言处理?
2. 哪个元素由安全模块验证，如果有两个具有相同ID的元素？ 有趣的是，Apache Xerces（Java和C++）的两个现有实现以不同的方式处理元素解引用。

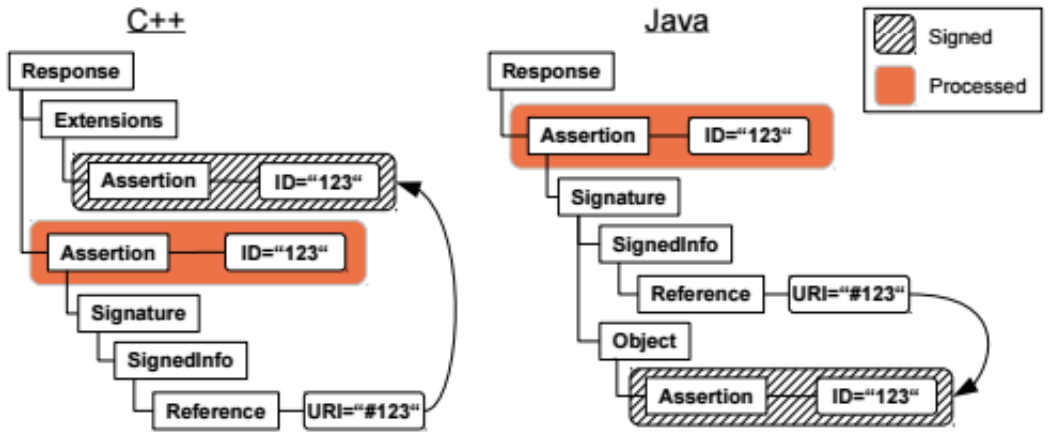


Figure 10: XSW attack on OpenSAML library.

```
<element name="Extensions" type="samlp:ExtensionsType"/>
<complexType name="ExtensionsType">
    <sequence>
        <any namespace="##other" processContents="lax" maxOccurs="unbounded"/>
    </sequence>
</complexType>
```

对于C ++，攻击者必须确保原始签名的断言在恶意断言之前被复制。 在Java的情况下，合法断言必须放在恶意断言之内或之后。总而言之，如果XML消息中发生了两个具有相同ID值的元素，则XML安全库仅检测到消息中的第一个（对于C++）或最后一个（对于Java）元素。这个属性使攻击者有机会使用例如 C++库的Extensions元素，其XML Schema在图11中定义。但是，Extensions元素不是我们的包装内容的唯一可能的位置。 SAML和XML签名的模式允许更多的位置（例如签名的Object元素，或断言的SubjectConfirmationData和Advice元素）。

以前描述的XML模式验证的行为强制OpenSAML使用包装的原始断言进行签名验证。 相比之下，应用程序逻辑处理了恶意断言的权利要求。 在图10中，我们介绍了这种新型XSW变体的具体攻击信息。

对OpenSAML的成功攻击表明，对抗XSW攻击可能会变得比预期更复杂。 即使应用了几项对策，开发人员仍应考虑底层库中的漏洞。 也就是说，XML Schema验证库中的一个漏洞可能导致执行成功的XSW攻击。

5.4 各种实现缺陷