Automatic Recognition, Processing and Attacking of Single Sign-On Protocols with Burp Suite

Christian Mainka[1] Vladislav Mladenov[2] Tim Guenther[3] Jörg Schwenk[4]

Abstract: SAML, Mozilla BrowserID, OpenID, OpenID Connect, Facebook Connect, Microsoft Account, OAuth — today's web applications are supporting a large set of Single Sign-On (SSO) solutions. Some of them have common properties and behavior, others are completely different. This paper will give an overview of modern SSO protocols. We classify them into two groups and show how to distinguish them from each other. We provide EsPReSSO, an open source Burpsuite plugin that identifies SSO protocols automatically in a browser's HTTP traffic and helps penetration testers and security auditors to manipulate SSO flows easily.

1 Introduction

Using username/password combinations to authenticate on websites is still dominating the Internet. From the security point of view the management of plethora login credentials is not a trivial task and carries many risks – users tend to use weak and easy to remember passwords or reuse passwords between different sites. Even if password managers are used, attacks are still applicable [Si14, Li14].

SSO systems simplify login procedures by using an Identity Provider (IdP) to issue authentication tokens which can be consumed by Service Providers (SPs). Thus, instead of managing multiple username/password combinations for each website, a user just needs an account at an IdP which can then be used to log in on an SP.

The importance of SSO has become more important in the recent years, since large companies like Facebook, Google, Microsoft and Salesforce offer different SSO services. For instance, Facebook's SSO service *Facebook Connect* allows its users to connect their Facebook account with various applications. More than 7 million applications use this protocol[We]. Additionally, a non-academic overview [Ja13] claims that 87% of U.S. customers are aware of SSO and more than half have tried it.

Today, there are several different SSO protocols. The most widespread are Kerberos, SAML, OAuth, OpenID, and OpenID Connect. Kerberos is provided in Microsoft's products like Active Directory Federation Service (ADFS) but rarely used in web applications.

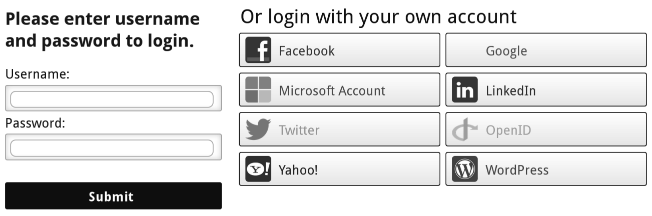
[1] Horst Görtz Institute, Ruhr-University Bochum, Germany, christian.mainka@rub.de
The research was supported by the *German Ministry of research and Education (BMBF)* as part of the `VERTRAG` research project.

[2] Horst Görtz Institute, Ruhr-University Bochum, Germany, vladislav.mladenov@rub.de
The author was supported by the SkIDentity project of the German Federal Ministry of Economics and Technology (BMWi,FKZ: 01MD11030).

[3] Horst Görtz Institute, Ruhr-University Bochum, Germany, tim.guenther@rub.de

[4] Horst Görtz Institute, Ruhr-University Bochum, Germany, joerg.schwenk@rub.de

SAML is a flexible and well standardized protocol offering extensive interoperability features commonly used in enterprise solutions, governmental services and large companies. OAuth, OpenID and OpenID Connect are less complex than SAML and easy to deploy. Thus, these protocols are mostly used for delegated authentication and authorization for websites and mobile devices. In recent years, companies have created and pushed their own SSO protocols: Facebook designed Facebook Connect on top of the OAuth specification. With Microsoft Account, Microsoft also offers an SSO protocol which is based on OAuth. Only Mozilla developed their SSO protocol Mozilla BrowserID from scratch.


Please enter username and password to login.

Or login with your own account

Fig. 1: Modern websites offer multiple login possibilities.

In summary, SSO is commonly used in all areas – desktop and web applications, mobile devices, government institutions and enterprise environments. In this focus we mainly concentrate on web applications. Figure 1 depicts a common example of what is called *social login* on a website. The user can either login using its username and password, or use one of his existing accounts (Microsoft, Facebook, Google, . . .). The hidden part of the *social login* is the underlying protocol: The user does not see (because it is not necessary) which exact SSO protocol is used. However, this information is important when it comes to security audits: a security auditor (pentester) needs to know which protocol is used so that he can evaluate its security.

A plethora design flaws and implementation errors in Kerberos [Sl01, Sh02], SAML [Ma14, So12], OpenID [WCW12, TT07, SHB12], OAuth [Eg13, YZ14], OpenID Connect [MM15c, MM15b, MM15a], Mozilla BrowserID [FKS14], and Facebook Connect [ZE14] led to critical vulnerabilities.

There exist different approaches to analyze SSO: (1) Via formal analysis the according protocol can be depicted, different threat scenarios can be automatically evaluated and protocol design flaws plus risks can be discovered. Unfortunately, implementation flaws cannot be detected via formal analysis. (2) Many researches concentrate on the analysis of existing implementations. The authors tend to introduce a novel tool, which provides an automated way to provide the security analysis. Unfortunately, such tools are limited to only *one* SSO protocol or *one* attacker model. An additional limitation is the extensibility in order to support more attack vectors and the false positive or false negative rates according the discovered flaws.

The limitations mentioned above are relevant for researchers elaborating novel attacks and security penetration testers, evaluating different services. Such analysis requires: (1) Recognition of protocols and relevant messages, (2) automated decoding of messages and security relevant parameters, (3) a flexible approach enabling the manual manipulation of different messages and parameters within the authentication protocol and (4) a set of existing attack vectors, which can be used for attacks.

To cover these limitations, we created a tool EsPReSSO[5] which is able to (1) detect and highlight SSO messages in the browser's traffic flow (i.e. the SSO token in the HTTP parameters) (2) determine the used SSO protocol. It currently supports all major SSO protocols that are used in modern web applications. (3) Additionally, EsPReSSO detects supported SSO protocols by just loading a website, e.g. a login page. (4) After the recognition of the SSO protocol EsPReSSO facilitates the manipulation of the related messages and automatically decodes and encodes them.

The main challenge tackled by EsPReSSO is the distinction between the different SSO protocols. This task requires an in-depth analysis of all protocols and detailed knowledge of the differences between them. For example, OpenID Connect and Facebook Connect are both based on OAuth and similar. Thus, a simple verification if an OAuth parameter is transmitted will not be able to distinguish between these protocols. Our paper will therefore give a detailed overview of recent SSO protocols and how they can be identified.

Contributions. The main contributions of this paper are the following:

- We provide an overview of seven modern SSO protocols. We classify them into *OAuth family* and *other protocols* and show, that the general protocol can be divided into a few generic steps among all those SSO protocols.
- We have created EsPReSSO, an easy to use open source Burpsuite plugin that automatically identifies SSO protocol messages and classifies them, so that security audits of modern web applications can benefit from it.

2 Foundations

2.1 Single Sign-On

SSO is a concept to login a user on an SP without storing any credentials on the SP. SSO therefore uses an IdP as a trusted third party. The IdP creates an SSO token, sends it back to the user, who passes it to the SP.

A generic description of SSO protocols is depicted in Figure 2. We will give more details on the concrete protocols in section 3. Figure 2 illustrates an abstract and generic protocol flow for modern SSO protocols like OpenID, OpenID Connect, SAML and Facebook Connect.

(1.) The user starts a login request using his user agent (UA) on the SP, for example by submitting his email address (Mozilla BrowserID) or his identifier URL (OpenID, OpenID Connect). (2.) Some SSO protocols then contact the IdP directly (server to server communication). This phase can be used to establish key material which is later used to sign and verify the messages or to determine the endpoint interfaces of the IdP, which will be used. Such an endpoint is for instance the login page at the IdP for the user. (3.) The SP responds to the first message with a token request. This message is then forwarded to the

[5] `https://github.com/RUB-NDS/BurpSSOExtension`

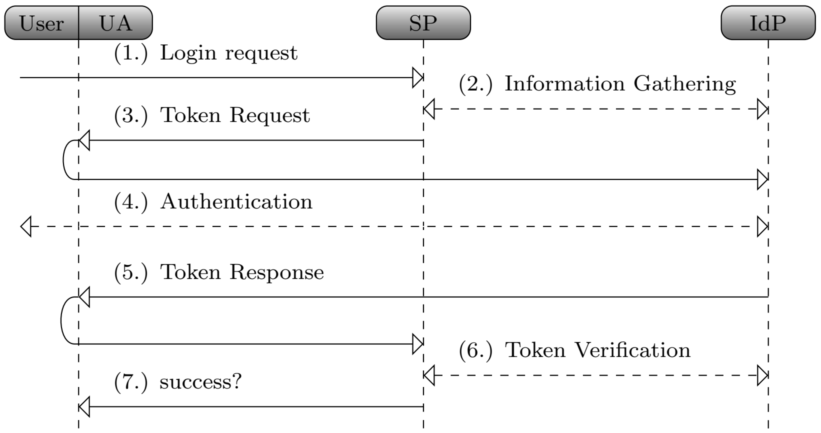


Fig. 2: Generic protocol flow for SSO protocols.

SP by the user (to be more precise, by his UA.). (4.) The user then authenticates to his IdP, typically by entering his username/password combination. Some protocols and IdPs require further user interaction in order to authorize the access to user's data like email address, nickname, birthday or gender. This step is often transparent for the User if he is already authenticated on the IdP. (5.) Next, the IdP sends the token response. This message contains all information that is necessary for the SP to identify the user. The message is forwarded to the IdP. (6.) The SP can then optionally contact the IdP again to verify the token response. Depending on the protocol, this is not necessarily (e.g. in SAML), because the token response contains a signature that can be verified.

2.2 Burpsuite

Burpsuite (Burp) is a penetration test tool by Portswigger[6]. It is available in a free and a commercial professional version. Burp acts as an intercepting proxy. This way, Burp can be configured on any UA as a proxy to log, intercept, display and modify HTTP traffic. The most commonly used UAs for Burp is a web browser, but it is also possible to configure it for any other application (e.g. Thunderbird, Skype, ...). In this paper, we use the free version of Burp. Features of the professional version are not necessary for our research.

Burp is often used by security auditors, researchers and penetration testers for the analysis of different systems. The core functionality of Burp is to intercept and display HTTP messages in a structured manner. Thus, a tester gets a quick overview of the target system, all transmitted messages and parameters. In addition, Burp provides a GUI allowing the full control over all messages - drop, forward, repeat, modify, send later, etc.. Thus, a tester can design different attack scenarios and execute them manually via Burp. The results of the attacks can be seen directly in the UA and analyzed by the tester.

[6] `http://portswigger.net/`

Simple parameter manipulations are supported by Burp and can be executed manually. However, more complex scenarios like decoding, manipulating and signing messages cannot be started in automated manner. In addition, manually analyzing each HTTP message can be time consuming and is often not necessary. In order to facilitate more complex scenarios Burp offers extension points, which allow writing custom features for it. Burp extensions can monitor and analyze any HTTP message that is passed through its proxy. Extensions can modify them and create new UI elements to display them.

3 SSO Protocols

This section will give a short overview of existing SSO protocols used in the web and introduces the necessary details used in EsPReSSO to identify them.

3.1 Protocol Classification

EsPReSSO is able to distinguish between seven different SSO protocols. We therefore classified them into two categories as shown in Table 1: (1.) SSO protocols belonging to the **OAuth-Family** and (2.) **other protocols**.

OAuth-Family		**Other**	
Decentralized	Monolithic	Decentralized	Monolithic
OAuth OpenID Connect	Facebook Connect Microsoft Account	OpenID SAML	Mozilla BrowserID

Tab. 1: Overview on existing SSO protocols used in the web and their classification.

The *OAuth-Family* consists of four different protocols. (1.) OAuth itself [RF] and (2.) OpenID Connect, which is an extension of the original OAuth protocol [Th14]. Both protocols can be used *decentralized*. By decentralized, we mean, that the protocol is independent of a specific provider. (3.) Facebook Connect [Mo08] and (4.) Microsoft Account [Mi08] in contrast are *monolithic*, because they relay on the Facebook resp. Microsoft servers. *Other protocols* are (1.) OpenID [sp07] and (2.) SAML [Or05], which are both decentralized, and Mozilla BrowserID, which is monolithic[7].

3.2 OAuth-Family Protocol Description

The following sections will give a quick overview of protocols of the OAuth family. We do not provide details on how the protocol works, but rather concentrate on the aspects that are necessary to distinguish them from each other. Our results are summarized in Table 2 on Page 124.

[7] Mozilla BrowserID allows one to setup one's IdP (*Primary IdP*-feature), but even in this use-case, the protocol contacts the Mozilla server at `login.persona.org` first.

3.2.1 OAuth

OAuth is an authorization framework that allows delegating access on specific resources to a third party. OAuth itself is not an SSO protocol [Sal14], but since previous research has shown, that developers tend to falsely use it for SSO [Ch14], we decided to add OAuth to the list of supported SSO protocols by EsPReSSO. Taking Figure 2, OAuth follows this protocol flow:

(1.) The user sends his login request to the SP.[8] (2.) The OAuth protocol does not use the *information gathering* phase, because all information on the IdP[9] is configured once beforehand. (3.) According to the specification [RF] within the token request the following parameters are required:`response_type` and `client_id`. The parameter `response_type` determines the *flow* that is going to be used. The most common flows are `code` and `token`. Other flows can be found in the specification [RF]. The parameter `client_id` is a unique string identifying the SP. Further optional parameters, which can be used to identify an OAuth token request are: `scope` for requesting permissions (e.g. the address-book or the calendar), `state` and `redirect_uri`. (4.) Then, the user has to authenticate to the IdP and authorize the requested permissions (`scope`) to the SP. (5.) The IdP generates the token response. If the *code* flow is used, the token response contains a `code` parameter. For the *token* flow, it contains a `access_token` parameter. (6.) The SP uses the received `code` or `access_token` to retrieve information about the user from the IdP and to authenticate him.

3.2.2 OpenID Connect

OpenID Connect is a decentralized SSO protocol by adding an authentication layer to OAuth [Th14]. The general flow is almost identical to OAuth as described in the previous section. Thus, the distinction between OpenID Connect and OAuth is not trivial and requires fine granular comparison.

According the specification a OpenID Connect token request must contain the following parameter: `scope, client_id, response_type, redirect_uri`. Unfortunately, the parameters are commonly used in OAuth too. Thus, the distinction on this level is not possible. However, in OpenID Connect the token request must contain the value *openid* in the `scope` parameter. Additionally, the token request can contain the parameter `nonce`, which is required within the *token* flow. Based on these characteristics the token request can be recognized.

The recognition of OpenID Connect token responses is more complicated and requires more detailed distinction. Within the *token* flow an additional parameter `id_token` will be sent by the IdP to the SP. The `id_token` is used only in OpenID Connect and provides information about the authenticated user. Thus, the identification of the token response is simple.

[8] In the context of OAuth, the *user* is commonly referred to as the *Resource Owner* and the SP as the *Client*. To simplify the description and to unify all SSO protocol, we strictly use user/SP naming.

[9] Again, we use the term *IdP* instead of the OAuth term *Authorization Server*. We also use the term IdP for the *Resource Server*.

The OpenID Connect token response within the *code* flow is identical to the OAuth flow. The only way to provide the distinction is to check the according token request sent before and bind both messages. This binding can be done by using parameters like `client_id`, `state` and `redirect_uri`, which are sent in the token request and token response.

3.2.3 Facebook Connect

Facebook Connect is a monolithic SSO protocol. It is based on OAuth and uses the same protocol flow as described in subsubsection 3.2.1.

The token request within the Facebook Connect protocol can be recognized by the following characteristics:

- The `scope` parameter can contain the value `signed_request`.
- In addition to the required OAuth parameters within a token request, the following parameters are sent: `domain, origin, sdk, app_id`.

Identical to OpenID Connect, the recognition of the token response is not trivial. Within the *token* flow, the parameter `signed_request` can be used. The value of this parameter is a JSON Web Token (JWT) containing information about the authenticated user. Similar to OpenID Connect the binding between the token request and token response via parameters like `client_id, state, redirect_uri` can be used.

Since Facebook Connect is monolithic, calling the public known SSO endpoints of Facebook's API can be used to identify the flow, for instance `https://graph.facebook.com`.

3.2.4 Microsoft Account

Microsoft Account is monolithic SSO protocol. It is based on OAuth and uses the same protocol flow as described in subsubsection 3.2.1. Microsoft Account token request can be easily detected by observing the `scope` parameter, which contains one of the following values: `wl.basic`, `wl.offline_access`, `wl.signin`.

Identical to OpenID Connect, the recognition of the token response is not trivial. Within the *token* flow, the parameter `authentication_token` can be used. The value of this parameter is a JWT containing information about the authenticated user. Similar to OpenID Connect the binding between the token request and token response via parameters like `client_id, state, redirect_uri` can be used.

Since Microsoft Account is monolithic, calling the public known SSO endpoints of Microsoft can be used to identify the flow, for instance `https://login.live.com/oauth20_authorize.srf`.

Protocol	**Message Type**	**Recognition**
OAuth	Token Request Token Response	Parameter: `response_type` Parameter: `code` OR `access_token`
OpenID Connect	Token Request Token Response	Parameter: `scope` contains *openid*, `nonce` Parameter: `id_token`

Facebook Connect	Token Request
	
	Token Response
	
	URL
		Parameter: `domain`, `origin`, `sdk`, `app_id`, `scope` contains *signed_request*
		Parameter: `signed_request`, `domain`, `origin`, `sdk`, `app_id`
		`http://static.ak.facebook.com/connect/xd_arbiter`
		`https://graph.facebook.com`

Microsoft Account	Token Request
	
	Token Response
	URL
		Parameter: `scope` contains *wl.basic, wl.offline_access* or *wl.signin*
		Parameter: `authentication_token`
		`https://login.live.com/oauth20_authorize.srf`
		`https://apis.live.net`
		`https://www.contoso.com/callback.htm`

Tab. 2: OAuth-Family message recognition and distinction

3.3 Other SSO Protocols

In the following sections, we describe SSO protocols that are not based on OAuth. We again focus on the properties which are important to identify the protocol rather than giving a complete protocol description.

3.3.1 SAML

SAML is a decentralized SSO protocol that uses XML to describe the security token. In the SAML protocol flow, there is commonly no interaction between the SP and the IdP [10], so Steps (2.) and (6.) in Figure 2 are skipped. The protocol flow is as follows: (1.) The user submits his login request to the SP. (3.) The SP generates the token request which contains a parameter `SAMLRequest`. The value of the parameter is basically XML and contains information on the to be used IdP (e.g. its URL). It is compressed using the deflate algorithm [De96] (optional), then encoded using Base64 [Jo06] followed by an URL-encoding [BLFM05]. (6.) The IdP generates the token response. This is again XML that is encoded using Base64 and optionally using URL-encoding. The result is stored in a parameter named `SAMLResponse`.

[10] An exception to this is the SAML Artifact Binding [Or05, Section 4.1.3]

3.3.2 OpenID

OpenID is a decentralized SSO protocol, but in contrast to, for example, SAML, it is *open* for dynamically using an IdP without any pre-configuration. By this means, anyone owning an OpenID can submit his identifier, which is an URL, to an SP in Step (1.) as shown in Figure 2. The SP will then discover the IdP in Step (2.) . He browses the URL and retrieves in this way the URL of the IdP. (3.) Next, the SP generates the token request and sends it back to the user. OpenID messages are easy to distinguish from other SSO protocols, since relevant all parameters start with `openid.*`. Message (3.) can be identified by the parameter `openid.mode=checkid_setup`. Authentication to the IdP is provided as usual in Step (4.) . The IdP then generates the token response in Step (5.) . This message can be identified due to the presence of a signature with parameter `openid.sig`. (6.) The SP can optionally send the token response to the IdP and sets `openid.mode=check_authentication` or he can choose to verify the signature on its own.

3.3.3 Mozilla BrowserID

Mozilla BrowserID is a monolithic SSO protocol developed by Mozilla and using Mozilla's server as an IdP during the authentication process. Interestingly, in Mozilla BrowserID using arbitrary IdPs is possible. However, Mozilla's SSO API is always called within the protocol flow.

The recognition of Mozilla BrowserID is possible by the detection of the HTTP parameter `assertion` containing information about the authenticated user within a JWT and a cookie named `browserid_state`. In addition, a JSON message containing key material can be used for the detection. The following parameters occurs within the message: `pubkey`, `p`, `q`, `g`, `algorithm`, `duration` and `email`.

4 EsPReSSO

This section provides a closer look on the design our Burp extension EsPReSSO.

4.1 Idea and Motivation

The Burp **E**xten**s**ion for **P**rocessing and **Re**cognition of **S**ingle **S**ign-**O**n Protocols (EsPReSSO), simplifies the analysis of SSO protocol flows. During our manual analysis of SSO, we often meet the problem to do the same repetitive work over and over again to determine the used protocol. To speed up the identification and to help inexperienced penetration testers, we decided to develop EsPReSSO.

Its simple idea is to have an automatic scanning utility that passively inspects a browser's traffic by scanning HTTP parameters and keywords. In the background the analyzing algorithm processes checks on the messages. If specific keywords and parameter-value pairs occur, the request/response is highlighted and marked as the recognized protocol. Additional we recognize SSO login possibilities by searching HTTP body responses, to track entry points for further research.

4.2 Design

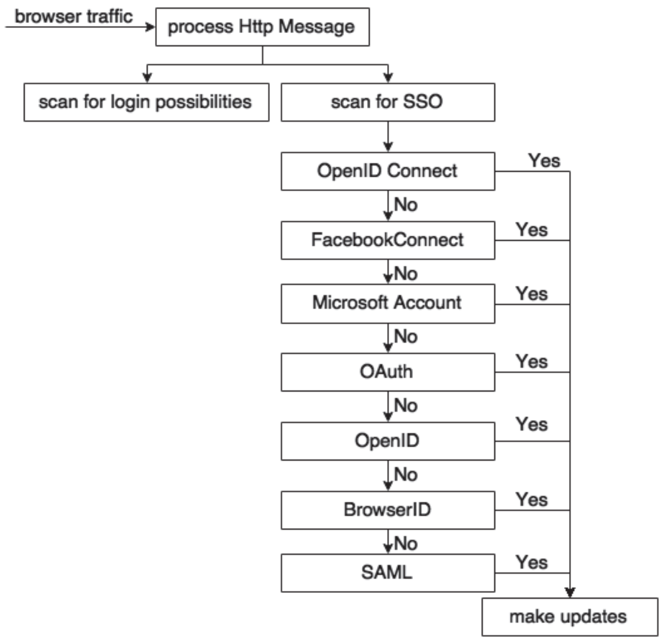


Fig. 3: Setup of the scanner.

EsPReSSO's core functionality is its scanning engine and the presentation of the results. One of our design goals is to stick as close as possible to Burp's user experience. By this means, we used existing structures like the logging mechanisms, the proxy history and its entries.

4.2.1 Scanner

The scanner carries out the detection of the SSO protocols according the described characteristics in section 3. Initially, the scanner uses Burp's interfaces and automatically receives all incoming traffic. Consequentially, it analyzes every loaded website for SSO login possibilities. Simultaneously, it scans the HTTP parameter and detects a SSO authentication process and the according SSO protocol.

The first submodule checks for the possibility to login with a specific SSO module, for example OpenID or Facebook Connect. This is implemented by searching the HTTP response messages through regular expressions for specific key words.

The second submodule inspects the HTTP traffic for specific properties that identify SSO protocols. It therefore searches successively for characteristics that are unique in each SSO protocol (cf. section 3). Please note the order of the given SSO modules, because distinguishing between protocols which partly base on the same protocol is difficult. *OAuth* is part of the protocols *OpenID Connect*, *Microsoft Account* and *Facebook Connect*, therefore we check these protocols first.

In addition, the scanner collects all collected information about the recognized SSO protocols, which allows the analysis afterwards.

4.2.2 Visualizer

Once SSO relevant parameters are detected, they have to be visualized. The Visualizer carries out this task by handling and filtering the collected data, converting the informations in human readable format (e.g. Base64-decoding or inflating) and calling different Burp APIs to display the results.

In detail, the Visualizer includes the following features:

Burp History Burp provides a history tab containing all intercepted messages. Thus, security auditors get an overview of the entire communication and can statically analyze the intercepted data. The Visualizer facilitates the evaluation process by highlighting the SSO relevant messages and by providing additional information about the recognized protocol.

SSO History A new Burp history window displays recognized protocols with additional data, for example, the used token and the protocol name. In comparison to the SSO History window, only SSO relevant messages will be displayed. The Visualizer provides more information about the messages, for example, the relation to other messages and the decoded content.

Follow SSO Flow By right clicking on a SSO History item a new tab is dynamically attached to the view with the complete protocol flow of the entry.[11] Token requests and responses will be assigned to each other, which facilitates the analysis of the entire protocol.

JSON Tab By analyzing the MIME-type of the HTTP messages, the Visualizer detects JSON messages and displays them. This feature is often used in OAuth to transmit data to the SP.

JWT Tab Protocols that are known to make use of JSON Web Tokens (JWT) get automatically a new tab to view the decoded JWT.

SAMLResponse/Request Tab Extra tab that displays the fully decoded and deflated SAML Request/Response messages.

[11] This feature is inspired by Wireshark's *follow TCP stream* feature

All new tabs come with syntax highlighting[12].

4.2.3 Manipulator

Security auditors often have to manipulate HTTP messages in order to simulate different attacks. Thus, in addition to the visualization of the protocols, EsPReSSO offers the possibility to modify the content of the messages.

In order to process the manipulations, the Manipulator offers the following features:

- Editable area containing all relevant parameters and enabling the modifications.
- Modifications will be detected and the old content will be replaced. The flexible architecture of EsPReSSO allows the manual or semi-automatic execution of modifications by choosing an attack vector from a predefined set of attacks.
- Data, which is transformed in a human readable format, will be transformed back to the original format. For instance, SAML tokens will be automatically decoded and — if necessary — deflated.

5 Related Work

SSO Security Tools. In 2013, Bai et al. [Ba13] have proposed AuthScan, a framework to extract the authentication protocol specifications automatically from implementations. The authors concentrated on Man-in-the-Middle (MitM) attacks, Replay attacks and Guessable tokens. More complex attacks like token manipulations were not considered. In the same year, Wang et al. [Xi13] developed a tool named *InteGuard* detecting the invariance in the communication between the client and SP to prevent logical flaws in the latter one. Another tool similar to *InteGuard* is *BLOCK* [LX11]. Both tools can detect and mitigate attacks, but cannot be used for penetration testing of existing implementations and manipulating the traffic. Zhou et al. [YZ14] published on USENIX'14 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. Further SSO protocols are not considered. In 2014, Mainka et al. [MM15d] published a fully automated tool acting as a malicious IdP for analyzing the security of OpenID implementations and described two novel attacks.

SSO extensions. In 2015 an extension called "SAMLyze" was published at Black Hat [Ba15]. Its goal is to pentest SAML SPs fast and easy against XXE, DTDs and to perform automatically a variety of SAML validations. In 2015 another extension analyzing SAML SPs was published [Bi15]. It contains two core functionalities: Manipulating SAML Messages and manage X.509 certificates.

However, both extensions concentrate on SAML but to not consider further SSO protocols.

[12] We use RSyntaxTextArea: `http://sourceforge.net/projects/rsyntaxtextarea/`

6 Conclusion and Future Work

EsPReSSO is the initial approach to create a tool capable to analyze different SSO protocols according their characteristics, to display all relevant parameters in a human readable format, and to manipulate the intercepted data in order to simulate different attacks. Thus, EsPReSSO facilitates the security analysis of SSO protocols.

EsPReSSO contains three different modules: Scanner, Visualizer and Manipulator. Each of these components can be easily extended. Thus, the detection of further protocols, further features regarding the depiction of the messages and manipulation possibilities can be added.

To the best of our knowledge, EsPReSSO is the first tool capable to detect, display and modify multiple different SSO protocols at the same time.

In future, EsPReSSO's functionality will be tested on a large set of websites and if needed modifications approving the detection will be implemented. Another issue is the enlargement of the available attacking set by considering attacks like XML Signature Wrapping (XSW) or attacks on JWTs.

References

[Ba13] Bai, Guangdong; Lei, Jike; Meng, Guozhu; Venkatraman, Sai Sathyanarayan; Saxena, Prateek; Sun, Jun; Liu, Yang; Dong, Jin Song: AUTHSCAN: Automatic extraction of web authentication protocols from implementations. NDSS, February, 2013.

[Ba15] Barber, Jon: , SAMLyze, August 2015.

[Bi15] Bischofberger, Roland: , SAMLRaider, Juli 2015.

[BLFM05] Berners-Lee, T.; Fielding, R.; Masinter, L.: , Uniform Resource Identifier (URI): Generic Syntax. RFC 3986 (INTERNET STANDARD), January 2005. Updated by RFCs 6874, 7320.

[Ch14] Chen, Eric; Pei, Yutong; Chen, Shuo; Tian, Yuan; Kotcher, Robert; Tague, Patrick: OAuth Demystied for Mobile Application Developers. In: Proceedings of the ACM Conference on Computer and Communications Security (CCS). ACM – Association for Computing Machinery, November 2014.

[De96] Deutsch, P.: , DEFLATE Compressed Data Format Specification version 1.3. RFC 1951 (Informational), May 1996.

[Eg13] Egor Homakov: , How we hacked Facebook with OAuth2 and Chrome bugs, Februrary 2013.

[FKS14] Fett, Daniel; Kusters, Ralf; Schmitz, Guido: An expressive model for the Web infrastructure: Definition and application to the Browser ID SSO system. In: Security and Privacy (SP), 2014 IEEE Symposium on. IEEE, pp. 673–688, 2014.

[Ja13] Janrain: , 2013 Consumer Research: The Value of Social Login, 2013.

[Jo06] Josefsson, S.: , The Base16, Base32, and Base64 Data Encodings. RFC 4648 (Proposed Standard), October 2006.

[Li14] Li, Zhiwei; He, Warren; Akhawe, Devdatta; Song, Dawn: The emperor's new password manager: Security analysis of web-based password managers. In: 23rd USENIX Security Symposium (USENIX Security 14). 2014.

[LX11] Li, Xiaowei; Xue, Yuan: BLOCK: A Black-box Approach for Detection of State Violation Attacks Towards Web Applications. In: Proceedings of the 27th Annual Computer Security Applications Conference. ACSAC '11, ACM, New York, NY, USA, 2011.

[Ma14] Mainka, Christian; Mladenov, Vladislav; Feldmann, Florian; Krautwald, Julian; Schwenk, Jörg: Your Software at my Service: Security Analysis of SaaS Single Sign-On Solutions in the Cloud. In: Proceedings of the 6th edition of the ACM Workshop on Cloud Computing Security, CCSW '14, Scottsdale, Arizona, USA, November 7, 2014. pp. 93–104, 2014.

[Mi08] Microsoft: , One account for all things Microsoft, May 2008.

[MM15a] Mainka, Christian; Mladenov, Vladislav: , Connect2id Acknowledgement, 2015.

[MM15b] Mainka, Christian; Mladenov, Vladislav: , CVE-2015-0959, 2015.

[MM15c] Mainka, Christian; Mladenov, Vladislav: , CVE-2015-0960, 2015.

[MM15d] Mainka, Christian; Mladenov, Vladislav: , Do not trust me: Using malicious IdPs for analyzing and attacking Single Sign-On (Full Version with Attachments), 2015. [online] `http://bit.ly/maliciousIdPs_fullversion`.

[Mo08] Morin, Dave: , Announcing Facebook Connect, May 2008.

[Or05] Organization for the Advancement of Structured Information Standards: . Security Assertion Markup Language (SAML) v2.0, 2005.

[RF] RFC6749, IETF: , The OAuth 2.0 Authorization Framework.

[Sal14] Salesforce.com, inc. Inside OpenID Connect on Force.com, 2014.

[Sh02] Shiflett, Chris: , Passport Hacking Revisited, 2002.

[SHB12] Sun, San-Tsai; Hawkey, Kirstie; Beznosov, Konstantin: Systematically breaking and fixing OpenID security: Formal analysis, semi-automated empirical evaluation, and practical countermeasures. Computers & Security, 31(4), 2012.

[Si14] Silver, David; Jana, Suman; Chen, Eric; Jackson, Collin; Boneh, Dan: Password managers: Attacks and defenses. In: Proceedings of the 23rd Usenix Security Symposium. 2014.

[Sl01] Slemko, Marc: , Microsoft Passport to Trouble, 2001.

[So12] Somorovsky, Juraj; Mayer, Andreas; Schwenk, Jörg; Kampmann, Marco; Jensen, Meiko: On Breaking SAML: Be Whoever You Want to Be. In: Presented as part of the 21st USENIX Security Symposium (USENIX Security 12). USENIX, Bellevue, WA, pp. 397–412, 2012.

[sp07] specs@openid.net: , OpenID Authentication 2.0 – Final, December 2007.

[Th14] The OpenID Foundation (OIDF): , OpenID Connect Core 1.0, February 2014.

[TT07] Tsyrklevich, Eugene; Tsyrklevich, Vlad: , Single Sign-On for the Internet: A Security Story, July and August 2007.

[WCW12] Wang, Rui; Chen, Shuo; Wang, XiaoFeng: Signing Me onto Your Accounts through Facebook and Google: A Traffic-Guided Security Study of Commercially Deployed Single-Sign-On Web Services. In: Proceedings of the 2012 IEEE Symposium on Security and Privacy. SP '12, IEEE Computer Society, Washington, DC, USA, 2012.

[We] Websites using Facebook Connect. visited on 2015-05-25.

[Xi13] Xing, Luyi; Chen, Yangyi; Wang, X; Chen, Shuo: InteGuard: Toward Automatic Protection of Third-Party Web Service Integrations. In: Proceedings of 20th Annual Network & Distributed System Security Symposium. 2013.

[YZ14] Yuchen Zhou, David Evans: Automated Testing of Web Applications for Single Sign-On Vulnerabilities. In: 23rd USENIX Security Symposium (USENIX Security 14). USENIX Association, San Diego, CA, August 2014.

[ZE14] Zhou, Yuchen; Evans, David: SSOScan: Automated Testing of Web Applications for Single Sign-On Vulnerabilities. 23rd USENIX Security Symposium, 2014.