Logout in single sign-on systems: Problems and solutions

Sanna Suoranta a,*, Kamran Manzoor a, Asko Tontti b, Joonas Ruuskanen a, Tuomas Aura a

a Aalto University, Espoo, Finland
b CSC – IT Center for Science, Espoo, Finland

Keywords:
Authentication
Single sign-on systems
Federated identity management

A B S T R A C T

Web single sign-on (SSO) systems enable users to authenticate themselves to multiple online services with one authentication credential and mechanism offered by an identity provider. The topic is widely studied and many solutions exist. However, logging out of a service using SSO has received less attention. While previous studies note that users want single logout when using SSO, most of the existing services do not offer it, and the identity providers do not even keep track of the open sessions. This article describes challenges related to logout in federated identity management and analyzes unexpected behavior in logout situations. The examples are from the Shibboleth SSO system. Based on the analysis, we give guidelines for implementing reliable logout and describe a polling-based solution for creating a system-wide logout mechanisms that only requires minor changes to the existing code and does not burden the identity provider excessively. In addition to the system-wide logout, our solution gives users the option to log out of only one service. A usability test was conducted to evaluate the solution. The results show that the users liked the ability to choose between the two logout options, but they did not understand the words used to describe them. Another observation was that a majority of the users do not log out of the services at all; they just close the browser window, which should be taken into account in the design of web SSO systems.

© 2014 Elsevier Ltd. All rights reserved.

1. Introduction

Service providers on the Internet want to authenticate their users in order to offer them better service. This authentication may be needed, for example, for profiling users, for targeted advertising, for charging a fee for the service use, or for protecting the users’ personal information. On average, users log in to eight services daily using different passwords (Florêncio and Herley, 2007). They have even more passwords for services they do not use daily, and this sometimes leads to a situation where users either choose poor passwords or recycle them for use with many services (Gaw and Felten, 2006). To help users cope with the authentication, many services employ single sign-on (SSO) technology, with enables users to use a single password for many services. There exists several solutions where, basically, the service itself is separated from the user authentication and authorization, which are then
offered by a separate authentication service. Even social media services such as Facebook and Google+ can be used to authenticate users for third party services. Open-source implementations of SSO systems are mostly based on either the Security Assertion Markup Language (SAML) (Cantor et al., 2005; Hirsch et al., 2005) or OAuth (Hardt, 2012) standard, which are also commonly used in commercial products and services. Various proposals have been made to enhance SSO systems, for example, with respect to their architecture (Cahill et al., 2011; Chadwick et al., 2011; Sun et al., 2010a; Takeda et al., 2006), strength of the authentication (Mustafic et al., 2011; Suoranta et al., 2012), usability (Linden and Vilpola, 2005; Sun et al., 2011a, 2011b), and privacy (Alsaleh and Adams, 2006).

Many of the SSO services consider their task complete after the user is authenticated at the beginning of a service session. However, the end of the session is also important. For example, if a user connects to the service using a public computer in a library or cafe and does not log out, the next user of the computer may gain access to the service with the previous user’s privileges. Even today when online services are increasingly accessed through personal laptops and smart phones, shared devices are used within families, between friends, as well as widely in developing countries. Moreover, the timeout periods are often quite long because users do not like to have to sign in repeatedly. Thus, the user may forget which devices and web browsers have ongoing open sessions. In any event, if the server end of a connection remains open when the client end closes, this may leave a door open for an attacker to gain access to the service using the half-open session in combination with some other information. Thus, it would be safer to terminate the session also on the server side.

In this article, we have investigated the logout and its implementation in services that use Shibboleth SSO at Aalto University. We found many problems in the logout procedures: none of the services implement single logout, and at least some server-side sessions are usually still active after logout. Since Shibboleth allows services either to handle sessions on their own or to outsource the session management to a local software component called the service provider, sessions managed by both should be completely deleted on logout. However, some of the services we examined leave either the service or service-provider cookies intact upon logout, which in turn allows users to get back in without authentication. Another significant problem was that the user could not, on clicking a logout button, know from where she would be logged out. Some services used local logout to end their local sessions only, while other services also ended the identity-provider sessions. In the latter case, accessing new services requires re-authentication, but other existing service sessions remain active. This is because Shibboleth does not support real single logout.

Based on our findings, we suggest a list of actions that help in implementing a more secure and usable logout for Shibboleth. Some of the solutions are general and could also be used to improve other SSO systems. Moreover, we implement a real single logout solution based on polling. By polling the identity provider, services can check if the authentication session is still active in the identity provider. The identity provider does not need to be changed, and nor does it need to keep track of the services the user has logged in to. In addition, we decided that the user should be able to choose between logging out of one service (local logout) and logging out of all the services at the same time (single logout). We conducted a usability test with 18 users to check the understanding of both logout options from the users’ perspective.

The rest of this article is organized as follows. In Section 2, we describe single sign-on systems in general. In Section 3, we present how logout works in different SSO systems, and in Section 4, we overview related work on logout. In Section 5, we discuss the problems of logout in practice. Then we describe our solution for logging out of the services in Section 6. The proposed solution is validated with a usability test in Section 7. Finally, Section 8 ends the article with our conclusions.

This article is an extended version of a previous conference paper (Suoranta et al., 2013).

### 2. Single sign-on systems

Single sign-on systems are such that the users can access several services by authenticating themselves only once. The SSO systems can be divided into four categories: local and proxy-based, pseudo and true SSO systems (Pashalidis and Mitchell, 2003). In the local pseudo-SSO system, a user’s device has a password manager that automatically works on behalf of the user when logging in to a service. The proxy-based pseudo-SSO system is similar, but the password manager is located on a network proxy. The local true SSO system relies on a trusted component in the client device, but otherwise is quite similar to the local pseudo-SSO. The proxy-based true SSO system has an external server for authenticating the user for services. In this article, we concentrate on proxy-based true SSO systems, often called federated SSO. For example, OpenID (OpenCommunity, 2007) and Shibboleth (Shibboleth Consortium, 2012) are proxy-based true SSO systems. In addition, many service providers allow other web service developers to use their user authentication with their proprietary protocols. For example, Facebook Login (Facebook, 2013) and Microsoft Live ID (Microsoft, 2012) provide centralized authentication for third-party web applications where users can log in using their existing identities at Facebook and Microsoft services.

In this section, we describe some popular SSO systems. First, we describe federated identity management systems such as SAML 2.0 and Shibboleth and the login process in them, then capability based authorization systems such as OpenID and OAuth, and finally briefly comment on social media services that are also used as SSO systems.

#### 2.1. Federated Identity Management and Shibboleth

Liberty Alliance Project (Liberty Alliance Project, 2003; Cantor and Kemp, 2004—2005) introduced the concept of Federated Identity Management (FIM). In FIM, service providers (SPs) delegate the user authentication to an identity provider (IdP). Nowadays, IdPs are often called identity service providers since they do not provide actual identities but services on top of identities. The services and the identity provider form a
federation. For example, a customer of an airline would be able to use her credentials to log in to a car-rental service without re-authentication. The federation can have several IdPs that the services choose to accept to identify their users, and the user can then choose with which to authenticate. Liberty itself did not succeed as a product but this fundamental idea is widely used nowadays. In this paper, we use the term identity provider for the entity handling the authentication or authorization of the users, and service provider or service for the actual service. Other technologies can have their own terminology for the participants.

The Security Assertion Markup Language (SAML) V2.0 (Cantor et al., 2005; Hirsch et al., 2005) specifies assertions that are used to communicate user authentication and the user’s attributes between the IdPs and services, which are actually called relying parties in SAML. Assertions are defined using eXtensible Markup Language (XML), and SAML defines the messages that are exchanged between the IdP and services.

Shibboleth, developed by the Internet2 project, uses the SAML messages (Shibboleth Consortium, 2012). It offers an open-source software implementation of the identity provider and service provider. Shibboleth requires the identity providers and service providers to form a federation before the IdP can authenticate users for services. They need to share metadata information that defines the addresses and public keys of the identity providers and the SAML attributes required by each service. If there are several IdPs in the federation, a discovery service is needed to allow users to choose between them. Shibboleth is used globally in many universities. For example in Finland, the HAKA federation operated by CSC uses Shibboleth to authenticate both student and staff to services of higher-education institutions (CSC, 2006) and to some external partners. Each university has its own IdP. Some of the services are shared while others are university-specific. Shibboleth is also used in the universities of other Nordic countries (Kalmar2, 2013), which form a wider federation.

Fig. 1 depicts the overall Shibboleth architecture. The user accesses a web service with a web browser. The identity provider (IdP) performs the authentication. Slightly confusingly, there is also a software component in Shibboleth called service provider (SP) running on the same server as the service itself. It handles the communication with the IdP as well as some local session information. The figure also shows the different types of sessions that may exist in the system: (1) IdP session between the browser and the IdP, (2) SP session between the browser and the SP component, and (3) service session between the browser and the service itself. There can be several services on the same server sharing one SP. Some of them can take care of their own session management and use the SP only for the login phase. Others let the SP save the session information on their behalf. The integration of the SP to all the organization’s services is the most demanding part of Shibboleth deployment. This paper was motivated by the observation that there is confusion and outright misunderstandings among the integrators about the way the sessions are created and deleted and how they relate to each other.

Fig. 2 shows how the initial login process works in the Shibboleth SSO system. When users want to access a service, they first start their web browsers and connect to the service web address (1). The browser window is redirected to the Shibboleth service provider component (2). The SP responds to the unauthenticated users by redirecting the browser to the identity provider (IdP) (3). If there are several IdPs, the users are first asked to choose one. The IdP authenticates the users by requesting credentials such as username and password and then creates an IdP session for them (4). It redirects the browser back to the SP with authentication assertions (5). The SP creates the SP session and service session to give the requested access to the users. Another step of redirection takes the browser back to the service page on the same server. If the users want to use another service with the same authenticated identity, the IdP does not need to ask for their credentials again. It only checks that the valid IdP session exists, and then it gives the authentication assertions to the second SP. The IdP and SP sessions are implemented with a combination of cookies and on-server state. The service session implementation depends on the service software. It is also worth noting that, in the current versions of Shibboleth, the IdP does not keep track of which SPs the users have been authenticated to.

2.2. Other SSO systems

OpenID is a widely used SSO technology (OpenID Community, 2012). Services either allow any identity provider or only
selected ones. Users choose a specific identity provider from that list to authenticate themselves. OpenID does not require a pre-established trust federation between services and identity providers. The service provider just links to the identity provider’s web site. Moreover, OpenID does not form a session where the identity provider would keep track of service providers to which the user has logged in. These can be the reasons why service providers are reluctant to take OpenID into use. Sun et al. (Sun et al., 2010b) list further problems of OpenID-based SSO and why services lack incentives to outsource user authentication. However, many service providers use the OpenID technology to implement closed authentication systems and accept only their own identity provider. OpenID has even been considered for use in governmental services in the USA (Thibeau and Reed, 2013).

Capability-based authorization systems are used for both user authentication and for managing access rights. For example, by using a social-network account for login on a third-party service, a user might also grant it controlled access to her social-network profile. The authentication service might not have a session for the user during the time the user is active; it may only provide the information for the service in the login process. Some of these systems are not single sign-on systems in the sense that they would allow getting into multiple services with one authentication. However, they reduce the number of passwords needed because the authorizing entity is the same for several services, and the same password thus allows accessing many services.

The OAuth 2.0 standard is a capability-based authorization framework developed by the Internet Engineering Task Force (IETF) (Hardt, 2012). In addition to web applications and web-browser-based services, OAuth works also for native applications. It allows, for example, a service to be constructed from several sub-services in such a way that the users need to authenticate themselves only once. A resource owner — or actually an authorization server on its behalf — grants an access token that the client presents (on the user’s behalf) to the service. If the token is valid, the service grants access to the client. The access token is a string usually containing the scope and duration of access, and it may have other information that is needed for granting the authorization. Its format and structure depend on the security requirements of the service. In addition to access tokens, OAuth uses refresh tokens. They are used to obtain a new access token from the authorization server when the old access token expires or the same authorization is needed again. The authorization server stores the tokens until they become invalid.

2.3 Social media services as identity providers

Several commercial organizations offer their authentication services for third party services. For example, service developers can use the social media service Facebook to authenticate users with Facebook Connect (Facebook, 2013). The Facebook Login offers application programming interfaces (APIs) for both native and web-based applications (Facebook, 2013) based on an OpenId-like proprietary protocol. Facebook Login means that the user grants the application access to her public profile and friends list. In addition, the application can ask the user to give permissions, for example, to publish content on their Facebook page or to extend access to further profile properties.

Google is another popular source of authentication. It provides an authentication and authorization API for third party service developers (Google, 2013). Google supports both OpenID Connect and OAuth 2.0; the latter is called Google+ Sign-In.

3. Logout in the SSO systems

Logging out means clearing the sessions that are created when the user is signed in. In this section, we examine the logout behavior of different SSO technologies. Service developers can either implement online services as native applications or they can be used on a web browser. Single sign-on systems can offer authentication in both cases, but we concentrate on the web case. The web-browser-based versions of the services usually rely on cookies to manage the sessions. There are two kinds of cookies: persistent and temporary. Session cookies are usually of the temporary type. Closing the web browser destroys the temporary cookies but persistent cookies are stored on the hard disk. If the users do not close the browser, the cookies sometimes allow users to get back into the services. For example, if the users have not logged out of Gmail, they are still treated as authenticated users during their Google searches even if they have closed the browser tab of the Gmail, which may be confusing for users.

In a traditional web services, the user session is usually terminated either when the user explicitly clicks on a logout link or when the session timeout is reached. In the SSO systems, logging out is not so straightforward. The users might want to log out either of a single service or of the whole SSO session. However, the result might be something quite different. We have observed the following cases:

- **Logout in the service application.** If the application takes care of its own user session management, it terminates the user sessions but the SP session still needs to be terminated separately. If the SP takes care of the user session management on behalf of the service, the service needs to register a handler in SP for logout.
- **Logout in SP.** If SP takes care of all the user session management, it offers an API to the services to start the logout in SP. In logout, the SP removes the sessions.
- **Logout in IdP.** The IdP takes care of the IdP session during which the user can log in to services without re-authentication. If logging in to other services is not wanted, the IdP session can be removed. Removing the IdP session does not affect the session in any way on the SP and the services.
- **Local logout.** If the users want to log out of a single service, they use the local logout that closes both the service and SP sessions related to that service but not the IdP session, nor the sessions of other services and SPs.
- **Single logout (SLO) i.e. global Logout.** If the user want to log out of all the services, they choose the single logout. In addition to the currently used service and SP, the IdP goes through all SPs and asks them to log out the user, and ends...
as to close the web browser.

In the SSO systems, the session should end either in a local logout or single logout. In practice, the latter is often a local logout combined with the IdP logout, not a real SLO. Correctly implementing the SLO is particularly important when shared computers are used e.g. by students in a university library. However, many users have personal computers, and thus a local logout of a single service might often be more practical, e.g. when the user will log in to another service with the same IdP after logging out of the first service. If also the IdP session has been terminated, the benefits of using the SSO system are lost since the user has to sign in again every time.

Next we go through how logout works in different SSO systems and federations.

3.1. SAML and Shibboleth logout

In SAML 2, services are usually applications running on a web server, which also runs the service provider software. The service provider software takes care of communication with identity providers. It can manage user sessions on behalf of services, but may services prefer to manage their own sessions. SAML 2 only describes the logout in general and different profiles complete it with conflicting suggestions. For example, the order of removing the sessions on the SP and IdP is not clear. The following describes the SAML2 logout with front-channel binding, which means that it uses the web browser to communicate between the different actors. Note that this process is not fully implemented in Shibboleth, as will be explained later.

1. The user chooses logout in a web service.
2. The service removes the user’s session.
3. The service redirects the web browser to the SP’s logout handler. It can set a return parameter if the browser should be redirected to a given URL after a successful logout.
4. The SP removes the user’s SP session.
5. The SP searches for the IdP’s logout service address from the metadata and redirects the web browser to it with the SAML2 LogoutRequest message.
6. The IdP removes the user’s IdP session (PreviousSession).
7. If necessary, the IdP asks an external authentication service to remove its session.
8. The IdP sends the SAML 2 LogoutRequest messages through the web browser to all the SPs except the one which started the logout. Each SP removes its sessions and asks the services to remove the sessions, too. All the SPs send messages about the success of the logout to the IdP through the web browser.
9. The IdP sends information about the success of the logout through the web browser, using the SAML 2 LogoutResponse message to the SP that started the process.
10. Finally, the SP can either tell the user that logout has been completed successfully or redirect the user to the URL given in step 3.

The SAML 2 logout provides SLO but requires the IdP to keep track of all the SPs on which the users are logged in, which requires more resources from the IdP. Moreover, if the users have logged in using different IdPs, only the services for which they have used the same IdP will be logged out.

While SAML thus specifies single logout and Shibboleth is based on SAML, Shibboleth 2 does not provide single logout; it typically just logs out the service provider session. That is, only steps 1–4 of the above process are implemented.

In the Haka federation of the Finnish higher-education institutions, a separate procedure for logging out of both the SP and IdP is offered (CSC, 2012). The beginning of the HAKA logout works in the same way as the SAML2 logout in steps 1–7, and the rest as follows:

8. The web browser returns to the URL if the SP provided one when it called the IdP.
9. The browser tells the user that she is logged out of the service and the IdP.
10. The user is encouraged to close the web browser to be sure that logout has been completed. This ensures that the user will not remain logged in on other IdPs.

The HAKA logout is similar to the SAML 2 logout, but the IdP does not communicate with other SPs, and thus the other service sessions remain open. This is because the IdP does not keep track of the SPs to which the users have authenticated themselves and has no mechanisms for communicating with them. Currently used Shibboleth SP 2 supports full SAML2 logout but the IdP 2 does not. A future IdP 3 will probably have a back-channel-based SLO where the messages are passed using SOAP.

3.2. OpenID and OAuth logout

Since OpenID services and identity providers do not form a relationship, single logout is not possible in OpenID. An OpenID service can only end its own session independently and locally because the identity provider may not even have an authentication session for the user, nor does it keep track of the services to which it has authenticated the user. In a way, the identity provider just informs the service about the user’s identity at the time of the login. However, the identity provider can keep a session for the user, and this session can be terminated in the logout. For example, Google staff in the developer discussion forum suggested that the service developer should either provide logout only from the service or redirect the browser to Google’s logout URL (Balfanz, 2009). However, the service should know which IdP the user has used to log in to the service in order to provide the correct logout URL. As the identity provider and service provider have no relationship beyond the authentication event, the identity provider does not need to know, and nor does it care, if the user wants to log out of a service. The OpenId single login
could, of course, be extended with a system-specific logout mechanism. In that case, the identity provider would keep track of the user sessions at services within the organization, and the services would be individually integrated into the system-wide logout mechanism. This naturally does not scale beyond the single organization.

OAuth 2.0 is based on access tokens which the client presents to the service. There are no negative tokens as the client could not be forced to present them to the service, but the OAuth Working Group of IETF is developing a method that allows the client to inform the authorization server that the token is not needed anymore (Lodderstedt et al., 2013). When a user logs out of a service, it can tell the authorization service that the access and refresh tokens can be removed from the database. If the revoked token is a refresh token, the authorization server may revoke the original access token and other refresh tokens based on that access token. This depends on the authorization server policy. If the revoked token is an access token, the refresh tokens may be revoked, too. This way, the revocation of tokens allows users to log out of a collection of dependent services. However, the draft standard does not define how the authorization server would inform the other services about the revocation. This means that the revocation information will be propagated only when the service access tokens expire and an attempt is made to refresh them. Immediate single logout is currently not possible.

3.3. Logout in Facebook

Facebook Login provides a logout function for service developers. It logs the user out of both the service and of the Facebook because otherwise users might get confused in a situation where they are logged in on the service only (Facebook, 2013). Logging out can also be implemented from the service only by revoking the login authorization token (Facebook, 2013). Then, if the user has logged in to Facebook, she will not be logged out and the user can continue browsing Facebook. However, if the user wants to get back into the service, she has to get permissions again for the service by going via the Login dialog. Like many other web based services, the Facebook Login relies on cookies. Cubrilovic (2011) describes how Facebook uses permanent cookies for the sessions so that, even after the logout, it can track users’ behavior.

4. Related work on logout in SSO systems

Single logout has not been studied widely although single sign-on systems have been developed and researched extensively. Many studies just mention single logout but do not elaborate its actions or significance in federated services.

Sun et al., (2011b) have tested the usability of OpenID as such and with their own browser that integrates support for OpenID. They concentrated on logging in to services but their own system offered the users also two logout options: only the service at hand, identified by its URL, or single logout as “all websites”. Even though they had the single logout mechanism prioritized as fourth in their requirement list, the usability test results do not offer any information about how the users chose between the two options or how they behaved in the logout. In their other article (Sun et al., 2010a), the login process of the system is described in more detail. The authentication sessions are stored in cookies but handling the logout process is not described.

Cahill et al., (2011) have created a client-based authentication system that has a trusted hardware-based secure container for user credentials in the user’s device. The system allows both active and passive authentication. The latter means, for example, facial recognition using the device’s camera or an RFID badge that is near the device. The system requires much from the hardware and the user is bound to using one device. Mustafic et al. (2011) describe a similar system that combines SSO with continuous behavioral-biometric-based authentication. Their system uses keystroke dynamics as a proof that the user is still the same as that authenticated at the beginning of the session. If the system notices that the user has changed, it starts the SAML SLO. However, this only works for services that require typing. Stajano (2011) suggests a similar system where the user is continuously authenticated using physical tokens, but the system design is only at the conceptual level and not yet implemented.

Linden and Vilpola (2005) have arranged usability focus-group interviews for the staff and students of Tampere University of Technology about logout in the early days of web SSO, in 2005. The university was using the Pubcookie single sign-on system in its intranet services such as exam enrollment. Hence, the system was familiar to the users. In addition to the focus groups, they conducted a survey of 58 students regarding their opinions on the single sign-on system. Their study presented three conclusions: users expect single logout when using an SSO system, users want to know if they are anonymous or identified, and users would allow a single sign-on system to expand over organizational borders. The first conclusion, requiring single logout when using SSO, is the most interesting of the survey results from our point of view. The students who participated in the research even considered the SSO system to be broken if single logout had not been offered as a possibility and the other services were still active after logging out of one of the services.

5. Problems of logout in federated SSO systems

Logout in the SSO systems is not straightforward: should the user be logged off from one service or from all the other services as well? Linden and Vilpola (2005) claim that users of SSO systems expect single logout. Sun et al. (2011b) state that SSO systems should offer an option to end all the service sessions at once. In this section, we describe in more detail what kind of practical problems can be found in the services at Aalto University in Finland. The services use Shibboleth for single sign-on and for authenticating both university staff and students. Because Aalto University belongs to the Haka federation, users of other universities can also access some of the services, e.g. in collaborative research projects. Moreover, some of the services are common to all or a number of universities in Finland.
5.1. Only service session removed

Half of the Finnish universities use the Oodi1 student register e.g. to record course grades and for signing up for courses and exams. The front-page of the service reminds the users to exit from all browser windows when logging out to prevent the next user from gaining access to the first user’s information. The warning is needed since when the user logs out, only the service’s local session is terminated and both the IdP and SP sessions remain. The service returns to the service front-page and gives no information about logout status.

In the Oodi login process, two sessions are created: an SP session with an 8-h timeout and a local Tomcat session with a 30-min timeout. The Oodi service copies all information from the SP session to a service session presented by a JSESSIONID cookie. When the user logs out of the service, only the Oodi service’s own session cookie is removed but the SP session still remains, not to mention the IdP session. In the same way, when the local session reaches timeout, the service does not inform the SP about the logout. This means that the user can get directly back into Oodi by clicking on the login link. No re-authentication is required because the SP session is still valid. Furthermore, this works even if the user (via some other site) has also terminated the IdP session.

In this case, the biggest security problem is that the users cannot get out of the service at all if they do not close the web browser. Logging out of both Oodi and the IdP is not sufficient. Moreover, the service does not inform the users about the still valid sessions. Even if they close the web browser, both the SP and IdP sessions remain open until timeout. The problem is Shibboleth specific since the session management is divided between the SP and the service application.

5.2. SP Session removed but service session exists

The libraries of Finnish universities have a joint portal called the Nelli2 for access to scientific publications and databases. When the users log out of the service, the connection is redirected to the IdP. The IdP tells the users the they have logged out of the IdP session and that there might still be active sessions, and only closing the browser will end them. Indeed, there still is an active session, because if the user immediately returns to the Nelli portal and press the login button, the user is readmitted without re-authentication. However, she does not get into new services since the IdP session has actually been removed.

The Nelli portal is implemented using the MetaLib program that uses Patron Directory Services (PDS) for authentication (Pennanen, 2009). When the users logs in to the Nelli portal, the service checks first a PDS_HANDLE cookie. If it exists, its information is copied to the main session. Otherwise the service asks for the user information from the SP. Nelli also creates a local session key that is stored in a ML_SESSION_ID (MetaLib Session ID) cookie and included in all page URLs. If this cookie does not exist, it is always created; thus removing it does not affect anything. Removing the key from the URL has no effect either, if the ML_SESSION_ID cookie exists. The Nelli service session does not end until the user spend time navigating on other pages for more than 10 min or presses the logout button. In addition to the service specific cookies, SP removes its session, but because the IdP session is still valid, the next user can get in without authentication. Removing first the SP session eliminates the possibility of logging out of the IdP because the necessary information was stored in the SP cookie.

When the user logs out of the Nelli portal, the service does not remove the PDS_HANDLE cookie even though the SP session has already ended. The problem in this case is that the service session remains open. Thus the user can get in without the service checking from either IdP or SP if their sessions are still valid.

In a way, the logout of the Nelli portal does not work at all. Different sessions are active, even though the user thinks that they have been ended. The problem is Shibboleth specific, but session timeout can be a problem in other SSO systems, too.

5.3. Local logout

The Aalto University Wiki3 tries to execute local logout. When the user logs out of the wiki, the browser returns to the login page of the wiki service. Only one line of text “You have been successfully logged out. “Return to wiki.aalto.fi or login again” differentiates the page from the original login page that offers several options to log in. The user may not notice the text at all since it is very unremarkable mention among all the other text on the page. This may lead to confusion as to whether the logout has really happened.

Actually the Wiki service tries to execute the SAML2 logout instead of the HAKA logout since it asks the SP to redirect the connection to its own logout pages and not to the IdP. Because the current IdP does not handle the SAML2 logout, the SP removes only its own session and the rest of the session removals fail.

The problem is that the user is not informed of which session she has logged out of, and that this is due to compatibility problems. If the user closes the web browser, the IdP session and possible other SP sessions remain open until timeout. If the user does not close the web browser, the next user can get into all services using the same IdP session, even to the wiki. The problem is generic in the systems.

5.4. Local logout Together with IdP logout

The Noppa4 portal is used for publishing university course material on the web. It was developed at Aalto University and later brought into use in two other universities. When the user signs into the portal, both the SP and Noppa’s local Tomcat sessions are created. All user information is copied to a service session identified by either a JSESSIONID cookie or org.apache.tapestry.locale.cookie. The SP session is valid for 8 h, but if it is unused, it expires in 1 h. The Noppa portal offers only the Haka logout i.e. local logout together with the IdP logout.

The user can continue using other services even though the IdP session has ended because the IdP does not have the

---

1 E.g. https://oodi.aalto.fi.
2 http://www.nelliportaali.fi.
means to inform the other services to end their sessions. Thus, the real SLO is not executed. If the user closes the web browser while the other services are still active, their SP and service sessions remain on the server until timeout. The problem is generic in the SSO systems and may confuse users who cannot separate the real SLO from logging out of the IdP.

5.5. Summary of the practical problems

In Shibboleth, the service can keep track of the user session by itself or it can delegate the task to service provider software running on the same web server. This means that the client web browser may have session cookies from the service and SP, and, of course, from the identity provider. In practice, the services at Aalto University did not remove all the necessary sessions when the user pressed a logout button, not in the service itself nor at the IdP. Given this inconsistent logout behavior, it is understandable that many services remind their user to close the whole browser after using the service: there is currently no other way to guarantee that the connections are really closed.

Another problem we found was that the users could not predict what will happen when they press the logout button. Some of the services closed, or tried to close as described above, only their own session and the SP session, logging the user only out of the local service and not from the IdP. On the other hand, some of the services also terminated the session on the identity provider, meaning that they executed the Haka logout. There was no rule defined to determine which logout feature a service should invoke; rather it seems to be an independent implementation decision of each service developer. Furthermore, sometimes it is hard to see if the logout was successful and which form of the logout was performed. Expert users knowing the Shibboleth architecture could see from the URL if the Haka logout was used because the page shown after logout was located on the IdP; otherwise it was on the service web page. However, some of the services were working so poorly that, when they tried to execute SAML2 or Haka logout, they ended up executing local logout, and that would have confused the users even more. In addition to the security problem of the users not knowing which sessions are closed and which are not, the main benefit of SSO is partially lost because the users may have to retype their login credentials at unpredictable times when accessing multiple services simultaneously.

5.6. Requirements for logout

Based on the problems described above, we defined the following requirements for logout in SSO systems:

- The user must be able to log out of a single service (local logout).
- After logging out of a single service, the user must be able to continue using other services.
- The user should be able to log out of all the services once (single logout)
- The user must be able to easily distinguish local and single logout from each others when starting the logout procedure.
- After logout, the user should be able to know if the logout has really happened (and which form of logout has occurred).
- The service and IdP should be able to cope with the closing of browser without logout.

As mentioned earlier, there are two types of SSO systems: the first type, e.g. Shibboleth, allows users to get into all services by authenticating themselves only once, and the second type, e.g. OpenID, allows users to log in to many services using the same credentials, but the credentials are asked again for each service. The latter does not have an IdP session, which means that the user cannot log out of the IdP nor close all services once. However, the users should be able to distinguish this situation from the single logout if they use both types of SSO systems.

In general, local logout is more relevant on the service side than for the user. All open sessions burden the service and if the users are not going to use the service anymore, they should be able to close the service session and, in this way, inform the service that it is no longer needed. On the other hand, unfinished things also burden the limited work memory of the users, and closing a service session may feel more secure for them even though they could easily get back with the help of IdP session. Later in Section 7, we observe how some users like to group services based on their security requirements and want to close the services with high security requirements as soon as possible. In addition, local logout is the only option in the SSO systems that uses authorization, e.g. in OAuth.

6. Solutions for logout in federated SSO systems

In centralized SSO systems developed from closed organizations such as businesses, an IdP typically keeps track of the services to which the user has been authenticated and communicates the logout to them through a backend channel such as a SOAP call. Thus single logout is possible. In contrast to centralized systems, federated SSO systems do not usually use backend channels for communication and an IdP does not know about active sessions on the SPs. However, there are several steps that could make the logout in the federated SSO systems more secure. At least, the user should be able to log out of the service that is being used and also end the IdP session when desired. Altogether, we have identified the following steps that should be considered for improving the security and usability of logout in the federated SSO:

- fixing and avoiding obvious bugs,
- enhanced cookie management in browsers to enable group deletion,
- unified and standardized process for ending service sessions,
- clearer and uniform user interface for logout, and
• browser-sent keep alive messages.

In the following sections, we discuss these in more detail. Then in Section 6.6, we describe our solution, a polling mechanism to check for the existence of an IdP session.

6.1. Fixing and avoiding obvious bugs

When a new service is integrated into the Shibboleth SSO system, its code has to be modified. The developers should decide if the service maintains the sessions itself or does it use the sessions provided by the SP. If the SP takes care of the sessions, the service just need buttons for the local and IdP logout. Pressing these buttons starts the logout procedures with the proper attributes in the SP, and the SP takes care of everything from there.

If the service takes care of the session and logout itself, the procedure is mostly the same as above but the service itself must take care of some additional tasks. First of all, the service has to remove its own session before calling the logout method of the SP. Equally important is having a handler for the IdP-initiated logout. For this, the service should register a single logout handler for the SP. When a SLO request arrives, the service has to remove all sessions for the user. The request can come through the web browser or using a SOAP interface. In the latter, it is not possible to remove the cookies from the browser, but the service can remove the session state from its own memory or database. Finally, the lifetime of the service session should be shorter than the lifetime of the SP session because otherwise the incoming logout requests do not work correctly.

6.2. Enhanced cookie management in web browsers

Cookie management in web browsers is mostly beyond the scope of this article but we make some brief notes here. Handling of the cookies has not changed much during the recent years. To mitigate problems caused by attacks, there are rules how cookies should be handled. These rules prevent the IdP and SP from seeing each other’s cookies.

For making single logout possible, the rules could be changed. For example, the IdP could create a collection of cookies connected to the SSO session. The SPs could add their own cookies to the collection. Then as a creator of this collection, the IdP could remove cookies and set the deletion rights to all servers of the federation. In the single logout, the IdP could remove all the SP cookies that are left over. However, changes in the cookie management may cause unexpected security weaknesses if not done with special care. Such changes could be considered, for example, in the standardization of HTML 5. However, the current model of session management with cookies is not likely to change quickly. Moreover, if the IdP keeps track of all SPs to which the user has logged in, there are implications for user privacy.

6.3. Standardized process for ending service sessions

A unified process for removing the session cookies and managing their timeout periods would prevent the current problems of logout. The SAML single logout is too complex and contradictory to work properly. For example, the order of the session removal is not clear. The standardization should redefine the single logout process because the currently used federation specific definitions are not sufficient.

Federations that use Shibboleth often provide test functionality for the service providers when they want to add their services to the federation. For example, for a service to join the Haka federation, administrative checks are required to ensure that the service does not violate the purpose of the federation and that the personal information of the users is in safe hands and really necessary for the service (CSC, 2013). However, joining does not require testing of the service. Based on our results, it is clear that the checks should also include the testing of service login and logout functions. In addition, implementation mistakes could be avoided if the federation provided examples of login pages as well as logout functionality and strongly recommended their use in the services. These recommendations should also include examples like how and where the functions should be specified in the configuration files.

6.4. Clearer and uniform user interface

It is hard for a user to separate a local and single logout and understand their differences when some services of the federation provide the first and some services the latter. It would be nice if the user could see on one page the services where the authentication sessions are still valid. Technically and from the user-interface point of view this is hard to arrange.

Especially important for the user is to know that the global single logout has really removed all the session information from all the services. Closing the web browser ends the sessions on the client side and solves the problem almost to the point of satisfaction. The sessions on the server side remains until timeout but they cannot be used without knowing the session cookies. The main danger is that the browser somehow leaks the cookies, e.g. by writing them onto the disk or by not using encrypted connections. Another problem arises if the user also has other web browser windows and tabs open and does not want to close them, but just logs out of one of the services. Web browsers becomes generic execution environments, akin to operating systems, and many uses simply never close the browser windows. At minimum, the web browsers should be improved to allow the ending of sessions without closing the whole browser.

For example, the Microsoft Outlook web application asks the user before logging in whether the client is a private or public computer. This affects the timeout for the session when the user is inactive. In the same way, the login to a service could give two options that finally lead to either local or single logout.

A unified user interface means that a federation or the SSO system standardization should define from the user’s point of view how the services should end the sessions. Because the user can work either with public or private computers, two options should be provided by all services: single logout that ends both the IdP and all service and SP sessions, and local logout that leaves the IdP session active. Also, all services should inform the user from where she has logged out.
In our test services, we offered two logout options for the users: local logout and single logout. This required defining two different logout initiators in the shibboleth2.xml configuration file, as shown in Fig. 3. In the logout, the connection is redirected to the corresponding location URL of the service provider. Two of the three test services were using the polling mechanism, and the third one closed the IdP session together with the local session if the user chose single logout, as is done in the Haka logout.

6.5. Browser-sent keep alive messages

Single logout changes the actions of the IdP and SP. For example in the original Shibboleth, the IdP takes care only of the validity of the IdP sessions, and the SP has the necessary user attributes in its memory. The IdPs and SPs could have been rebootted since all necessary state information has been stored in the cookies on the user’s web browser and the other information could have been fetched again. For single logout, the IdP has to store information about all SPs where the user is logged in. This would mean that, in addition to transitory information that can be lost, the IdP also holds security-critical state data.

An alternative automatic logout mechanism that avoids creating such issues for the IdP is based on AJAX. The web browser could send keep alive messages to the SPs and IdP when the user has an open window for a service. When the user navigates to some other web page than that of the service, the browser simply stop sending these messages to the SP, which can then end the user’s session. This way, the sessions will not be “left behind”. Moreover, when the user has moved out of all the services of the federation, the web browser stops sending the keep alive messages to the IdP, too. Then the IdP can remove the IdP session and, to make single logout absolutely sure, start the IdP-initiated logout for all SPs. However, there are still two problems: This approach burdens the IdP and SP with additional keep-alive messages, and it does not work if JavaScript is turned off or the web browser stops sending the keep alive messages to the SPs and IdP.

6.6. Polling mechanisms to check for the IdP session

Compared to centralized SSO systems, a real single logout is seldom offered by open SSO federations because the IdP does not keep track of the services. Thus, we needed to design our own solution for single logout. We implemented a simple polling mechanism for a service to check whether the IdP session is still active or not. This allows single logout for the services in a federation — but only if the user has used the same IdP to log in to the services and if all the services implement the polling mechanism. If the user has used multiple IdPs, the single logout is limited to the services that used the same IdP for authentication.

The polling mechanism works as follows. The users log in to a service that uses the Shibboleth SSO system as they would log in to any service. The system creates a user sessions for them in the SP, possibly also in the service, and in the IdP. The Shibboleth SP software keeps a timer for polling the IdP. For testing, we used a very short timeout period, only 10 s, but in the real service the timeout could be couple of minutes to avoid overloading the IdP. When the timer runs out, the SP first ends the service and its own sessions and then sends a message to the IdP to ask if the session is still active. This message used is actually the same message that the SP sends at the beginning of the user session to the IdP. If the IdP session is active, the IdP sends back the credentials and the SP starts its own and the service sessions again. Most of the time, the user does not see any difference compared with a service that does not use polling, and she can work normally in the service. The user observes the difference only if the IdP session ends, either because the user logs out of another service that also terminates the IdP session, or because the IdP session has reached its timeout. The IdP sessions are typically very long, for example 8 h, whereas service sessions are short.

In the polling, if the IdP session does not exist, the service closes its session permanently, and the user will be prompted with a new authentication request upon reloading the service web page or clicking any action on the service. This prototype shows that the polling can be made work from the security point of view.

The main benefit of our solution is that it is simple and that no new messages are required. It is easy to deploy in service providers and requires no modifications of the IdP. The amount of code needed at the SP is minimal as it amounts to only timing out the local sessions.

A drawback is that, because the service kills its own session first, the information has to be stored somehow in the case when the IdP session is still active and the user wants to continue using the service. This works well in services that follow the RESTful design principles but not in highly stateful services. The current polling implementation also causes usability problems in applications that have not been designed to handle the SP and service session termination and refreshing gracefully. Such applications may lose unsaved data, such as unsubmitted web form contents, and would need to be modified to support a more asynchronous style or communication. Moreover, if the timeout interval is too short, the IdP can be overburdened by the polling. The polling is not necessary when the web page is idle and no new data is retrieved from the server, but modern web applications often involve almost continuous communication between the client and the server. Another drawback is that if all the services in the federation do not implement the polling mechanism, the user has no way of knowing when the single logout really works for all services and when it logs out of a few services only.

6.7. Summary of the solutions

This section has described two types of solutions for logout in federated SSO systems: the first ones are more guidelines and two last ones are technical solutions based on either keep-alive messages or polling of the IdP. Most of the guidelines...
are at the general level, e.g. avoiding bugs and creating a clear user interface should always be followed. Cookies are the way to provide session state in web services even though there are a lot of problems in their use – developing better cookie management could be one way to improve the SSO systems. For the technical solution, we have chosen the polling because it is simpler and requires less modification of the SSO architecture and implementations.

Next, we discuss the usability test that was conducted in order to find out what users think about SSO and logout.

7. Single logout usability test

Usability is part of system acceptability (Nielsen et al., 1993). It consists of learnability, efficiency, memorability, satisfaction and avoiding errors, all which are important also from the security point of view. The most used method for testing the usability of a system is a usability test: a group of users perform a set of tasks using the system in order to improve the user interface of the system in its iterative development process, or the usability test can be used to compare alternative solutions in order to find which one is the best. Typically, the users perform the tasks that represent the normal use of the system, and the users “think aloud” and describe what they are doing in the test. After performing the test tasks, the users are interviewed to find out more about their thoughts.

Even though the users give their subjective opinions of the system, a few users can spot the most significant problems in the system. The purpose of the usability test is to find problems and understand how users behave, not to produce statistically accurate quantitative measurements of the detected problems. The test is considered reliable if changing the test user group does not change the result of the test. According to Nielsen, often only 10 users are enough to get reliable results (Nielsen et al., 1993). Sauro et al. have developed a formula estimating the required test user group size. For example, if the probability of a problem occurring is 25% and the test user group has 17 users, at least one user spots a problem with 99% probability. With occurrence of 10% and 19 test users, at least one user spots the problem with 85% probability (Sauro and Lewis, 2012). In order to create a valid usability test, the user group should be chosen from among users of the actual system and not from some other field, and the test should represent the actual tasks performed in the system. In our usability test, we had a group of 18 frequent SSO users. The tasks chosen for our test were such that the users encounter them also in their real life at work.

The goal of our usability test was to find out what the users do when they log out of services and whether they can understand what happens. In the test, we decided to offer the users two ways to end their sessions in a service: they could either log out of the currently used service or they could log out of all the services that they had opened during the SSO session. The first feature we called logout, but a good term for the second one was much harder to find. Our pilot test usability expert said that the term “single logout” would not be understood as logging out of all the services, but only of a single service. Thus, we ended up having different term in each service. After the users had tried the system, we also asked them which terms would have been the best for describing the action. Next, we explain the usability test in more detail.

7.1. Usability test participants

We had 18 participants, nine male and nine female, recruited from the staff of Aalto University. They did not get anything as a reward for doing the test. Most of them are working as support staff. Four of the participants were researchers with a doctoral degree in computer science or information system science. Of the others, six had a Bachelor or other college degree and eight had a Master’s degree. Five were aged between 25 and 34, nine 35–44, two 45–54, and two were over 55 years old. Eight characterized themselves as ordinary users, seven as professionals, and three as experts in computer science. None were experts in computer security. All our test users were native Finnish speakers, but all the user interfaces of the tests were in English. They were not asked about their language skills, but all of them use English in the regular course of their work at the university. Table 1 gives a summary of the participants’ background.

All the participants had prior experience in single sign-on systems because the services of Aalto University use Shibboleth for authentication. Ten characterized themselves as ordinary users of computer security, and eight as professionals but none as experts. In addition to the Shibboleth SSO, seven had used either Facebook or Google to sign in to some other service. Only two had used any other single sign-on systems than these.

As background information, we also asked how many services they sign in to during an average day, how many passwords they have, and how they cope with remembering the passwords. The findings are also listed in Table 1. Six users said that they log in daily to 3–5 services, ten to 6–10 services and two log in to over 10 services. On average, this means that users log in daily to 6–10 services. The result is the same, an average of eight daily used passwords, which Florencio et al. also found in their research (Florencio and Herley, 2007). Three of our users claimed that they had 3–5 passwords, five had 6–10 passwords, two had 10–15 passwords and eight had over 15 passwords, which means that the users had on average 10 to 15 passwords. Some of our users mentioned that using single sign-on at work has reduced the number of their passwords. Three used password managers offered by a web browser for some services. Three used the operating system’s key ring or their own encrypted files and scripts for storing passwords because they did not trust the browsers. Nine users

| Table 1 – Background of the participants. |
| Age | 25–34 | 35–44 | 45–54 | Over 55 |
| 18 users | 5 | 9 | 2 | 2 |
| Degree | Bachelor or college | Master | Doctor | Other |
| 5 | 8 | 4 | 1 |
| Number of daily used services | 3–5 | 6–10 | Over 10 |
| 6 | 10 | 2 |
| Number of passwords | 3–5 | 6–10 | 10–15 | Over 15 |
| 3 | 5 | 2 | 8 |
occasionally used the same password for several services, but they have categorized these services as not critical (e.g. movie ticket shop), and they did not use the same password for work and leisure-time services. Six had problems remembering passwords. The users told us that they made remembering easier by recycling passwords, by using partly the same password on several services, by using some familiar word, a number series or a sentence as the basis for forming a password, or by writing the password down for at least at the beginning of its use.

7.2. The test setup

In the usability test, the user performed three tasks. The scenario was that they were working at the university, had come back from a business trip and had to claim travel expenses. They first had to log in to a “Travel expense service” and add the travel cost information to a form as shown in Fig. 4a. All the information for the travel bill was not given in the first task. The daily allowance had to be checked from another
service, “Aalto Intranet” shown in Fig. 4b. The users had to log in and check the amount, and then they were asked to close the service and go back to the travel billing service. As is the practice in usability tests, we avoided using the words shown in the user interface, so that the users had to decide themselves what action was required in the task.

Then, while completing the travel bill, they had to use the “Room reservation service” shown in Fig. 4c because a professor called and asked them to book a room for a meeting. After making the room reservation, they were asked to close the service pages, since otherwise their computer would be slower. At the end of the test, i.e. after submitting the travel bill, they were asked to close all the services and to log out of the computer because a system administrator would come to install new software during the lunch break.

All the services in the test offered two logout options: local logout and single logout. As mentioned above, before the actual user tests, we had a usability expert check our test. Based on the suggestions, we decided to also test different wordings because she pointed out that security terminology can be unfamiliar to the users. Thus, all the single logout alternatives had different options: the travel billing service used “global logout”, the intranet service used “single logout”, and the room reservation used “SSO logout” for logging out of all services. The acronym SSO was chosen because it is used by the real services of Aalto University on a page that is shown to the user when logging out. All the logout alternatives were positioned in the upper right corner where logout buttons normally exist in the services. Both logout options were always visible, so that the user did not need to open any menu to see the options.

### 7.3. The test results

In the end interview, we asked if the users had noticed the logout options and how they had understood them. We also collected all the background information as well as their opinions on using the single sign-on system.

We asked the users to give their opinions on the nine claims that are shown in Table 2. They could choose between whether they fully disagree, somewhat disagree, do not agree nor disagree, somewhat agree, fully agree with the claim, or if they cannot answer the question. In answer to the first question, the test users said that they knew how the single sign-on system worked. It was emphasized in the interview that the claim meant understanding that there is a separate service for authentication and that the authentication is valid for many services, and not that they should know the technical details of how SSO works. In answer to the second question, the users trusted the SSO technology to provide authentication for the services. For them, the organization providing the authentication mattered more than the one offering the services.

We partly used the same questions as Linden and Vilpola (2005) in their study. These questions are marked with * in Table 2. Compared to their study, most of the claims got similar answers. Our users wanted a little more similar layout for the services than their users. Two of our users said that the layout should be similar for logging in and out, but the layout of the service itself did not matter. Also, our users wanted a little more strongly that their name or username should be visible while using the service, to show whether they are using the service anonymously or as identified. However, there were two major differences. Our users were much more reluctant to use separate passwords for services. More importantly, our users did not want the single sign-on system to log them out of all the services when they log out of one service, whereas the users of Linden et al. wanted that to happen. This may be because of the different user background: our users were university staff that use personal computers and many services simultaneously throughout the day. The users of their quantitative study, on the other hand, were students and, in 2005, personal laptops were not as ubiquitous among students as they are today. Another reason may be that there are now more web-based applications and keeping them continuously open has become a common practice. Moreover, the single sign-on systems are more familiar now as they have been in use at universities for several year.

Table 3 shows the number of selected logout actions during the tasks in our test. In the second task where they were asked to close the intranet service, three users chose the local logout and left the browser window open, and two chose the

### Table 2 – Claims about single sign-on systems.

<table>
<thead>
<tr>
<th>Claim</th>
<th>Average (range 1–5)</th>
<th>Standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>I know how the single sign-on system works.</td>
<td>3.06</td>
<td>0.91</td>
</tr>
<tr>
<td>I trust the functioning of the single sign-on system.</td>
<td>4.33</td>
<td>0.58</td>
</tr>
<tr>
<td>I rather log in separately to every system than use single sign-on.*</td>
<td>1.33</td>
<td>0.58</td>
</tr>
<tr>
<td>Using of single sign-on should be limited to the services of one organization (e.g. the university)</td>
<td>2.93</td>
<td>1.18</td>
</tr>
<tr>
<td>If I log in to a service of another organization using single sign-on, the system should ask again for the password.*</td>
<td>2.81</td>
<td>1.29</td>
</tr>
<tr>
<td>The organization that provides the service matters, when single sign-on is used.*</td>
<td>4.00</td>
<td>0.96</td>
</tr>
<tr>
<td>The organization offering the authentication matters.</td>
<td>4.20</td>
<td>0.86</td>
</tr>
<tr>
<td>All services using single sign-on should have a similar layout.*</td>
<td>3.06</td>
<td>1.30</td>
</tr>
<tr>
<td>I want to know if I use the service anonymously or as an identified user.*</td>
<td>4.39</td>
<td>0.76</td>
</tr>
<tr>
<td>My name or username should be always visible when I have logged in to the service.*</td>
<td>4.00</td>
<td>0.88</td>
</tr>
</tbody>
</table>

*Stand means that the claim meant understanding that there is a separate service for authentication and that the authentication is valid for many services, and not that they should know the technical details of how SSO works.

---

In the second task where they were asked to close the intranet service, three users chose the local logout and left the browser window open, and two chose the
single logout, of which one left the browser window opened and the other closed it. Thirteen users just closed the browser window, and only three of them were left wondering which logout possibility they should choose. The others did not take into account the logout options at all. In the third task where the users should close the room reservation service, five users chose the local logout and one chose the SSO logout, whereas twelve just closed the window. In the last task, closing the travel expense system, six users chose the global logout and left the window opened, four chose the local logout and left the window opened, and two just closed the window. At the end of the test, seven closed the browser first and then logged out of the operating system, but eleven just logged out of the operating system without closing the browser separately.

After the background information interview, we asked if the users noticed the logout alternatives. To start with, three of the users did not notice the logout buttons at all in the tasks where they were asked to quit the intranet and room reservation services. They noticed the buttons only in the last task when they were asked to close all services and log out of the computer. Seven missed all the logout alternatives in all the services; they only closed the browser or logged out of the computer directly. Eight noticed the logout possibilities but most of them were confused about which to choose and, as a result, some only closed the browser window without logging out and some just guessed what to click. In the interview, they explained their choices as follows.

- “Global logout sounded reasonable for logging out of all services, but, for me, single and local logout seemed similar.”
- “I did not understand, but in the end, global logout sounded a natural way to finish.”
- “I did not ponder the difference, I just pressed the first button which said logout.”
- “I did not ponder much. For me, local meant this, but I did not understand what the global would be linked to.”
- “I thought about it hard but I could not understand, so I just closed the browser window. Afterward, local could be understandable but global, no.”
- “I did not understand, so I just clicked one of them to see what happens.”

Only three out of the 18 users followed the expected path by using the local logout for the intranet and room booking services, and the global logout for exiting the travel billing system. Moreover, they were not those who claimed to be experts in computer science. It was a surprise that almost half of the users (44%) just closed the browser window without logging out of the service.

In the end interview, we also asked what they do in their real work when they finish using the service. Nine said that they log out of the services, six that they do not log out of the services, and three answered that it depends on the service: they log out of financial or HR services but not from the teaching and research-related intranet services. Eleven claim that they had read the web page that is shown after logging out of a service — it usually says that the user should close the browser in order to reliably get out of the service. Only three users close the browser after using a service, 10 close only the browser window, and five do not close the browser at all. Seven of those who do not close the browser said that the other web browser windows or tabs usually have unfinished tasks. One did not know what to do in order to act securely, and one said that he/she is just lazy and reopening the browser would be too slow. Five close their browser at the end of workday, and three rely on locking the screen. These actions were quite similar to how they acted in the usability test.

Finally, we asked if the users have encountered a surprising situation where they have had to authenticate when they thought they were already logged in. Thirteen out of eighteen remembered such a situation. Six users said that the logout itself was not the surprise, but it was the short timeout period the service was using. Two users said that they had not found any logic in how the services sometimes automatically ended the sessions, and they mentioned Facebook and Google as examples of that. One supposed the reason for the unexpected re-authentication was problems with Virtual Private Network (VPN). On the other hand, seven users have been surprised by a situation where they were already logged in on a service when they thought they had not yet been authenticated. The users said that this may have been because they had closed the browser window but not logged out, or because they had forgotten that they were already signed in. An example mentioned by two users was Google: users were surprised that they were authenticated users in Youtube when they had another browser window with Gmail open. This means that they did not know that the services belonged to same SSO federation. One user described a situation where the web browser had crashed and he/she had to boot up the computer, and still he/she could get into the service. The reason might be that the web browser had had no time to remove the cookies, but they still existed due incomplete closing of the browser.

### 7.4. Suggestions based on the usability test

The usability test showed two major problems: users do not log out of the services before closing the web browser window and they do not understand the wording used by security professionals for different logout options.

<table>
<thead>
<tr>
<th>Task and service</th>
<th>No closing</th>
<th>Only browser tab</th>
<th>Local logout</th>
<th>Local logout, tab closed</th>
<th>Single logout</th>
<th>Single logout, tab closed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Travel bill service during the test</td>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Close the intranet service</td>
<td>13</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Close the room reservation service</td>
<td>12</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Close the travel bill service in the end of the test</td>
<td>6</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3 – Logout of the services in the usability test.
When the users do not log out from the services, it means that the service session remains open on the server side until timeout closes it. This is not the most secure way of closing the sessions, and it requires memory on the service side. Moreover, if the users leave the service open and only close the browser window when using a personal computer, they may do the same when using a public computer because they are accustomed to act that way. As a solution, the service developers should react to the closing of the browser window as a user logout of the service. Web pages can be programmed to detect the closing of the windows and take action before the page is actually closed. However, many users do not want to exit the web browser at all since it has become an execution environment for all types of services and acts as a workspace for the applications. This is in contradiction with requirement to exit the browser in order to really logout from a service. It means that exiting the browser cannot be the only way to log out of a service.

The users do not understand the words used for describing the different logout options. They understand the concepts of single sign-on and both the local and single logout, but they do not know which term describes the action they want to perform. It would be better to call the local logout by its service name, for example, “log out of ServiceX” or by using an expression such as “log out of this service”. The single logout should be called “log out of all services”, even though the wording is quite long. Our users said that “global logout” describes better than “single logout” the case where the user wants to log out from all services. However, it may be misleading because the logout applies only to the services that use the same identity provider and not to all possible services in a multi-IdP federation.

In our usability test, one reason behind the confusion between single logout and logging out of one single service may be the language and culture of the Finnish users: we used the Finnish term “kertakirjauminen” for the single sign-on but mentioned also the English word “single sign-on” that should be known for the users from their daily work. For Finns, it may be confused with “single-use” (in Finnish “kertakäyttö-”), and because Finns are used to authenticating themselves to online banking with one-time (single-use) passwords (The Finnish Bankers’ Association, 2005) (in Finnish “kertakäyttösalasana”), which does not provide single sign-on.

Often, educating the users is seen as a solution to security and usability problems, but who will do that for Internet wide services? An organization using single sign-on can train its users, but for the general public, education is hard to provide. We can only assume that the users will learn the importance of logging out while they use the services and ask other, more experienced users when something does not work as expected. This means that the service developers have to offer the users solutions that do not lead to problems even when the users have not understood the security paradigms behind the service. One part of usability is that the users should be able to avoid errors and recover from them (Nielsen et al., 1993). This is even harder in the field of security where actions cannot be revoked afterward.

8. Conclusions

Ending service sessions is almost as important as the beginning of the session that starts with user authentication or authorization. If the user does not get out of the service safely, her privileges might be misused. Current web single sign-on systems do not provide clear termination for the user sessions. Unclear federation of the services can lead to situations where the users are often surprised when they are logged in to services automatically. Another way to surprise the users is that they need to re-authenticate when they think they were still logged in on a service, for example, because logging out of another service has unexpectedly terminated their session also on other services.

In this article, we have investigated logging out from web-browser-based services that use Shibboleth SSO for the authentication of the users. Many of the problems that we found in the services are easily solved by careful implementation and testing of the services. An SSO federation could provide common guidelines and require testing before accepting services. Especially the session management between the service itself and the service provider software component of Shibboleth requires attention from the developers. A unified process for removing the session cookies and managing their timeout periods would prevent problems of logout in SAML and Shibboleth, but we did not deal with the cookie management improvements in this article.

The end of the authenticated service sessions has not gained the attention it deserves. We tested many services that use Shibboleth as an SSO system and almost all of them had some problems with logout. Because some of the session information remains when the service ends in partial logout instead of local logout, the next user of a shared computer can access the service with the previous user’s account. The single logout does not work either. Even though the IdP removes the IdP session, the other service and SP sessions remain active. Often services rely on closing the web browser but, even then, the server side sessions remain until timeout. Small changes such as unified logout process and user interface would allow the services to provide at least local logout instead of partial logout. Furthermore, an SSO federation should have unified user interface for login and logout to help users to understand how they work.

In order to understand the user behavior during logout, we implemented a set of services that offered two logout alternatives: local and single logout. For the single logout, we created a polling mechanism that checks whether the IdP session is still active. If the user has used single logout in a service, the other services using the same IdP will terminate their sessions, too. We concluded that the service sessions urgently require a standardized process for ending the sessions with a unified user interface. Another option for providing single logout would have been sending keep-alive messages from the service to the identity provider, but it would have burdened the IdP more than polling because it would require the IdP to keep track of the service sessions in addition to its own session.

Our usability test end questionnaire results show that the users understand the concepts of single sign-on system, single logout, and local logout. They wished for similar layout for the services, but emphasized that they meant the login and logout procedures in particular, not necessarily the service itself. More importantly, our usability test reveals that many users do not log out of services; they often just close the web
browser window. This leaves the session open at the server end. It means that the service developers should also react to the closing of the window, not just to the explicit logout. However, many users do not want to exit the web browser at all since it has become an execution environment for many services. This means that the service providers should offer other ways of logging out of the service than just exiting the browser.

In the SAML-based Shibboleth, where the IdP maintains its own session to avoid the need to re-authenticate the user, logout can be either from the service only or from the IdP, too. In OAuth, the identity provider does not necessarily keep a service-specific authorization token. Thus, only local logout from a service is possible. Currently, many services do not offer logout at all. Because many users use both types of SSO federations, offering logout options is important for creating a unified user experience. Moreover, our usability test showed that the users do not understand the terminology “single logout”, “global logout” or “local logout” in the way the security experts use them. This means that the service developers should label the logout options carefully, for example using longer descriptions such as “log out of all services” for global logout and the name of the service for local logout.

Acknowledgments

We thank Mari Tyllinen for helping to design high-quality usability test and Andrey Lukyanenko for helping with statistics.

REFERENCES


Hardt D. The OAuth 2.0 authorization framework. RFC 6749, IETF; October 2012.


Sauro J, Lewis JR. Quantifying the user experience, practical statistics for user research. Elsevier; 2012.