Aalto University
School of Science and Technology
Faculty of Information and Natural Sciences
Degree programme of Computer Science and Engineering

Pekka Silvekoski

Client-side migration of authentication session

Master's Thesis
Espoo, February 14, 2010

Supervisor: Professor Tuomas Aura, Aalto University
Instructors: Sanna Suoranta Lic.Sc.(Tech.), Jani Heikkinen M.Sc.(Tech.), Aalto University

Aalto University
School of Science and Technology
Faculty of Information and Natural Sciences
Degree Programme of Computer Science and Engineering

ABSTRACT OF
MASTER'S THESIS

Author: Pekka Silvekoski

Title of thesis:
Client-side migration of authentication session

Date: February 14, 2010 **Pages:** 10 + 71

Professorship: Data Communications Software **Code:** T-110

Supervisor: Professor Tuomas Aura
Instructors: Sanna Suoranta Lic.Sc.(Tech.), Jani Heikkinen M.Sc.(Tech.)

The internet has changed to a more mobile environment where users alternate between desktop and mobile devices. The sessions the user starts on one device do not automatically move to a new device when the user changes to it. Session migration allows the user to move the sessions with him.

This thesis concentrates on the migration of authentication sessions. Web-based authentication services create an authentication session with the user that is used to re-authenticate the user when he moves on the internet. Migrating the authentication session with the user removes the need for the user to log in again on every service in the new device.

The focus of this thesis is studying single sign-on (SSO) authentication systems and their session handling. Implementation is a prototype application that migrates authentication sessions of different SSO systems. Furthermore, this thesis discusses what needs to be changed on the client and server-sides of the transfer for the migration to work smoothly.

Migration of internet sessions is a studied topic. The whole internet session from a browser has been migrated with different means. This thesis focuses only on a one part of the internet session namely the authentication sessions. When only the authentication session is migrated, the other sessions do not interfere with the studying of the authentication session migration.

Keywords: authentication, session, SSO, mobility, identity, migration
Language: English

Aalto-yliopisto
Teknillinen korkeakoulu
Informaatio- ja luonnontieteiden tiedekunta
Tietotekniikan koulutusohjelma

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä: Pekka Silvekoski

Työn nimi:
Client-side migration of authentication session

Päiväys: 14. helmikuuta 2010 **Sivumäärä:** 10 + 71

Professuuri: Tietoliikenneohjelmistot **Koodi:** T-110

Työn valvoja: Professori Tuomas Aura
Työn ohjaajat: Tekniikan lisensiaatti Sanna Suoranta,
Diplomi-insinööri Jani Heikkinen

Internet on muuttunut yhä liikkuvammaksi ympäristöksi. Käyttäjät vaihtavat pöytäkoneiden ja mobiililaitteiden välillä. Käyttäjän yhdellä laitteella aloittamat sessiot eivät siirry uudelle laitteelle, kun käyttäjä vaihtaa sille. Istunnon siirtämisen avulla käyttäjän istunnot voivat siirtyä käyttäjän mukana.

Tämä opinnäyte keskittyy autentikointi-istuntojen siirtoon. Web-pohjaiset autentikointipalvelut luovat käyttäjän kanssa autentikointi-istunnon, jolla käyttäjä autentikoidaan, kun hän käyttää Internetiä. Autentikointi-istunnon siirto mahdollistaa käyttäjälle laitteen vaihdon ilman, että hänen täytyy kirjautua uudelleen jokaiseen palveluun.

Tässä diplomityössä tutkitaa kertakirjautumisjärjestelmiä ja miten ne käsittelevät istuntojaan. Toteutuksena on prototyyppi, joka siirtää useamman kertakirjautumisjärjestelmän autentikointi-istuntoja laitteelta toiselle. Tutkinnan kohteena on myös miten asiakaspäätä ja palvelinpäätä autentikointi-istunnossa on muutettava, jotta autentikointi-istunnon siirto toimii saumattomasti.

Internet-istunnon siirtämistä on tutkittu paljon. Eri tapoja on käytettä koko web-selaimen istunnon siirrossa. Tässä työssä keskitytään vain yhteen Internet-istunnon osaan, nimittäin autentikointi-istuntoon. Kun siirretään pelkästään autentikointi-istunto, muut Internet-istunnot eivät häiritse autentikointi-istunnon siirron tutkimista.

Avainsanat: autentikaatio, kertakirjautuminen, identiteetti, migraatio, mobiilisuus, sessio
Kieli: englanti

Acknowledgements

First, I would like to thank my supervisor prof. Tuomas Aura, and instructors Lic.Sc.(Tech.) Sanna Suoranta and M.Sc.(Tech.) Jani Heikkinen. The instructors helped me a lot with the topic of the thesis and our weekly meetings helped me to finish the thesis in timely fashion. They also allowed me to loan equipment for development and testing of the implementation in this thesis.

I also want to thank my family and friends. Niko Laaksonen for proof reading most of the text and providing useful tips with English language, my sister D.Sc.(Tech.) Maija Honkela for reading the thesis and providing useful advice and of course my mother for pretty much everything.

Espoo January 16th 2010

Pekka Silvekoski

Abbreviations and Acronyms

AS	Authentication Server
GTK+	GIMP Toolkit
GUI	Graphical User Interface
IdP	Identity Provider
OBEX	Object Exchange
QoS	Quality of Service
RFCOMM	Radio Frequency Communication
SAML	Security Assertion Markup Language
SDK	Software Developers Kit
SGT	Service Granting Ticket
SP	Service Provider
SPP	Serial Port Profile
SSO	Single Sign-on
TGS	Ticket Granting Service
TGT	Ticket Granting Ticket
WEP	Wired Equivalent Privacy
WLAN	Wireless Local Area Network
WPA	Wi-Fi Protected Access

Contents

List of Tables

List of Figures

Chapter 1

Introduction

On computers, user usually starts his first session by turning on his computer or logging into a workstation. When user starts different programs, all of them have their own sessions and session information. The sessions change during the user's work. If a user changes from one device to another, all these sessions have to be started from the beginning.

Sessions are especially important in the internet. The web services create sessions with users to identify the individual user's connections from other HTTP connections. The web servers keep track of the sessions with the state management protocol of HTTP that uses cookies to store information on the user's machine. If the web service wants to give protected information to the user, it needs to confirm the user's identity. The user's identity authentication is usually done with a separate authentication service such as a single sign-on (SSO) system. The SSOs have their own sessions with users just as other web services.

Session migration allows the user to continue his sessions on a different machine. Session migration is a studied topic. For example, Internet browser [10] and multimedia sessions [31] have been migrated. The session migration applications transfer the session information from one device to another. The migration allows internet browsing or a multimedia session to continue from the same spot on different devices in a mobile environment. For example, a user can start to watch a movie from the internet on his mobile device while travelling on a train. When the user arrives at home, he transfers the movie session to his home entertainment system that is connected to the internet and the movie continues from the same spot it was left off on the mobile device.

Multitude of different single sign-on (SSO) systems exist that offer authenti-

cation services on the internet. SSO allows the user to authenticate himself to different services with one login. The SSO systems have different ways for user authentication and identity information storage and passing the information between the actors of the authentication system. Different levels of authentication are used depending on the security requirements of the accessed resources.

One goal of the SSO systems is to increase the usability of authentication sessions for the user by diminishing the amount of logins users have to do. Being able to migrate the authentication sessions of the SSO systems adds even more usability. Eventually, the authentication sessions and thus identity information moves with the user from one device to another without users having to logout and login on every device for every service.

1.1 Use case scenario

Alice works in a company that has several different services accessible through the web with a web browser. She is also enrolled in the local university which also has many web services. Both the workplace and the university have their own single sign-on (SSO) systems. With the SSO systems Alice only has to remember one user name and password for the workplace and one for the university to use all of their services.

In the morning, Alice is drinking her coffee at home and she is logged on to the SSOs of her workplace and university. It is autumn and new semester in school is beginning. She is checking timetables of her courses and comparing them to her work calendar to prevent work and studies from overlapping too much. Alice notices that some changes need to be done to her work calendar but she needs to catch the next bus to work or she will be late from an important meeting. To continue the synchronisation of the timetables, Alice transfers her SSO sessions from the home computer to her pocket PC. This way, Alice can continue her session on the bus without having to re-authenticate herself and setting everything up again from the start.

Eve is in the same bus as Alice. She notices that Alice is using her calendar on the mobile device. Eve wants to know if Alice has a date planned with Bob and decides to capture Alice's SSO authentication session to access her calendar. Eve has her own mobile device with her. She alters her homepage URL to look similar to university's calendar applications URL. She sends this altered URL to Alice with the schools instant messaging system and tells her to check a funny picture she has drawn. Alice checks the picture and at the

same time a weakness in her browser makes it think the modified URL is the calendar application of the university and sends the SSO session cookies to it. Though to Eve's surprise, the SSO system can differentiate between her replay attack cookies and Alice's legally transfered cookies and she cannot pretend to be Alice.

After the bus ride Alice arrives at her workplace and transfers the SSO sessions from her pocket PC to her work computer. She quickly checks the system of the workplace for new messages about the meeting without having to re-authenticate because she still has her original session from home open and with her. Alice notices nothing new and goes to her meeting.

1.2 Problem statement

As the use case scenario states, one user can have multiple sessions on different SSO systems simultaneously. The sessions are tied to the devices they were started on. If the user changes to a new device, he has to re-authenticate himself to the SSO systems. This can be tedious especially on mobile devices.

Migrating the authentication sessions of the SSO systems with the user removes the need for re-authentication. The authentication is meant to identify the user and not the device that is used. The identity information given to the service has the user's information. The identifier that tells which authentication session belongs to the user is stored in the client-side of the SSO system. Thus, the migration is possible to do with minimum changes to the server-side. The client-side transfer allows the migration of different kinds of SSO systems simultaneously without actions in the server-side.

To be able to migrate the authentication session, we must find out how the SSO systems store the authentication session in the client machine, how this session can be distinguished from the other sessions stored in the device, how this session information can be extracted from the original device and imported to the target device, and how the information can be transferred between the devices participating in the migration. The devices and information passing between them are assumed to be secure in this study.

1.3 Organisation of the thesis

This thesis consists of eight chapters: Introduction, SSO, Environment, Design, Implementation, Evaluation, Discussion, and Conclusions. SSO chapter

tells about the different approaches to the single sign-on designs. Also, some example SSO systems are described. Environment describes the central technologies concerning the SSO systems and the transfer methods needed for the authentication session migration.

Design chapter outlines the plans for the implementation of the authentication session migration application prototype done in this thesis. It also contains criteria for the evaluation of the implementation. Implementation chapter has the description of the implementation of the authentication migration prototype. The end of the chapter includes also the findings of the experiments done with the prototype.

The evaluation against the criteria presented in the Design chapter is in the Evaluation chapter which also tells about the problems encountered during the implementation and our own evaluation on how well the implementation fills our expectations. Discussion chapter discusses a variety of topics that surfaced during the research and implementation phases of this thesis. Conclusions chapter presents the final conclusion of this thesis.

Chapter 2

Single sign-on

Expanding internet offers more and more possibilities for different services to exist. Many of these services require user authentication. Different single sign-on (SSO) systems simplify these authentication processes from the user's point of view. The SSO systems provide access to multiple services with one login, and they offer benefits from both the point of view of usability and security.

- **Usability**: Only one login is needed for several services.
- **Usability**: Organisations can use authentication services outside their own systems when a trust relationship exists. The organisations that trust each other form federations.
- **Usability/security**: User has fewer user names and passwords to remember. This diminishes the chance that user writes the user names and passwords down.
- **Security**: People tend to use the same password in many services and when this password is compromised changes have to be done in all services. SSO needs the password change only once for all the services.
- **Security**: SSO helps to implement and keep the security policy of the company in effect [24].

The SSO systems are usually be divided into two interacting parts: identity provider (IdP), and service provider (SP). IdP handles the user authentication and grants an access ticket. The access ticket contains the identity

information of the authenticated user. The SP uses this information to decide what resources in its service belong to the user. If the user has not been granted an access ticket, when he tries to access a protected resource, the SP directs the user to the IdP for authentication. The IdP gives the access ticket to the user to pass to the SP or the IdP gives it directly to the SP.

The SSO systems have evolved from being centralised to decentralised and finally into federated systems. Some implementations have been created to be centralised and later updated to the more modern versions as time has passed. The next sections describe these approaches and give examples of the systems created to implement them. Also, table 2.1 shows a comparison between the three approaches.

SSO approach	**Number of IdPs**	**Placement of IdPs**	**Trust relationships**
Centralised	one	inside own organisation	not needed
Distributed	more than one	inside own organisation	not needed
Federated	more than one	inside and outside own organisation	needed between organisations

Table 2.1: Comparison of the three different SSO system approaches

2.1 Centralised approach

The centralised approach stores the identity information in a central location. The central user identities can be accessed through a single server or multiple servers. The identity information can also be copied to several databases but in this case some method of synchronisation is used to keep the different databases identical. Both the SPs and the centralised IdP are maintained by the same organisation. Thus, the SPs and IdP are trusted at all times.

2.1.1 Kerberos

Kerberos is one of the oldest SSO system implementations. It is defined in Internet Engineering Task Forces RFC 4120 [33]. RFC 4120 calls Kerberos

a network authentication system but it can be regarded as a single sign-on system since it provides access into several resources with only one login. The original Kerberos is a centralised SSO system as it uses a central server to give the tickets to users but it can be extended to work as a federated SSO system [18].

Kerberos is a token-based authentication system. It consists of three different parts: an authentication server, a ticket granting server, and the SPs that provide the services the users want to access. The authentication server (AS) and the ticket granting server (TGS) can be thought to form a single IdP. Kerberos uses symmetric cryptography to encrypt the tokens. The AS and the different SPs share a secret that is used to encrypt and decrypt the tokens.

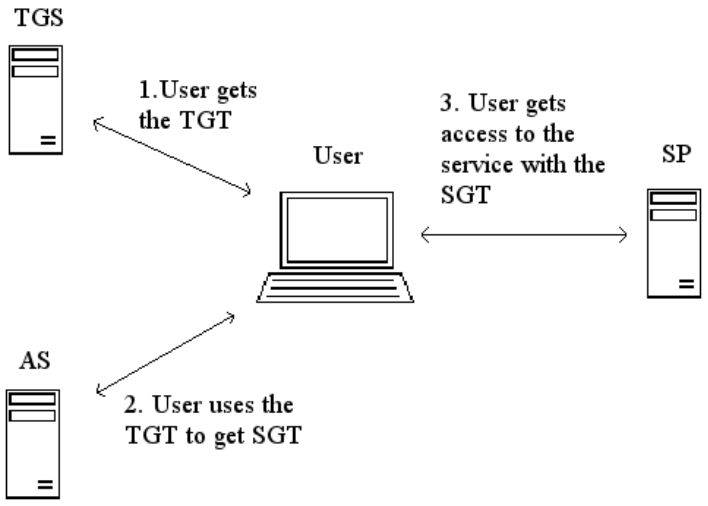

Figure 2.1: Kerberos SSO system data flow

Figure 2.1 shows how the access tickets are passed around in the Kerberos system. First, the user has to authenticate himself with the TGS to get a ticket granting ticket (TGT). If the user has a TGT, AS gives user a service granting ticket (SGT) with the users identity credentials. The SGT is used to authenticate the user with all the SPs. If the user already has a valid service granting ticket from the TGS, he does not have to re-authenticate himself. This makes Kerberos a SSO system. Both TGS and SGT contain information about the user's network address. This prevents a replay attack from a different network address with the captured tickets.

2.2 Distributed approach

The distributed system has more than one identity provider (IdP). The different IdPs have different user identity databases. The user has to have his identity authenticated only on one of the IdPs to gain access to the services the SPs provide. User usually has the choice which IdP to use for the authentication. All the IdP servers are under the control of the same organisation. Therefore, the servers are regarded as trusted in all cases.

2.2.1 OpenID

OpenID [5] is designed as a federated SSO system. It is mostly used by commercial web services. The commercial services have been slow to adapt trust relationships with each other. Thus, many of the them use the OpenID system only within their own organisation. In this case, OpenID can be regarded as an example of a distributed system because it has no trust relationships with IdPs outside its own organisation.

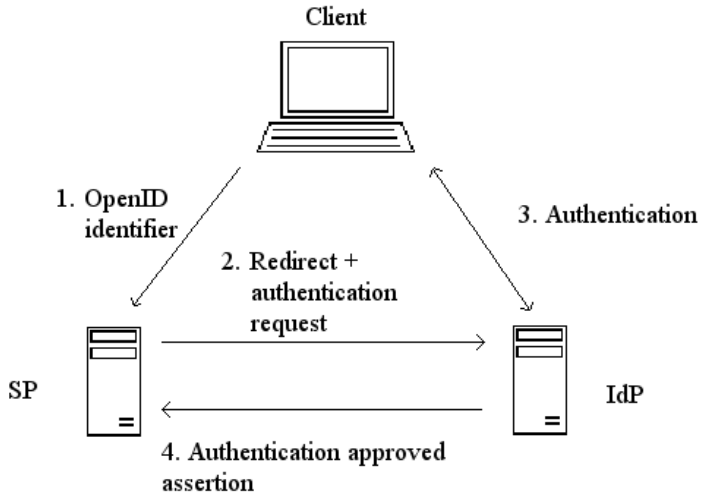

Figure 2.2: OpenID SSO system data flow

Figure 2.2 shows the data flow of an authentication in OpenID. OpenID user has an OpenID identifier which he uses for the login. With the identifier, the SP finds out which IdP to contact for the authentication. The SP then

redirects the user to the chosen IdP with an authentication request. IdP checks if the user is allowed to be authenticated in the server. The server authenticates the user and redirects the user back to the SP with an authentication approved assertion or authentication failed message. The approved assertion contains the user identity information. The SP checks the validity of the identity information. If the information is valid, the SP gives the user access to the resource.

OpenID allows flexibility for the SPs. OpenID does not force the SPs to store the authentication session identifier on the client-side separate from the other web session information. Therefore, it is possible that the authentication session migration cannot be done on an OpenID SP without transferring also other session information.

The OpenID IdPs have no default name for the authentication session cookie. Newer version usually use a combination of name of the IdP and openid tags, for example, "_exampleidp_openid". The OpenID IdP authentication session has no protection against replay attacks that use captured session cookies. The SP side may have protection depending on the implementation of the SP side. Livejournal, for example, offers the users option to force the server to accept the Livejournal session cookies only from the current IP address of the user.

2.3 Federated approach

Federated systems have more than one authentication server. These servers are situated in several organisations. The organisations accepting authentications from outside their own systems must have trust relationships. The organisations cannot accept identities provided by untrusted IdPs. The choice of which of the trusted IdP servers to use is usually given to the user. The IdP and the SP decide what information about the user's identity is passed to the application requiring the authentication. IdP chooses authentication method depending on how strong authentication the accessed resource demands.

The organisations form the trust relationships with business contracts which define how the identities and identity providers are used between the organisations. A circle of trust emerges between the organisations. One organisation can trust the trusted partners of the other organisation after it forms a trust relationship with it. [47]

2.3.1 Shibboleth

Shibboleth [8] is a open source federated single sign-on system developed by the Internet2 Middleware Initiative. Shibboleth uses security assertion markup language (SAML) to pass authentication and identity information between the IdP and SP servers. Figure 2.3 shows an example of a Shibboleth authentication.

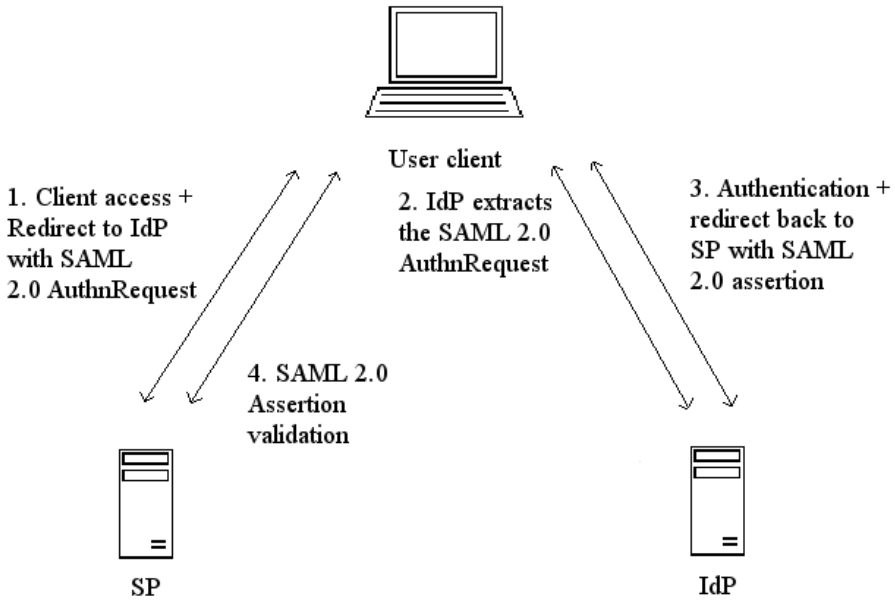

Figure 2.3: Example of a Shibboleths SSO authentication procedure

1. User accesses protected resource on a web server. Shibboleth's SP notices this and asks with a list which server user wants to use for authentication. Before redirecting the user to the IdP, SP stores a SAML 2.0 AuthnRequest in the browser.

2. When the client contacts the chosen IdP, the IdP extracts the authentication request from the browser. With the authentication request, the IdP knows how to authenticate the user and redirects the user to a login page. If the user already has an authentication session with the IdP, login is not needed.

3. User authenticates himself to the IdP by giving a user name and password for the SSO service. After the user has been authenticated, the IdP collects information it has in store about the user. This information is filtered to make sure that only relevant information is revealed to the service. The information is modified to fit in a SAML 2.0 assertion. This assertion is encrypted, signed, and stored in the client. The user is redirected back to the SP with the assertion.

4. When the client contacts the SP again, the SP extracts the assertion from the client. The SP performs the needed security checks to know the validity of the assertion. The SP extracts the information from the assertion and passes the information to the protected service in an understandable form. After this, the service can use the identity information given by the Shibboleth to grant the user access to the service.

Shibboleth uses cookies for session management. When client accesses the Shibboleth system, it first sends the cookies that belong to the system. These cookies contain the session information corresponding to the user's session stored in the Shibboleth system. Transferring these cookies transfers the session.

SP side of Shibboleth has protection against replay attacks that use captured session cookies. Two attributes control the protection by checking the user's IP address: CheckAddress, and ConsistentAddress [27]. By default, these are set as true in version 2.0 of Shibboleth.

CheckAddress makes sure that the client connects to the SP and the IdP from the same IP address. Purpose of CheckAddress is to prevent attackers from capturing the connection during redirects but the attribute is recommended to be set as false because solutions like NAT might make the client use different IP addresses on different networks. It is possible with Shibboleth that the SP and the IdP are in different networks.

Client has a separate authentication session with both the SP and the IdP. ConsistentAddress protects only the sessions with SPs. ConsistentAddress checks that the client sends the cookies to the SP always from the same IP address. If the IP changes, the SP rejects the session cookies and redirects the client back to the IdP for re-authentication. ConsistentAddress must be changed to false for the session migration to work properly. If the attribute is set as true when the authentication session is migrated, the SP rejects the connection from the new IP and redirects user to the IdP. After the redirect, the IdP simply re-authenticates the user automatically with the transferred

IdP session information and creates a new SP authentication session for the user.

2.3.2 Identifying cookies of Shibboleth

Shibboleth does the session handling with cookies. It uses both session cookies that expire when the user agent exits, and persistent cookies that expire when a specified time comes. Figure 2.4 shows the cookies Shibboleth uses. Both the IdP and SP side have their own cookies for each session. The IdP and SP issue these when the user is authenticated. User does not have to re-authenticate himself while he has the session cookies. A third cookie is issued when the SP decides which IdP the user wants to access for the authentication. This cookie is used when re-authentication to the SP is needed. From the cookie, the SP knows straight what IdP to use for the user authentication.

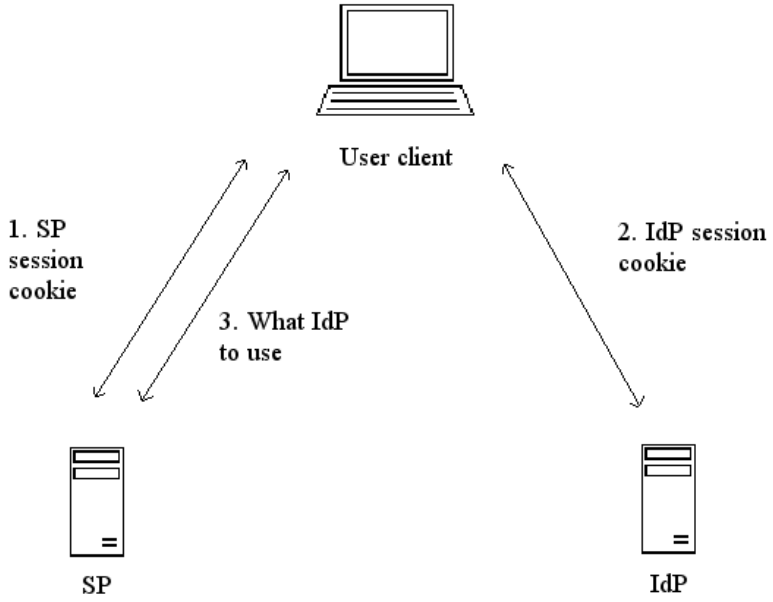

Figure 2.4: Cookies Shibboleth uses for session handling

SP cookie is a session cookie. Normally the name of it starts with a tag "_shibsession_". Application code is appended to the end of this tag. The name can be changed with a attribute in the SP configuration but the default name is usually used.

IdP cookie is a session cookie. The naming of the IdP cookie varies. The default name is "_idp_session" but it can be changed from the system attributes. The name of the cookie depends also on the authentication method used. Shibboleth may use outside methods for the authentication and in that case the session authentication cookies come from that authentication application.

The third cookie Shibboleth system has is the cookie containing information about the IdP that the SP should contact for the authentication. If the user has already chosen an IdP server to use, the SP automatically redirects the user to this IdP. This cookie is usually a persistent cookie and naming of it varies.

2.4 Other categorisations for SSO systems

Other categorisations for the SSO systems exist. These categorisations usually include systems that cannot be regarded as true SSO systems. Thus, they are wider than the categorisation introduced in the earlier sections of this chapter. This section introduces three of these other categorisations.

In 1995, T.A. Parker outlined two ways of doing SSO systems [36]. These ways were scripting and approach using authentication and access tickets. Scripting stores the user name and password in the user's machine and provides them to the web services when required. This takes away the need to type in login information from the user but it does not really offer any other benefits of the SSO systems. The scripting still logs on the user on every service separately. The access ticket approach is described in the beginning of this chapter.

SSO systems can also be categorised depending on how they transfer and store user credentials [18]. This categorisation divides the systems into five categories varying on what they are based on: token, PKI, credential synchronisation, secure client-side credential caching, and secure server-side credential caching.

Token-based systems store encrypted credentials in tokens after the user has been authenticated. These tokens are stored in the client machine which gives them to the SPs the user wants to access. A token-based system is similar to a PKI-based. PKI-based systems use public key infrastructure to encrypt and validate the tokens.

Credential synchronisation has a separate database of user credentials for

all different authorities. The credential databases are kept identical with synchronisation. This architecture is not a true a SSO system because it only keeps the same credentials on different systems. User still has to authenticate himself on different systems separately.

Secure client-side credential caching unlocks the credentials from a primary authentication authority and stores them securely in the system of the client. These credentials are then used to authenticate the user on further logins to other services. Secure server-side credential caching stores the credentials in a central database. The secondary authentication authorities retrieve the credentials from the central database.

Third way to categorise is actually a taxonomy. The SSO systems are divided based on where the identity provider (IdP) is placed [37]. These are divided into local and proxy solutions. Also, division to pseudo-SSOs and true SSOs is made.

Local pseudo-SSO stores the user names and passwords in the client, and gives them automatically to the service requiring authentication. This is the scripting approach mentioned earlier. Proxy-based pseudo-SSO stores the user information for authentication in a proxy. The user must trust the proxy to store this data. At the beginning of the session user authenticates himself to the proxy. The proxy does all the further authentications to the services.

Local true SSO has a component in the client machine that authenticates the user. The service trusts this component to give the right information. Also, the integrity of the component must be ensured. Proxy-based true SSO has an external server that handles the user authentication and information passing. A trust relationship exists between the SP and the external server. The SSO systems described in earlier sections all fall into this category.

Chapter 3

Environment

This chapter looks into the technologies and protocols single sign-on (SSO) systems use to handle user sessions. The technologies needed for the session migration are also discussed.

SSO systems use several protocols and technologies. HTTP state management protocol uses cookies for state management. The HTTP connections are otherwise stateless. The SSO systems use cookies for session handling. Different SSO systems use the cookies in different ways. Also, different architectural approaches have been proposed for the SSO systems based on the use of cookies.

Cookies are handled differently on different client-side platforms. Some operating systems leave cookies to the applications while some operating systems handle the HTTP connections and cookies for the applications. Different browsers also have their own ways for handling the cookie storage. There are also differences between browsers in the desktop environment and in the mobile environment.

Also concerning the authentication session migration, some things must be discussed. A migrated session from a new platform will look like a replay attack on the server-side. Also, the session migration needs a way to pass the session information between the participating devices. The first section tells about session mobility and the concepts it contains.

3.1 Session Mobility

Raffaele Bolla et al. [11] define four kinds of mobility: personal, service, terminal and session mobilities. With personal mobility, the user is not fixed to one position with the device but can move around. Service mobility allows the user to access services in a consistent way with the same personal settings and without changes in the quality of service (QoS) from different devices. Terminal mobility assures that the mobile device retains connection to the service when the device changes from one network to another. Session mobility, also known as session migration, moves the session with the service with the user when the user changes devices. All these four sides of mobility are necessary for an ubiquitous computing environment.

Services usually create the sessions with the device that the user is using. Session mobility ties these sessions to the user and not the device. The user is free to change from one device to another depending on his needs at any given time and the session follows with him. This adds to usability because user does not have to start everything from the beginning when he has to change devices.

The problem in moving the session from device to device with the user is also called user-level handoff. This is in contrast to the term host-level handoff used for terminal mobility. Cui et al. [17] show three problems that have to be considered when implementing session mobility:

1. Session information has to be captured, stored, and forwarded to the target device of the migration in a seamless manner. The session has to continue where it was left off.

2. The target device of the migration is usually very different from the original device. The session has to be modified to fit the target device and the personal settings the user has on it.

3. The migration has to be optimised to work fast enough when compared to starting the session from beginning on the target device.

Session mobility has been researched in internet browsing [46][26][10] and multimedia sessions [12][31]. Session mobility allows the user to browse for a long time on one device and change to a more suitable device if his situation changes. Multimedia sessions are especially concentrated on migrating a video session. Also, medical environment takes advantage of session mobility [20].

3.2 Cookies

The Internet cookie is defined in RFC 2965 of Internet Engineering Task Force [29]. Cookies are used by the web servers to store information in the client-side user agents for state management of the HTTP connection. With the cookie state management, the server gets information about the client straight from the client itself and does not have to keep a separate list about the information and which client the information belongs to.

Cookies are used for a number of things. They are commonly used by web sites as shopping carts for the items that a customer wants to buy from an Internet shop, they can be used for session handling when login is required, or they are used to track the users behaviour on the internet.

Web server first sends a cookie with a Set-cookie response header when the user agent requests a web page. User agent stores this cookie and sends it back to the web server when the user requests a web page with a corresponding domain and path again with the user agent.

There are two kind of cookies: session cookies that do not have an expiry date and are discarded when the user agent exits, and persistent cookies that have an expiry date and are discarded when this time comes. Cookies have name, value, and attributes. Value is the information that the web server wants to store in the cookie. Cookie attributes are

- **Domain** of the web server that sent the cookie.
- **Path** is the URL of the page on the server that generated the cookie response.
- **Max-Age** is time that the user agent stores the cookie. After this time, the cookie is discarded. If the max-age attribute has not been set, the cookie is discarded when the user agent shuts down.
- **Secure** tells if a secure connection must be used to send the cookie to the web server. By default, cookies are sent over an insecure channel.
- **Port** tells the port where the cookie is sent. By default, any request-port can be used. The port tells if the cookie is a HTTP cookie or if it used with some other protocol.

3.2.1 Cookies in different operating systems

This section tells how cookies are handled in PC and Mac environments. Browsers in both Linux and Windows operate the same way regarding cookie handling. Mac OS handles the cookies in a different way.

PC environment has several options for browsers. This section compares the three biggest browsers, namely Internet Explorer (IE), Mozilla Firefox, and Opera, on how they handle and store cookies. All these browsers store the persistent cookies in the file system and the session cookies in memory. There is no common way on how they store the persistent cookies. The browsers store the persistent cookies in different places and in different formats. Table 3.1 shows a comparison of cookie handling between these three major web browsers in PC environment.

Web browser	**Accessing cookies in memory**	**File for cookies**	**Storing format**
Internet Explorer	not possible	in separate files	text
Mozilla Firefox	user side scripting	cookies.sqlite	sqlite
Opera	manually	cookie4.dat	Opera's own format

Table 3.1: Comparison of cookie handling on different browsers

Internet Explorer (IE) offers some options for extendibility to the user. These options mostly include possibilities to manipulate the graphical user interface (GUI). IE has no options for the manipulation of the stored cookies. The persistent cookies can be accessed in the file system. IE stores the cookies in two places. Cookies are first stored in temp files and later transferred to the browsers cookie directory. All cookies of IE are stored in separate files which makes identifying individual cookies easy.

Mozilla Firefox has wide extendibility. All projects of Mozilla are open source and offer many possibilities for a developer to change the functionality of the browsers. With extensions, the browsers outlook and many other features can be controlled. These features include the possibility to fully control the cookies the browser has in store. The possibility to control the cookies makes Firefox the best option in the PC environment for extracting and importing the cookies.

Opera offers a possibility for the user to manipulate the browsers stored cookies with an editor built into the browser. With the editor, user can add and manipulate both the persistent and session cookies. Opera does not offer a possibility to extract the session cookies from memory with a user-side script. The persistent cookies can be accessed from the cookie4.dat file they are stored in. Thus, to extract the session cookies the user would have to first change them to persistent cookies with editor and then retrieve the cookies from the cookie storing file.

Cookie handling in Mac OS is different from the cookie handling in the PC environment. The Mac OS offers a package for the HTTP connections. Thus, the operating system also handles the cookies. This is an integrated approach to cookie handling and offers an opportunity for all applications to store the cookies in a centralised location. With the Mac OS HTTP package, the cookies can be added and manipulated freely. Thus the cookie extraction is possible for all browsers and applications on the Mac OS. The software developers kit (SDK) that offers access to the HTTP package is not available for free.

3.2.2 Browsers and cookies on mobile platforms

A wide variety of browsers exists for mobile devices. Different devices have different operating systems, and correspondingly, different web browsers. A browser that would work on all or even most mobile operating systems does not exist. In this section, we go through browsers MicroB, Fennec, IPhone's Safari, Mobile browser for S60, and Opera Mini as examples of mobile browsers. Other browsers for mobile platforms exist, for example, Opera Mobile, IEMobile, and NetFront. These browsers are not included in this research because they offer no new angle to the cookie handling or other operations from the desktop versions or other mobile browsers discussed here.

Mobile devices have less resources available than desktop computers. Thus, the browsers must be more lightweight. The mobile browsers usually have the same inner workings with the desktop browsers but the main difference is in graphical user interfaces (GUI) and on how the GUIs are implemented. For example, all Mozilla-based browsers have the same component libraries for cookie handling, but the placement of extension buttons and menus are done differently on different browsers. Mozilla-based browsers include Firefox, MicroB and Fennec.

MicroB is a browser made by Nokia for the Maemo operating system that is used in Nokia internet tablets. This browser is compatible with the Mozilla

browsers in other aspects than graphics. MicroB uses GIMP Toolkit (GTK+) used in the Maemo for graphics instead of the Mozilla's XUL XML user interface markup language that is used in other Mozilla browsers. GTK+ is a more lightweight solution and is thus better for a platform with limited resources. If a Mozilla browser extension uses graphics, it has to be ported to work in the MicroB. Otherwise, it should work fine without changes.

Mozilla community is developing a new browser for mobile platforms called Fennec. At the moment, a beta version for N800 and N810, and an alpha version for window mobile exist, and Mozilla is also developing a version for Symbian OS. Fennec will have the same libraries and XPCOM components the normal Mozilla browser has, but the GUI is different. Fennec uses XUL for graphics just as Firefox but the GUI of Fennec is more lightweight. The Mozilla community does not restrict the modification of the GUI but offers a set of guidelines how the extensions should and should not modify the GUI. Most Firefox extensions should work on Fennec. Only changes to follow the Mozilla community guidelines are needed.

The mobile version of Mac's Safari browser used in IPhone is very similar to the desktop version: operating system handles HTTP and cookies. Symbian operating system that is used in many smart phones has a browser similar to the mobile Safari called Browser for S60. Browser for S60 and Safari use the same Webkit engine for the GUI. Thus, Safari widgets made to change the GUI should also work in Browser for S60 though the original GUI is different on the browser. Symbian OS also handles the HTTP connections and cookies for the Browser for S60.

Opera Mini has a different approach on making a lightweight browser. Opera Mini does browsing through a proxy that compresses and preprocesses the web pages to better fit in the mobile environment. With the compression, download times of web pages are faster because mobile devices usually have a limited bandwidth. The Opera Mini browser does not have all the functionality of a traditional browser because the preprocessing only gives the browser what it should display and not the original HTML code. The Opera Mini proxy keeps track of the web server cookies and handles the cookie transactions.

3.3 Cookies on single sign-on systems

Cookies in general are used for state management of HTTP connections. SSO systems use cookies for session management to see if the user has al-

ready authenticated himself with the SSO system. Several Web-based SSO systems use cookies for session handling [40]. SSO systems can also use other approaches. Higgins selector-based identity framework does not use cookies [2]. It uses information cards and tokens. Also, other SSO systems using tokens or PKI exist. These architectures need some client side infrastructure or more administration than a cookie-based system. Using cookies is usually the fastest approach [16].

Vipin Samar proposes three approaches for cookie-based SSO system in his paper Single Sign-On Using Cookies for Web Applications [40]. These approaches are a centralised cookie server, decentralised cookie server, and centralised cookie login server approach. These are described next in more detail.

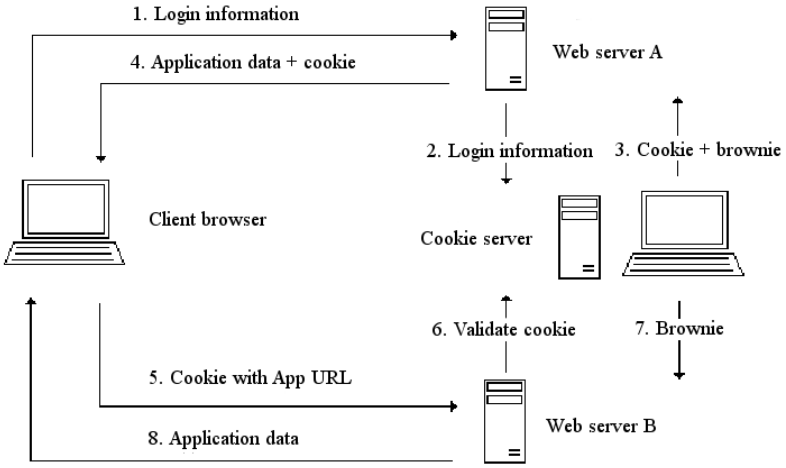

Figure 3.1: Control flow of centralised cookie server approach

Brownie is a type of cookie that servers use for communication between each other. Centralised cookie server approach uses brownies to pass information between the cookie server and the web server user wants to access. Normal cookies store data in the client browser. Web servers of the centralised cookie server approach do the user authentication individually [40] or the cookie server does the authentication [16]. Figure 3.1 shows how data flows in the centralised cookie server approach:

1. Client browser starts the session with the web server A and sends its login information to the server.

2. Web server A passes clients login information to cookie server for validation.

3. Cookie server validates user authentication and sends back a cookie for identifying that the client now has a session in the cookie server and a brownie containing the user's identity information.

4. Web server A gives the cookie from the cookie server to the client. Now the client browser can authenticate its further sessions to web servers with this cookie.

5. When the client wants to use an application in the web server B, it sends the cookie server cookie first to the server B. The web server B is in the same SSO domain as server A.

6. Web browser B validates the cookie with the cookie server.

7. Cookie server again gives the user information in a brownie but passing the cookie back is not needed since the client already presented it.

8. Web server B gives client access to the application that it requested.

The decentralised cookie server approach has no cookie server. Web servers authenticate the users and create the cookies themselves. Server stores more information in the cookie given to the client. This information includes user name, user IP address, web server name, and cookie expiry time. The server digitally signs the cookie information. Decentralised cookie server approach lessens the amount of management needed. However, it makes it harder to change the information structure of the SSO system because it has to be done on all web servers using the system. Also, the size of cookies is larger and the SSO cookies cannot be used for state management.

The third approach, i.e. centralised cookie login server, does authentication and cookie creation in one place. When client browser tries to login without the SSO cookie, web server redirects the connection to the login server. Login server does the authentication and issues a cookie that it sends to the web server with further requests. Login server re-directs the client server back to the original web server, and the server grants the client its own session cookie. Figure 3.2 shows a control flow model of this approach. SAML-based SSO systems use the centralised cookie login server approach.

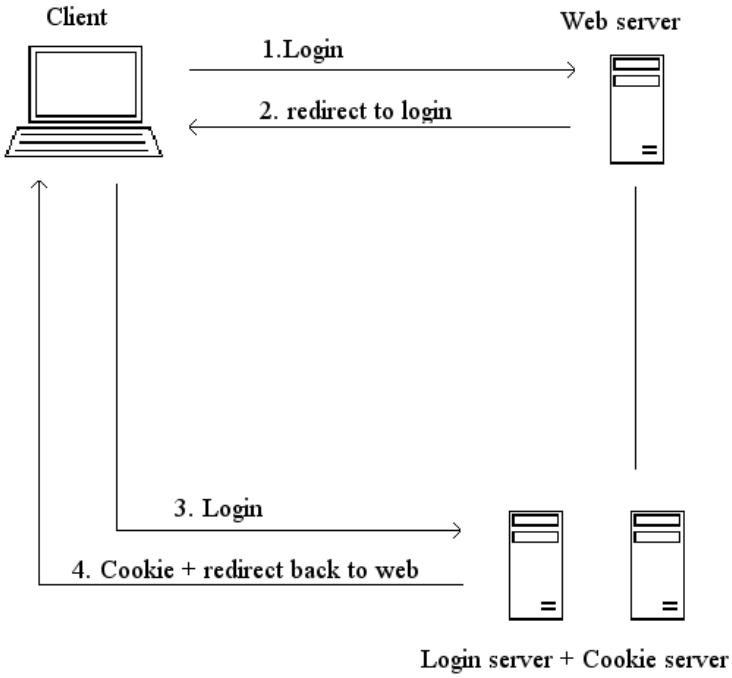

Figure 3.2: Control flow of centralised login server approach

3.4 Cookies and security

A number of things affect the cookies that are used in SSO systems. Park and Sandhu [35] identify three types of threats when using cookies on the web. First, cookies are normally in clear text in the web. This makes them vulnerable to snooping. Second, cookies are stored in web browsers in clear text and can be accessed by the user. The user can thus alter the cookie data and pose as a different user. Third vulnerability is the way the client chooses where to send the received cookies. Attacker can make a URL impersonating the original web site, and make the client browser send it the cookies instead of the original site. This is called the cookie-harvesting threat. Harvested cookies can be used to perform a replay attack.

Snooping threat is easy to protect against. SSL protects the connection when sending cookies. Still, this does not help to protect against the other threats [35]. Encrypting the cookie data prevents the attacker from seeing the cookie information but it also may open a vulnerability. If an attacker gets the session key used to encrypt the data, he can do serious harm to the system. This is why the cookies should not contain any user information

preferably only the session identifier in the SSO system. [16]

The cookie harvesting vulnerability causes the web browser to send cookies to the wrong recipient because browsers identify the servers only from the URL and if the attacker can impersonate the server with the URL the browser thinks it is the original server. Attacker captures the cookies with this vulnerability and uses them on replay attacks. Cookie handling in general needs a bigger change to correct this problem but protection against it can be achieved. Three methods exist for the protection: checking that the cookies come from the same IP address that they were sent, storing users password as a hash in the cookie and asking the user for them on new web sites, or using digital signatures to check the identity of the user every time he connects [35]. Only the first method works in the SSO systems. The other two render the SSO unable to work properly.

Cookies are vulnerable in the client system. Therefore, persistent cookies should not be used. Session cookies will be destroyed when the user exits the session but persistent cookies usually stay on the system longer. Likewise, the session should have an expiry time in the server to prevent an attacker from using the hijacked session forever. [21]

3.4.1 Cookies vs. certificates

One way for authentication in the internet is the use of certificates. RFC3280 of Internet Engineering Task Force [25] defines the web server certificates. In a way, certificates are the opposite of the client authentication with cookies. Web services use the certificates to prove their identity. A certificate contains a signature of a third party that assures that the certificate holder is who he claims to be. A web certificate also includes the web servers public key for SSL/TLS connection creation.

The checking of certificate authenticity in web browsers is left to the user. If a user trusts the certificate issuer, he can accept the certificate to be trusted. This is the same kind of trust relationship that has to exist between a service provider (SP) and identity provider (IdP) in a federated SSO system. The difference is that security policies of an individual users may be more lax in what they accept as trusted than with the SPs.

Web servers supply the certificates at the beginning of a connection to the client much like the client supplies the cookies to the server. Certificates have an expiration date which is longer than the lifetime of a session cookie. Certificates are resigned with the certificate issuer when the expiration date is reached. The update of a certificate on a web browser is done automatically

if the user chooses to trust the certificates from a certain source or the user is asked every time a certificate changes if he still trusts the source.

3.5 Replay attacks

Replay attacks are attacks where the attacker eavesdrops a data exchange, copies parts of the exchange, and modifies and replays these parts to the participants of the exchange or other targets. Replay attacks can be done without decrypting or even understanding the encrypted data. Syverson gives a complete taxonomy of replay attacks in A Taxonomy of Replay Attacks [48]. The taxonomy has two component taxonomies: the origin of the message, and the destination of the message.

Syverson divides the attacks into two categories depending on their origin: run external attacks, and run internal attacks. Attacker uses messages from outside the current run of the protocol in run external attacks. Correspondingly, run internal attack uses messages from inside the current run of the protocol. The run external attacks can be further divided into classic attacks and interleaving attacks. The classic attack uses messages from a previous run of the protocol run. The interleaving attack uses messages from a protocol run at the same time.

Depending on the target of the messages, the attacks are divided into two categories: straight replay, and deflections. Straight replays replay the message to the intended original receiver after a delay. The deflection can be done back to the sender or to a third party.

An attack where attacker captures and replays the authentication session cookies to the SSO server is a classic straight replay attack. The cookies are captured from a previous exchange of the server and the client and then replayed straight back to the server.

3.5.1 Protecting against replay attacks

Several solutions exist that protect against replay attacks. The SSO systems usually use IP address checking against replay attacks with stolen cookies. The server accepts cookies only if they come from the same IP address they were issued to. This limits the cookies to only work from the original device. For the authentication session migration we need to consider other methods to protect against replay attacks to prevent the replay protection from discarding the transferred session authentication cookies.

One way for protection against replay attacks adding information to the messages. The information is checked to ensure that the information has not been used before or if it belongs to a different process. The information is usually something that the application itself does not need for the process to work. [15]

Systems must make sure that the sessions are set to expire at some point. If attacker can get access to the session with a replay attack, he still cannot create a new session. Expiring the sessions prevents the attacker from abusing the hijacked session endlessly. Ye et al. show in Efficient Cookie Revocation for Web Authentication [49] that many web services do not delete the session from the server when user logs out. They only remove the session cookies from the client machine. The session continues in the server until it expires. This lets the attacker exploit the hijacked session even if the user thinks that it has ended.

Protection against modified messages can be done by ensuring the contents integrity. This integrity ensuring can be done by adding hashes of the message content to the message. By checking the hash, it can be seen if the content has been tampered with.

3.6 Transport Layer Security (TLS)

Transport Layer Security (TLS) is a security protocol that creates secure transport layer connections between devices. TLS is the protocol usually used for securing cookie transfers. TLS is defined in RFC5246 of Internet Engineering Task Force [19]. It is based on the older secure sockets layer (SSL) protocol.

TLS uses both asymmetric and symmetric cryptography. Web browsers get and store the public keys of servers in the same certificates that authenticate the servers identity. The asymmetric public key cryptography protects the exchange of the shared secret that the symmetric cryptography uses. The shared secret encrypts the data transfer between the devices. TLS can use many different ciphers for the transactions. These collections of ciphers are called cipher suites, and TLS negotiates in the beginning of a connection which suite will be used. A weak suite means a vulnerable connection.

TLS has a feature called the session resumption that is described in RFC 4507 [39]. The session resumption can resume a previously created TLS session with storing the session information in a ticket in the client-side of the TLS connection. Also, the shared secret that the TLS connection uses is

needed for the session resumption. Thus, a replay attack with a stolen ticket would not resume a connection. The session resumption resumes the TLS connection but it will not resume the data connection the TLS connection was protecting [28].

3.7 Transfer method

Desktop computers and mobile devices have several technologies for data transfer including the SSO session data. The data can be transferred either through wires or wirelessly. The wireless technologies include Bluetooth and WLAN. The transfer does not have to be done straight from one device to another. A third device can be used as a storage for the information. The information can be retrieved from this storage later when needed. The next three sections look into these possible ways of data transfer. The fourth section compares the pros and cons of the solutions.

3.7.1 Bluetooth

Bluetooth is a connection technology where individual machines are connected without cables or plug-ins commonly used to connect mobile phones with computers. Bluetooth is based on a client-server model where client searches for devices in its area and looks at their services. These services include access to internet, and file transfer or sharing. Devices can be paired in Bluetooth to work as trusted devices with each other.

Server adds the services it offers to a service record. This record is given to the client when it searches for servers and their services. If the client finds a service it wants, it connects to the server and requests access to the service and server may or may not allow the client to use the service. Most devices that have Bluetooth enabled can work as both client and server. Devices such as earphones naturally cannot work as servers.

Bluetooth uses Radio frequency communication (RFCOMM) to pass communication between applications on different devices. RFCOMM is created on the top of Logical link control and adaptation protocol (L2CAP) which handles the communication between devices. RFCOMM is also known as the serial port emulation, and it offers the serial port profile (SPP) service in the service record. The sockets SPP creates between machines offer a reliable data stream much like TCP.

Bluetooth uses Object exchange (OBEX) protocol to transfer binary objects.

Infrared Data association originally created the OBEX for data transfer on infra red connection but the Bluetooth Special Interest Group has also adopted it. Thus, OBEX is not an integral part of the Bluetooth, and it is not included in all the Bluetooth libraries even though many of the Bluetooth services use OBEX for file transfer. OBEX uses RFCOMM for the data transfer.

Bluetooth uses encryption to protect the link layer connections. Each Bluetooth device has a unique address. With the unique address, Bluetooth devices are identified individually. Devices can be paired to accept connections and data from a selected unique device. If the devices have not been paired, the device asks the user to accept the connections. The pairing feature adds to usability if the other device is trusted. The paired devices have a shared secret that is used to authenticate the devices to each other. The secret is needed because Bluetooth addresses can be faked [23].

3.7.2 Wireless local area network (WLAN)

WLAN is a wireless option to make a local area network. WLAN connects two or more devices to form a network. Many mobile devices have the option to connect to a WLAN network and WLAN can also be added easily to a desktop computer if it does not already have it.

Computers use WLAN to connect wirelessly to the physical LAN network to offer mobility for laptop users. WLAN connects computers to LAN networks through access points. With the connection to access point, the computers work in the LAN as any other machines. WLAN can also form direct P2P connections without using an access point. These connections are called wireless ad-hoc networks. They are faster to setup and can work more flexibly than the traditional way of connecting in WLAN. Wireless ad-hoc networks are not meant to be permanent solutions for the network but a small group of computers can form fast and simple temporary connections with them.

Older security mechanism of WLAN is called Wired Equivalent Privacy (WEP). It protects the WLAN network and the data transfers in it. WEP has had problems with its security but the IEEE 801.11i version of the WLAN has addressed many of these vulnerabilities. The security mechanism based on most of the features of IEEE 801.11i is called Wi-Fi Protected Access (WPA). Also, poorly configured WLAN network will have security problems. Therefore, depending on the network the data transfer may or may not be secure. [34]

3.7.3 Internet

Internet can also transfer and store a browser session. This is a common and researched way to transfer web browser sessions. Hsieh et al. [26] introduce three approaches for the browser session transfer from platform to platform: client-based, server-based, and proxy-based. Several implementations for these approaches exist. Single implementation can also be a hybrid of two of the approaches [10].

The client-based approach needs a small client side application to be installed. This application tracks or has access to the browsers session information. On transfer, the application sends the gathered session data over a P2P connection to the new platform or stores the data in a server where it can be accessed. Client-based approach allows easy access to the browser data, but the transfer is more complex. Also, the used browsers have to be similar, or importing and exporting of the browser session information can be difficult.

The server-based approach stores the session information on the web servers. This approach is often used in web shops. User can save his situation in the shop, and the shop will restore the user to this session when he logs to the system next time with any browser. This approach will not store the whole session of the user but only the part that is in the web server that offers the session storing service.

The proxy-based approach does surfing on the web through a proxy. Proxy monitors browsing and stores the essential session data. Still proxy lets all the session information to the client machine. For example, cookies are stored both on the proxy and the client browser. When user wants to resume the session, he simply contacts the server and authenticates himself as the user who owns the session. The proxy-based approach allows session to be resumed even if the connection of the original session was broken. [13]

Connections from the client to the proxy are protected with SSL/TLS in the internet solution of browser session transfer. Much of the security of the transfer depend on the proxy. The proxy would have to be trusted to be secure if the user does not have the proxy under his own control.

3.7.4 Comparison of the transfer methods

All the three options, Bluetooth, WLAN, and the internet transfer, are available on most mobile devices and possible to get on all desktop computers. Availability is therefore not a deciding criteria.

Bluetooth and WLAN comparison shows that both offer very similar link layer connections that are created between machines. WLAN has to have additional application for the service discovery on different devices and a protocol to transfer the needed data between machines. Bluetooth offers these services amongst its protocols. Service discovery on devices is part of Bluetooth and smaller amounts of data can be easily transferred with RFCOMM and larger files with the OBEX file transfer.

Bluetooth and the internet transfer have more differences than Bluetooth and WLAN. The client-based approach in internet transfer is very similar to the implementation with Bluetooth. A client side application gathers the needed data and sends it to the new device in both methods. The only difference is on the link layer transfer method used. Bluetooth finds the new device for the session easier than the search through TCP/IP that the internet option does. Bluetooth only needs the devices to be close to the each other and the target device to advertise the session transfer service. TCP/IP needs the address or the name of the new machine to be known thus making the transfer a bit more complicated.

Server-side approach does not work with session transfer [45]. Resuming the saved session on the server would need the user to authenticate himself as the owner of the session. This renders the SSO session transfer pretty much useless because authentication for the saved session does not differ much from re-authentication to the SSO system.

The proxy-based approach for the session transfer offers some benefits. The user would not have to resume the session immediately and would not even have to know the new device before saving the session because the session could be retrieved from storage of the proxy anytime and anywhere. Still, the authentication to the proxy is needed. Surfing the web through a proxy makes browsing probably a little slower on a desktop computer. This defect could be compensated if the proxy offered some compression and preprocessing for the mobile browser similar to the Opera Mini discussed in section 3.2.2, thus combining the functionalities of proxies.

Security-wise Bluetooth is deemed to be more secure than WLAN [23]. In this thesis, the devices used in the transfer are assumed be secure and under the user's control. Bluetooth has only two devices in the transfer: the device with the original session, and the device where the session will be transferred. Internet transfer through a proxy adds a third device to this transfer. Bluetooth is thus a more simple solution from the security point of view than the proxy transfer. Compared to the internet transfer, all parts of the Bluetooth transfer are under the user's control. Parts of the internet transfer are always

more public and have to be trusted.

In this thesis, the control of the cookie data is important. The session migration need only the data of the authentication session. Building a proxy-based session transfer from scratch would be too big project for this thesis and might run into problems that are not inside the scope of the problem. Using a ready made implementation of the proxy would transfer too much of the browser session data and not give a full control on what is transferred. Therefore, client-side Bluetooth transfer is the best option to make the data transfer lightweight and flexible.

Chapter 4

Design

This chapter looks into the design of the implementation made in this thesis. The goal is to design a system for transferring session cookies of a cookie-based SSO from one client platform to another. These transferred cookies are used on the new platform to continue the existing SSO session. We chose to transfer authentication session of a cookie-based SSO system because it can be done without significant changes to the server-side of the system. Token-based SSO system such as Higgins selector-based identity framework needs more changes to the client-side program for the authentication migration to work.

The cookie-based SSO session migration needs three components to work: first the cookies are extracted from the original machine, second the cookie information is transferred from the original machine to the target machine, and third the transferred information is imported to the target machines system in the correct format. The cookie information is altered for the transfer. The importation on the target machine creates new cookies with the transferred information, and places the new cookies in the systems cookie storage. These new cookies are used to continue the SSO session with the target machine.

Figure 4.1 shows an example on how the system transfers the SSO session cookies from a desktop computer to a mobile device. Cookies are extracted and imported with a browser extension in this example.

1. Browser extracts cookies with the browser extension and writes them into a cookie file.

2. Browser starts transfer client, and provides it with the URL the user is on at the moment and the location of the cookie file.

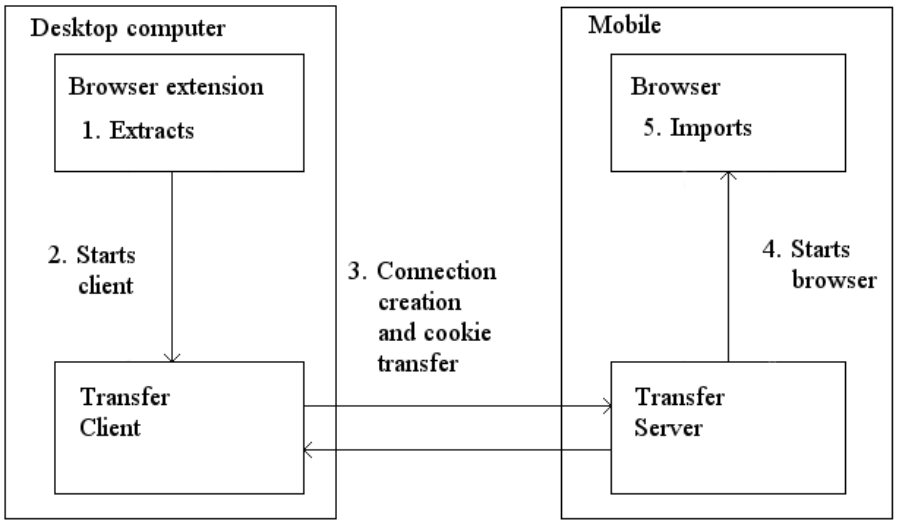

Figure 4.1: Example of a cookie transfer

3. Transfer client contacts a server, and sends the cookie information and the URL. Server writes the cookie information to the cookie file in its own system.

4. Server starts browser with the given URL on the mobile device.

5. Mobile devices browser imports the cookie data from the file before starting the browsers user interface. Now user can continue the SSO session started on the original platform.

The transfer works the same way from the mobile device to the desktop computer. Thus, the transaction applications are equal, similar, and not based on the client-server model. Client and server parts here are differentiated because the client part attempts the connection to a server part, which listens for the connections. The server is started before the client. A file is used to store the extracted cookie data, and to pass the cookie information between the components.

4.1 Cookie extraction and importation

Cookies must first be extracted from the original machine to start the SSO session transfer. Cookies are extracted from the operating systems cookie storage or from the web browsers storage depending on the approach used on the platform. The two approaches operating systems use for the cookie handling are discussed in the section 3.2.1. Both approaches usually offer an interface for the cookie management. Both the operating systems and web browsers cookie management interfaces work in similar manner from the user's point of view.

The cookie extraction applications task is to fetch the wanted cookie information from system and pass it to the data transfer application, which handles the transfer between devices. The application also imports the transferred cookies to the target system.

Cookies in SSO systems are usually session cookies. Thus, the cookie information-extracting application must have access to the cookies the browser has stored in its memory. If browsers manage the cookie handling in the system, the cookie extraction is done with a browser extension to get access to the session cookies. Otherwise, the cookies are handled with a component in the operating system. However, the browser extension is needed to give the user a way to start the session transfer from the browser. Also, the current URL of the browser is extracted with the extension and passed to the transfer application.

4.1.1 Cookie data

The cookie extractor stores the extracted cookie information into a file. The file passes the information from the cookie extraction component to the transfer component. The data transfer between the browser extension and the transfer component needs the cookie file. If the operating system handles the cookies, the cookie file is not needed because the cookie extraction and transfer are done in the same place. Also, the transfer component sends the information to the target platform either as a text string read from the file or the whole file in binary format.

The cookie extractor writes the cookie text file in the same format the cookies are in stored the memory. SSO Session cookies are encrypted, and thus have special characters. Keeping the character encoding same prevents the special characters from changing.

The SSO session transfer does not need to transfer all the information about the session cookies. Section 3.2 discusses about cookies and their names, values and attributes. Four attributes that need to be extracted are name, value, domain, and path. The name and value represent the most essential information of the cookies. These are the values the web server stores in the cookies. The cookie handler on the target device of the transfer also needs to know who to send the cookies to. The owner information of the cookie is in the domain and path values. The section 3.4 discusses how all SSO cookies should only be sent through a secured connection and only session cookies should be used. Thus, we assume the secure attribute of the cookies as set on and max-age attribute is not needed. Last attribute, port, is rarely used and its transfer is not necessary.

```
_shibsession_3cad23363448f312010095cf28faa32b291d5129
_024484db1a482356d3e9080f925f761b
wwwlogin.tkk.fi
/
_shibstate_3cad23363448f312010095cf28faa32b291d5129
https%3A%2F%2Fwwwlogin.tkk.fi%2F%3Fbroker_for%3DLnRray5maS8%3D%26redirect_to
wwwlogin.tkk.fi
/
```

Figure 4.2: Example of the cookie file contents

Figure 4.2 shows an example of how the cookie data is stored in the cookie file. The cookie file has two cookies. Each cookie is four lines long. First line is the name, second value, third domain, and fourth and final line the path. All lines have only the data the browser gives as the values. This way, the data can be easily imported back to the browser environment. The cookie information file starts with an empty line.

4.2 Data transfer

The data transfer transports the cookie data from the original platform to the target platform. The data transfer applications parts are divided into a client and server. Client part transfers the cookie information to the server. Server listens and waits for the clients connections and stores the received data.

The cookie data is transported between platforms as a data stream containing a string or as a binary file with a file transfer protocol. The amount of the cookie data is relatively small because we only want to transfer the SSO

session cookies. The cookie transfer is a copy. Thus, after the transaction the application keeps the cookie data in the memory of the original device but removes it from the cookie files. This prevents the session cookies from being left in storage in the file system when they are deleted in the browser at the end of a session.

Data transfer client reads the cookie information from a file pointed to it by the cookie extractor, establishes a connection with the data transfer server, and transfers the read information to the server with the current URL of the browser. The user starts the client with a menu option or button in the browser. The client closes after the transaction with the server has been done. Before closing, the client erases the cookie file.

The data transfer server starts the transfer service and waits for a connection. When connection is established, the server receives the cookie information and stores it to a file. The information includes the URL for the web browser. The data transfer server starts the web browser of the device with the URL. User must start the server when he wants to transfer the SSO session, and the server will close after the transaction with the client is carried out. This prevents further transfers from tampering with the user's current session.

4.3 Evaluation criteria

The following criteria is used to evaluate whether the design and implementation carried out in this thesis are successful. The testing of the implemented SSO session transfer system has been done with existing SSO system implementations. The list offers explanations as to why certain criteria was chosen and what is good to remember on the implementation stage.

1. *The transferred SSO session should continue to work on the target platform as it has worked on the original platform. In other words, the transferred SSO session should stay the same and no new SSO session should be created on the basis of the transferred information.*

   The web server often creates its own cookies for the actual access to the services of the web server using the identity given by the SSO. The SSO system may use this information to create a new SSO session if the original SSO session does not exist anymore.

2. *Transferred cookies should be identical enough to work on the new platform.*

All the needed values and attributes are sent to the target platform. Also, the values should stay the same from platform to platform. This may pose a problem if different encodings are used on different devices.

3. *The transfer should only affect the client side of the SSO session. No changes or only minimum changes should be made on the SP or IdP sides of the SSO system.*

   We want to be able to transfer multiple SSO sessions simultaneously. If this is possible with changes only on the client-side for all the SSO systems, the session transfer is easy to implement and to use.

4. *The transfer should be faster and require less input from the user than logging out and re-authenticating himself on the target platform the conventional way.*

   The transfer would be near useless if re-authentication would be faster. However, typing the user credentials on a mobile device is often harder than on a desktop computer. The only place where user input might be needed is in the choosing of the new platform if more than one Bluetooth device offering the same service exists in the region.

5. *User should be able to continue his browser session from the same location on the target platform. The whole internet session does not need to be migrated, only the SSO authentication sessions.*

   Transferring the whole internet session would be out of the scope of this thesis. Transferring the current page though should be easy and offers a substantial usability advantage when the user does not have to browse back to the page on the target platform.

This criteria fills the fundamental aspects of the authentication session prototype to be implemented. If some other points arise during the implementation, they are added when the implementation is evaluated. The evaluation of the implementation against this criteria is in section 6.1.

Chapter 5

Implementation

When we started to do this thesis, we did not know if it was possible to migrate an authentication session to a different platform on the client-side of the system without help from the server-side or external proxy. The goal was to find out if the migration was possible, and if not, what things needed changes for the migration to work.

First we examined how the authentication systems store their session information. The session information is stored on the clients with cookies of the HTTP state management protocol. At the same time with the cookie research, we found out how to transfer the cookie information between platforms. From the start, Bluetooth seemed to be the most natural way to transfer data between a desktop machine and a mobile device.

At the start, we did not have knowledge on how the cookies are stored or handled by the browsers and operating systems. It quickly came clear that Firefox was the only browser that had the possibility to extract and manipulate cookies easily. Therefore, Firefox was chosen as the browser for the PC environment part of the authentication session migration prototype. We also found a new browser for mobile devices, Fennec, which is developed by the same Mozilla community that is responsible for Firefox.

We needed an existing SSO implementation to test the authentication session migration on. We studied if it was possible to use Shibboleth for this purpose. Shibboleth is in common use in our university. Later, popular SSO system OpenID was also used in the testing.

This chapter looks into the cookie extraction and manipulation, the transfer of the cookie information between platforms, and the importation of the cookie data on the target platform. The last sections look into the exper-

iments done with the implemented authentication session migration prototype. The authentication session migration prototype was implemented for devices presented in the first section.

5.1 Devices

This thesis has implementation of the authentication session migration prototype for two devices: a desktop computer, and a Nokia E90 Communicator. Python is the programming language for Bluetooth part of the implementation. Python is available on both devices but the devices have different versions. This is not a problem but it is good to remember the different version numbers.

The desktop is a everyday computer that has Microsoft Windows XP operating system (OS). Thus, the computer has wide selection of different applications and libraries. The Nokia N810 Internet Tablet is a mobile device. It uses Maemo OS that has the beta version of the new Mozilla-based Fennec browser available. Mozilla-based browsers on both the original and target device makes the exporting and importing of the session information easier. Table 5.1 lists the platforms of the devices, their operating systems, browsers, and Python versions used in the implementation.

Device	**Operating system**	**Browser**	**Python version**
Desktop PC	Microsoft Windows XP	Mozilla Firefox 3.0.13	Python 2.6
Nokia N810 Internet Tablet	Maemo	Fennec beta	Python 2.5

Table 5.1: The devices chosen for the implementation

5.1.1 Cookie extraction

The authentication session migration prototype needs to extract the session cookies of the SSO system. The cookies are then transferred to the target device for the session migration. Both selected browsers, Firefox and Fennec, are Mozilla-based and handle cookies the same way. Thus, browser extensions extract the session cookies on both devices.

Coding language for Mozilla extensions is Javascript. Mozilla browsers use Cross Platform Component Object Model (XPCOM) components to add custom functionality to the browser. Through these XPCOM components, Mozilla browser extensions can access and use cross-platform libraries.

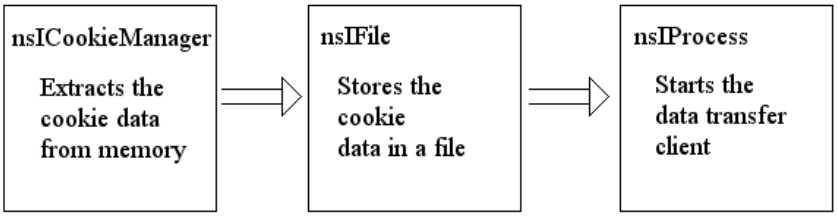

Figure 5.1: XPCOM components of the extension implementation

Figure 5.1 shows the XPCOM interfaces this implementation uses in the order they are roughly called in the code. First nsICookieManager [42] extracts the cookie data from the memory of the browser. NsIFile [32] stores the cookie data in the right format into the cookie file. Last, the nsIProcess [44] interface starts the data transfer client that transfers the cookie file data to the target platform. The extension imports the cookies the other way around. First the cookie data is read from the file with the nsIFile interface and then the extension adds the cookies to the memory of the browser with the nsICookieManager interface. The cookie importation executes when the browser is started. Next we look into these interfaces and processes in more detail.

XPCOM interface nsICookieManager manipulates the cookies in this extension: it retrieves, adds, and removes cookies. Retrieved cookies are in the form of nsICookie [41] XPCOM component. NsICookie uses UTF-8 encoding. The nsICookie component has nine attributes. The extension for nsICookie, nsICookie2, gives additional six attributes. All of these attributes are read only. If user wants to modify a cookie, he creates a new cookie with the modified attributes and destroys the old cookie. The following attributes are extracted from the nsICookies for the transfer:

- **Name** contains the name of the cookie that the server uses to identify it.
- **Value** is the actual information the server stores in the cookie.

- **Host** tells the URL of the web server that owns the cookie.
- **Path** has the path in the web server for the service that owns the cookie.

The next attributes are assumed to have a certain value or are not needed in the migrated cookies:

- **IsDomain** tells if the cookie should be sent to all web pages in the domain. A dot in the beginning of the host attribute tells that the cookie is a domain cookie. Thus, the host attribute already tells if the cookie is a domain cookie and the attribute does not have to be transferred.
- **IsSecure** sets the cookie to be sent only through protected connections. SSO session cookie should only be sent through secure connections and thus we can assume this attribute to be always true.
- **Expires** tells the browser when the persistent cookie should be removed from the storage as seconds since Jan 1, 1970. If the cookie is a session cookie, this attribute is not needed. The Migrated SSO cookies are all session cookies and this attribute is not needed.
- **Status** has the P3P status of the cookie. P3P is a protocol that allows the web sites to tell the intended use of the information they gather with the cookie. Authentication session cookies do not use this attribute.
- **Policy** is also connected to the P3P protocol and is not needed in authentication session cookies.
- **CreationTime** contains the time the cookie was created. The creation time of the migrated cookies is the time they are imported to the browser.
- **Expiry** has the actual time the cookie expires.
- **IsHttpOnly** is set if the cookie is a HTTP only cookie and should only be sent through HTTP connections. The port attribute tells the cookie is HTTP only if it set as 80.
- **IsSession** tells if the cookie is a session cookie. All the authentication cookies are session cookies. Thus, this attribute can be set as true on the target device and does not need to be transferred.

- **LastAccessed** is the time the cookie was last accessed.
- **RawHost** has the host name of the cookie without the domain dot if the cookie is a domain cookie.

First version of nsICookieManager offers two methods: remove and removeAll. RemoveAll removes all cookies from the browser and remove removes only individual cookies. Remove needs three attributes to identify the cookie: domain (host), name and path. NsICookieManager also offers an enumerator attribute that contains the cookies as a nsICookie components described earlier. The SSO authentication migration extension uses the enumerator to go through the cookies and chooses the SSO cookies for the transfer. The newer NsICookieManager2 extension gives additional methods for the cookie manipulation:

- **Add** adds a new cookie directly to memory of the browser. The authentication migration extension uses the add method to import the cookies on the target device. The add method is not the recommended way to create new cookies to the browser [43]. The recommended nsICookieService adds cookies when a page is loaded. Thus, it is not suitable for this implementation, and the implementation uses add method of the nsICookieManager to add the cookies directly to memory of the browser.
- **CookieExists** checks if a cookie exists. The check needs a complete nsICookie component as an attribute.
- **CountCookiesFromHost** counts the cookies corresponding to a given host.
- **GetCookiesFromHost** returns the cookie corresponding to a given host. Knowing all the hosts that have given a SSO session cookie during the users browsing would be hard. It is easier go through all cookies and decide from the cookie values if it is a SSO session cookie. Identifying the SSO session cookies is not easy and is discussed further in the section 7.2. The getCookiesFromHost returns an enumerator containing the cookies as nsICookie components.
- **ImportCookies** imports all cookies from the cookie files the browser uses. Browsers can use this method to transfer all the persistent cookies from a different browser. Thus, the authentication migration extension has no use for this method.

Add method of the nsICookieManager has eight attributes. The extracted values, name, value, host and path, are the first attributes in the add method. The host is called domain in the nsICookieManager. The last four attributes are set as follows:

- **Secure** assures that the cookie is only sent through a secure connection to the server. As the SSO cookies should only be sent through secure connection, this is always set as True.
- **IsHTTPOnly** tells if the cookie is only sent through HTTP connections. This setting is not normally used and can be set as False.
- **IsSession** tells if the cookie is a session cookie and not a persistent cookie. SSO session cookies should always be session cookies. Cookie importation sets the isSession attribute as True.
- **Expiry** tells the time the cookie is deleted. This attribute has no meaning if isSession is set as true because the session cookies are deleted when the session ends. Only session cookies need to be imported.

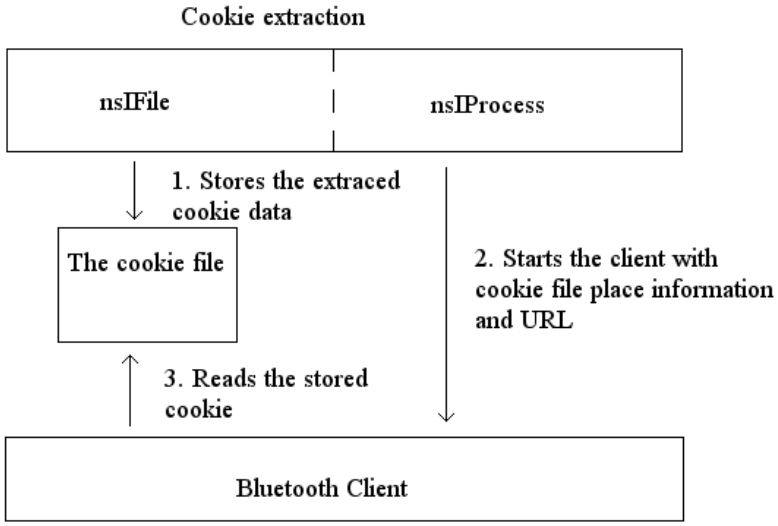

Figure 5.2: Interaction of the cookie extraction extension with the Bluetooth client

Figure 5.2 shows how the nsIFile and nsIProcess components interact with the Bluetooth client. NsIFile interface stores the cookie information to be

transferred in a file in the root directory of the web browser extension. NsICookie uses UTF-8 encoding. To ensure that the cookie information stays the same, it is made sure that the data stays in UTF-8 coding when written to a file. NsIFile offers a normal variety of functions for file manipulation. It is the standard way of doing the file I/O in Mozilla extensions.

NsIProcess starts the Python application for the Bluetooth transfer between platforms: the nsIProcess starts the Python interpreter with the Bluetooth client file, current URL of the browser, and the position of the cookie file in the file system as arguments. The Bluetooth client file contains the code to be interpreted and other arguments are passed to this interpreted program. The cookie file is located in the root directory of the extension.

There is one difference between the Firefox and Fennec versions: Fennec does not have drop down menus. The method to start the cookie transfer is therefore a graphical button in the tool bar. Fennec documentation does not recommend this method, but at the moment, no other solution is offered. A better place to put buttons for applications will probably be introduced later by the Mozilla community. The authentication migration extension starts on the Firefox from a menu item in the tools menu.

5.2 Cookie transfer

Based on the discussion in the section 3.7.4, we chose Bluetooth as the technology for the cookie information transfer. Bluetooth works in a client-server model and offers suitable protocols for this implementation. Task of Bluetooth is to pair the devices and transfer the cookie information between the them. Process in Bluetooth consists of three steps: device discovery, service discovery, and connection creation. After these steps, the connection passes data between the connected devices.

Both devices, the PC and the Internet Tablet, have the client and server applications. The authentication session migration works thus both ways. There are no differences between the client applications on either device. The server applications differ on how they start the web browsers of the devices.

We chose Python as the programming language for creating and using the Bluetooth connection. Python needs an external library for the Bluetooth operations. The library we chose is called PyBluez [7]. It is based on the C library Bluez. PyBluez has a version for both the Microsoft Windows XP on PC and the Maemo OS on the N810 Internet Tablet. This allows us to use

the same code on both devices.

PyBluez does not have support for OBEX that is the file transfer protocol included in Bluetooth. It was first planned to be used in this implementation. However, the amount of cookie information should stay fairly small, and small amounts of text can also be transferred easily with the RFCOMM serial port profile (SPP). If a OBEX file transfer is needed, libraries such as PyOBEX exist.

RFCOMM connection in PyBluez works with virtual sockets. A socket is created on both ends of the connection. The socket connection passes data both ways. User writes data into the socket and the recipient on the other end reads the data from his socket. PyBluez appends the new data in the socket to the end of the earlier data. The socket removes the data when it is read. Thus, the simplest solution is to send all the data with one transaction.

5.2.1 Bluetooth client

Figure 5.3 shows a possible communication sequence between the Bluetooth client and server parts. The right side shows how the client acts in the cookie transfer transaction: the browser starts the client that begins with the device and service discoveries. PyBluez does these both with the find_services function.

Correct Bluetooth services are discovered based on an Universally Unique Identifier (UUID) that are unique for each application. With UUID, the client can be sure that it contacts the right service. The client always asks the user to choose the right device from a list of the discovered devices with the SSO authentication migration service. When only one service is discovered, only one option is on the list. If no devices with the service are discovered, the dialog informs the user.

A graphical user interface (GUI) dialog gives the options for choosing the right device for the transfer to the user. The dialog is done with GTK+. It is easier to find and install libraries for windows than for N810. Thus, we chose the native graphical library of the N810 GTK+. The actual GUI is very small and uses only the basic elements. The suitability or the complexity of the library was therefore not an important aspect. Figure 5.4 shows a screen capture of the GUI dialog.

After finding the right device, the client attempts to create a connection with that device. If the server accepts the connection and a socket is successfully created, client transfers the cookie information file as a sting and appends

the browsers current URL from parameters given by the Mozilla extension to the transfer socket. The URL is separated from the cookie information with a delimiter. The data ends with an end tag to let the server know when the transfer is complete. After the transaction is finished, the client closes.

5.2.2 Bluetooth server

The left side of the figure 5.3 shows the actions of the server in the Bluetooth cookie transfer. Server starts the SPP service and waits for a connection. When client connects, the server accepts the connection and waits for the client to transfer the cookie and URL information string. If the string ends with the designated end tag, the server stops listening for more information and starts handling the string. First it takes the delimiter, end tag, and the URL from the end of the string. URL is stored in a variable. Remaining string is the cookie data that is written to the cookie file.

Last job for the server application is to start a web browser with the same URL that was open on the original device. This URL was given with the transferred cookie information string. The windows version of the server application starts the web browser with the Python webbrowser library. Fennec has no support in the webbrowser library as it is a new browser. Fennec starts with the subprocess command in Maemo. Subprocess starts Fennec by executing a shell script that has the line that starts Fennec.

5.3 Experiments

The authentication session migration prototype was tested on two systems: OpenID [5], and Shibboleth [8]. OpenID, and Shibboleth are described in sections 2.2.1 and 2.3.1 correspondingly. The authentication session was also successfully migrated on Central Authentication Service [1] SSO system deployed in a project of the university. The test show that the authentication session migration prototype migrates the authentications sessions successfully. Testing also shows how the SSO systems work with the migrated session. The testing checks if the sessions can be continued from both devices after the migration and if terminating the session from one device also prevents the authentication session from working on the other device.

5.3.1 OpenID

The authentication session migration prototype was tested with OpenID. We chose Livejournal [4] as the service provider (SP) and ClaimId service as the OpenID identity provider (IdP). Livejournal is one of the OpenID SPs that accepts identity information from outside its own organisation. ClaimId as the outside identity provider offers a clear distinction between the IdP and the SP.

The session identifier of the ClaimId is in cookie named "_claimid_openid". Migrating this cookie to the target device migrates the authentication session of the IdP. Thus, we were able to migrate the authentication session of a OpenID IdP to a target device. The session worked on the target device as it worked on the original device.

The OpenID SPs have flexibility on how they are implemented. The session migration for the Livejournal SP needed migration of two cookies: ljloggedin, and ljmastersession. Presumably, the ljloggedin cookie contains information about the user's login to the Livejournal. Ljmastersession probably has only session information. We cannot be sure if we migrated also other information than the authentication session but the test shows that the authentication session is migrated with these two cookies. Livejournal offers a possibility for the user to set an attribute that checks that the cookies come every time from the same IP address. Setting this attribute on prevented the session migration from working for the Livejournal SP.

ClaimId does not force the session cookie to be sent through a secure connection with the secure cookie attribute. However, the cookie transactions to the ClaimId server use a secure connection. Livejournal does all the cookie transactions through the normal non-secure HTTP connection. Thus, we imported all the cookies to the target device with the secure attribute as false. This was against our assumption about the SSO systems we did in the section 4.1.1 and is a clear security vulnerability on both the ClaimId IdP and the Livejournal SP.

During the testing we did not remove the session from the original device when we migrated the authentications session to the target device. After the migration, we were able to use the authentication session from both devices with no problems. After logging out from Livejournal on one device, the authentication session did not work from either device. The IdP, ClaimId, did not remove the session from the server if the session was terminated on one of the devices.

5.3.2 Shibboleth

The authentication session migration prototype was tested on Shibboleth deployed at our university. The testing was done on one IdP and three different SPs. All the SPs had the consistentAddress attribute that rejects the cookies from wrong IP addresses as true and checkAddress set as false. This is the normal setup for a Shibboleth SSO system.

The consistentAddress check prevented the migrated authentication sessions of the SPs from working from the target device. The cookie that has the authentication session information starts with "_shibsession_". The end of the cookie is the application ID of the SP. Migrating this cookie to the target device migrates the session.

The session cookie of the IdP of the SSO system of the university is called "pubcookie_s_SSO". The authentication on the Shibboleth's IdP depends on the authentication method the accessed resource wants the IdP to use. The Shibboleth system of the university uses Pubcookie [6] for the authentication on the IdP. The login cookie from the Pubcookie contains the authentication session information of the IdP.

Migrating the authentications session cookie of the IdP enabled the authentication to the different SPs without having to re-authenticate. The SSO system asks the user on every authentication if the identity information of the IdP can be passed to the SP but this function can be turned off. If the function is turned off, the user cannot see the difference between a migrated SP authentication session and a new SP authentication session that is done with the migrated IdP session. Although, it is not the same thing to create a new SP authentication session as migrating the session from the original device.

The Shibboleth system used in the testing had a possibility to logout from both the SP and IdP. Testing the SP logout functionality was not possible because the migrated SP session did not work. The logout functionality of the IdP was tested. After the authentication session migration, the logout from the IdP ended only the sessions from one device. The other copy of the session worked without a problem. Also, other SP sessions were not ended with the IdP logout.

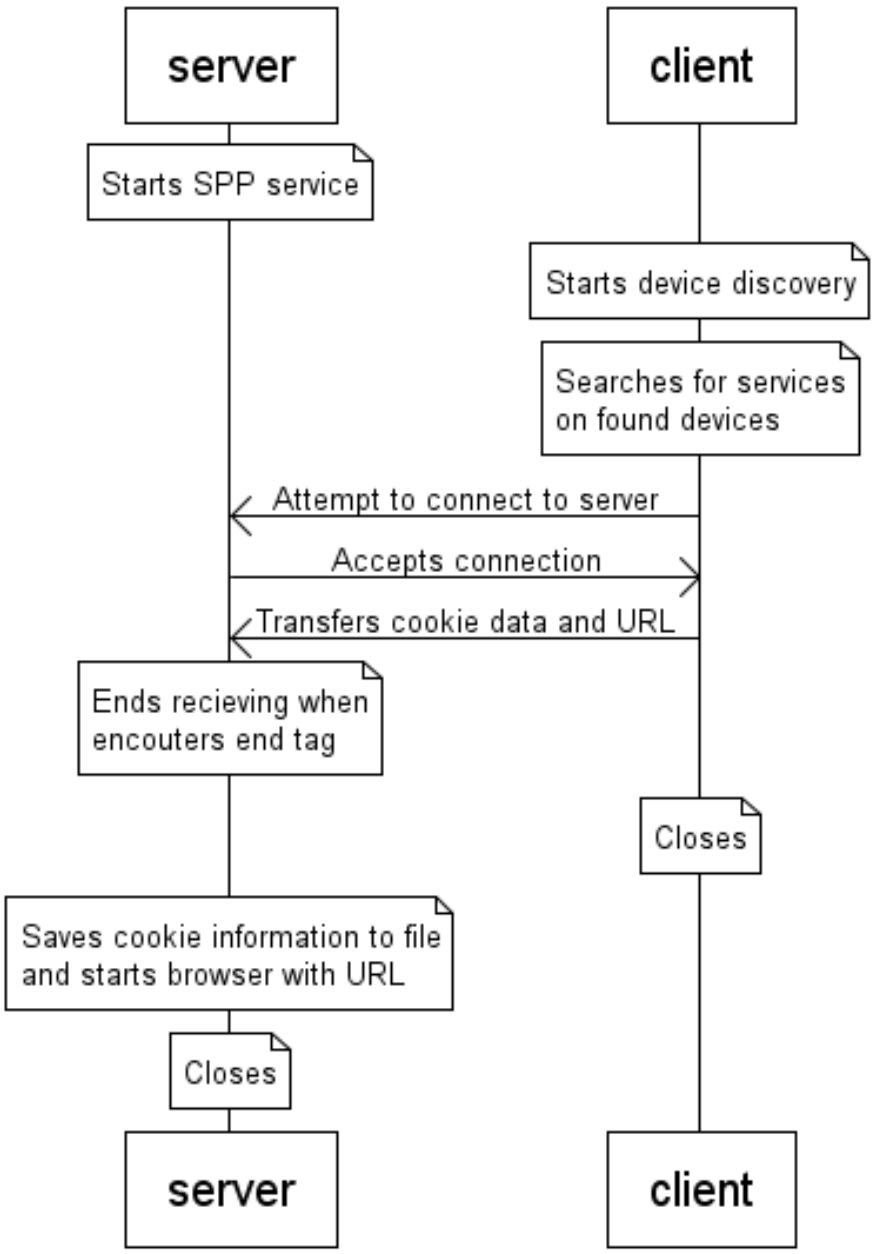

Figure 5.3: Bluetooth client and server interaction

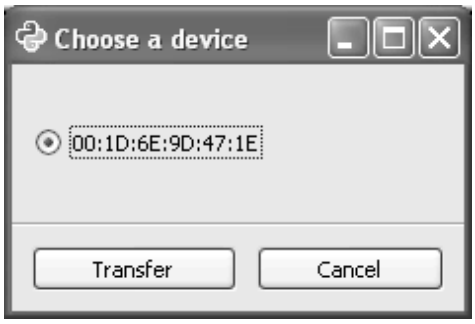

Figure 5.4: The migration target choice dialog

Chapter 6

Evaluation

This chapter discusses how well the created implementation of the prototype of the authentication session migration application answers the problems we are dealing with in this thesis. The implementation is evaluated against criteria stated earlier. The chapter also has discussion about the problems that were encountered during the implementation. The last section tells how well in our opinion the implementation fills our expectations from the beginning of this project.

6.1 Evaluation against the criteria

This section looks into how well the implementation fills the criteria given in the section 4.3. First is the same criteria that was discussed before and then is the evaluation how well the implementation fills the criteria.

1. *The transferred SSO session should continue to work on the target platform as it has worked on the original platform. In other words, the transferred SSO session should stay the same and no new SSO session should be created on the basis of the transferred information.*

   The experiments showed that the same SSO session continues on the Shibboleth SSO system with the transferred cookies. The SPs replay attack protection that checks the IP the cookies come from prevents the migrated authentication session from working. In these cases, the migrated IdP session created new SP sessions.

2. *Transferred cookies should be identical enough to work on the target platform.*

In section 4.1.1 we decided not to transfer all the values and attributes of the cookies. Some attributes are assumed to be with certain values. These assumptions of the attributes of the cookies were right in most of the experimented systems. Only difference was discovered in the testing with the Livejournal SP of OpenID. Livejournal sends the SP authentications session through unprotected connections. The OpenID tests had to be done with the isSecure cookie attribute as false.

3. *The transfer should only affect the client side of the SSO session. No changes or only minimum changes should be made on the SP or IdP sides of the SSO system.*

   In general, the authentication session migration was possible without changes to the server-side of the SSO system. The authentication cookies were enough for the session migration and no other information needed to be transferred. Migrating cookies does not need changes to the server-side on normal SSO systems.

   The session transfer on SPs of the Shibboleth systems did not work because it has protection against replay attacks that send the authentication session cookies from a different IP address. This protection needs to be altered for the authentication session migration to work for the SP. The migrated IdP session created a new SP session automatically. The OpenID SSO system used in the testing also had the same protection but it had to be turned on separately by the user. Section 7.4 discusses about how the servers can distinguish replay attack from a legally migrated session.

4. *The transfer should be faster and require less input from the user than logging out and re-authenticating himself on the target platform the conventional way.*

   Otherwise, the migration is faster than re-authentication on the mobile device, but if many Bluetooth devices are in range the device and service discovery of PyBluez may take a long time. The tests showed that the PyBluez device and service discovery does the discovery at least four times on windows machines. This is probably to make sure that all services are discovered but can take a lot of time. The Maemo version of PyBluez did the discovery only once. Searching for only services on paired devices circumvents this problem.

   The target platform choosing was done with a GUI which offers the possible devices that have the authentication session migration prototype service. The GUI offers a fast way for the user to choose the target

device and prevents the session migration from transferring the session information to a wrong recipient.

5. *User should be able to continue his browser session from the same location on the target platform. The whole internet session does not need to be migrated, only the SSO authentication sessions.*

   Most browsers can be started with a command line command or a shell script with a starting URL as a parameter. This was however not possible in Fennec, but because only beta version was used the functionality will perhaps be added to the final release version.

After the implementation and tests one more criteria came up. First is the criteria, then the explanation of the criteria, and last how well the implementation fills the criteria.

1. *The transfer should only move the authentication session information from one device to another.*

   The aim is to migrate authentication sessions of different kinds of SSO systems. All SSO systems do not use cookies for the session handling. If a group of SSO authentication sessions are migrated, it is not efficient to transfer other kinds of useless information with the migration and may it may even be a security problem.

   The filling of this criteria depends on the SSO systems practices on what they store in the cookies. All cookie-based SSO systems do not store the authentication session information to individual cookies with no other information. The authentication session information cannot be extracted from the cookies because the value of the cookies is encrypted. If the SSO system stores the authentication session information in separate cookie, this criteria is filled.

6.2 Implementation problems

During the implementation of the SSO authentication session migration prototype, some problems were encountered. The first choice as the programming language, Java, was not suitable for the mobile device N810 internet tablet. Later we decided to find out if it is possible to implement the authentication session migration prototype to Nokia E90 Communicator that uses Symbian operating system but the interface to the cookie manager component that handles the cookies is not available for free. Also, the Fennec

beta version does not have all the same functionalities as the Firefox. Next sections discuss more about these problems.

6.2.1 Programming language

The first choice was to use Java as the programming language for the Bluetooth cookie transfer application. Java is a common language for web applications and it has compatibility with most platforms. Though, it was not possible to use Java: Java micro edition (JME) used usually on mobile environments is the only Java edition that has native support for Bluetooth. Java standard edition (JSE) used in the desktop computer must use external library for Bluetooth. N810 internet tablets operating system Maemo has support for neither JME or JSE. It has its own project for Java called Jalimo [3]. Jalimo does not have a working Bluetooth library that would also work in JSE. Thus, both platforms would use own libraries for the Bluetooth implementation and the implementations would have to be different.

Python offers a better choice as the programming language with the PyBluez library that handles Bluetooth. The change to Python did not cause any major problems for the project. The Bluetooth transfer application is fairly small and simple. Thus, no prior knowledge about Python was needed. The change from Java to Python only wasted a days work.

6.2.2 Symbian

During the implementation, we decided to find out if it was possible to implement the authentication session migration prototype on Nokia E90 Communicator that uses the Symbian operating system. Symbian offers a HTTP connection package for the applications and thus handles the cookies for them as mentioned in section 3.2.1. This is a different approach from the browser handling the HTTP connections and cookies.

Symbian controls the cookies with a cookie manager component that offers an interface to operate the cookies. It is similar to the way the browsers handle cookies. The cookie manager interface is not included in the software developers kit (SDK) that is available free for Symbian developers. Thus, we were not able to implement the authentication session migration prototype this way.

The free SDK of Symbian has the possibility to extract the cookies from the header collection of the HTTP session. The default web browser of E90

Communicator is called the Web Browser for S60. Extracting the HTTP session from the Web Browser for S60 would give access to the cookies of the session but the extraction is not possible in the browser. Thus, the cookies extraction for the authentication session migration prototype is not possible and this part of the prototype cannot be implemented on Symbian with these tools.

The Symbian operating system has no support for PyBluez that is used on the other devices for the implementation of the authentication session migration prototype. Symbian has its own library for the Bluetooth operations. Authentication session migration prototype needs its own code for the cookie information transfer on Symbian. Thus, we did not implement any parts of the prototype on Nokia E90 Communicator.

6.2.3 Fennec

The Bluetooth server of the authentication session migration protocol starts the web browser of the target device with the URL of the original devices web browser. The beta version of Fennec does not have the functionality to start the browser with the URL. The Mozilla community will probably correct this problem to the final release version of Fennec because their goal is to make Fennec work the same way as Firefox. The user has to browse to the right page himself in the Fennec when using the authentication session migration prototype.

6.3 Self evaluation

The beginning of the chapter 5 explains what was our expectations when we started to do this thesis. We had no knowledge if the authentication session migration was possible or how much changes would have to be made to the SSO system to make it work. We thought that a self made dummy SSO system would have to be made. The goal of the implementation was to prove that SSO authentication sessions can be migrated to a different device.

The authentication session migration prototype transfers the authentication session to a target device and the session continues without problems on the new device. The prototype has been used to migrate sessions on existing SSO systems successfully as described in section 5.3. Thus, compared to the expectations the prototype is a success.

Bluetooth is a good method for the cookie information transfer. It is simple

to set up and use. All the devices in the implementation had Bluetooth available easily. On the mobile devices, it was native and on desktop PC a Bluetooth dongle provided the service. PyBluez is easy to use but a very simple library for the Bluetooth protocols. On a bigger project, it is better to use some other more complex library that gives more control to the developer.

The shortcoming of the prototype is that it works only on very specific platforms. The prototype migrates only sessions from Firefox browser of PC and from the Fennec browser of N810 Internet Tablet. The Mozilla-based browsers are the only ones that gives access to the cookie handling to all developers. The implementation to Symbian was not possible because we only had access to the free SDK that does not contain the cookie manager interface. The same situation was with the other browsers of PC. Further work based on the prototype is needed to make it work on wider amount of devices or a different approach such as use of a proxy can be taken. The use of a proxy to migrate authentication sessions is discussed in the section 7.1.

Chapter 7

Discussion

The authentication session migration prototype was implemented on two devices: a desktop computer, and an internet tablet. It is important to discuss how the prototypes findings can be spread to a wider amount of devices and platforms. The cookie extraction of the implementation was done with a web browser extension. Other means to extract the cookie information exist that do not have to be separately implemented on all different browsers. These methods are discussed in section 7.1.

Another discussion point is how the authentication session migration can be made to work on all the different cookie-based SSO systems. The authentication session cookies need to be identified before they can be migrated to the target device. This is discussed in section 7.2.

The federated SSO systems have different means to identify which IdP the user wants to use for authentication. Some systems store this information in a cookie. Thus, to migrate the whole SSO sessions functionality this cookie must also be identified and transferred. The identification of this cookie is discussed in section 7.3.

Some SSO systems protect the authentication sessions by checking that the cookies always come from the same IP address. This prevents the migrated session from working. The SSO systems need a alternative solution for the replay protection. The other solutions that exist for replay protection are discussed in section 7.4.

The whole internet session stored in the browser can be migrated with the authentication session migration. The section 7.5 tells what other data need to be migrated with the authentication session information to transfer the whole session of the internet browser. The section 7.6 shows what conse-

quences arise when users and applications can access the cookies storages on the computers. The last section 7.7 tells what has to be considered if a application is developed based on the prototype implemented in this thesis.

7.1 Session transfer solutions

In general, user has three ways to migrate the authentication session: with a client-side application, browsing through a proxy on the client's machine or by browsing through a proxy on a third-party machine. The client-side application is usually a browser extension. The application handles the authentication session extraction from the browser, the transportation of the information, and the importation of the session data on the target platform. Every different type of browser needs its own individual implementation of the application.

The proxy on the client machine gives the session migration service to all the browsers operating on the machine. Thus, the different browsers do not need a different client application implementations. On the other hand, all the different operating systems need their own implementations. The migration itself between devices is similar to the client-side application. The proxy handles the cookie extraction and importation of the session. Some operating systems, such as Symbian and Mac OS, handle the HTTP connections for the applications as discussed in the section 3.2.1. This approach offers the same services and advantages as the proxy on the client machine for authentication session migration.

Browsing through a third party proxy approach is independent from the browsers and operating systems. The user browses through a proxy that is on a third machine. The proxy monitors and stores the session information. When the user changes device, the proxy adds the old session information to the new HTTP connection. As the proxy is not under the user's control, it has to be trusted.

The implementation itself is easiest in the client-side application approach because the components for cookie manipulation and HTTP connection handling already exist. The problem with the cookie management components is that all browsers do not give free access to them. Also, implementation for different browsers has to be done individually.

The client-side proxy gives the implementation to all the applications working on the device. If the devices operating system handles the cookies, the approach is as easy to implement as the client-side application approach. If

the operating system does not offer a cookie management component, the client-side proxy has to be implemented the same way as the third-party proxy.

The third-party proxy works for all devices that use the HTTP protocol. Thus, it is the most comprehensive approach of the three and needs only one implementation for all devices. It is also the most complex to implement. The proxy has to separate and store the cookies from HTTP headers. The session stored in a proxy can be resumed from any device and at any time after the original session has ended. The third-party proxy needs some sort of authentication from the user to identify the session that belongs to him.

Implementations using these approaches exists for the web browser session migration. Song et al. [46] describe a client-side implementation of a internet session migration application. The application takes a snapshot of the internet session with a browser extension and stores it in a proxy. The target device of the migration retrieves the session snapshot from the proxy. Although the extraction of the session is client-based it still uses a proxy for the transfer. The session extraction in the client is similar with the implementation in this thesis but the use of proxy adds complexity to the session transfer. Also, taking the session snapshot requires more changes to the browser than extracting just the cookies.

Adeyeye et al. [10] propose a client-side implementation that transfers the session to the target device using session initiation protocol (SIP) [38]. The transfer with SIP is done through a proxy or straight to the target device. The proxy is used to provide web browser registration and session data encryption. The SIP protocol is imported as a Mozilla Framework XPCOMM component. Thus, the implementation works only on Mozilla-based browsers just like the implementation in this thesis. Importing the protocol stack to other browsers is a more substantial task than just gaining access to the session cookies.

Canfora et al. [14] describe an implementation of proxy-based browser session migration. The proxy stores and re-uses the user's credentials used in the web application authentications. Thus, the proxy also works as a proxy-based pseudo SSO. Gaining access to the session stored in the proxy requires authentication. Hsieh et al. [26] have also implemented proxy-based implementation of the browser session migration. This implementation also includes a small client-side application that can be used to extract the session information that cannot be monitored with a proxy.

All the implementations that require an authentication to the proxy used in the transfer are not suitable for the authentication session migration because the authentication could as well be done in the SSO system. Also, all the

implementation use internet for the session information transfer. This is more vulnerable than the Bluetooth transfer because all the parts of the transfer are not under the user's control. SIP has the same vulnerabilities as a HTTP transfer [22]. Bluetooth restricts the area of the possible attacks.

7.2 Identifying the session cookies

Name and domain attributes distinguish the session cookies from other HTTP cookies. The session cookies cannot be identified by their values because the values usually contain only the encrypted session identifier. Newer versions of Shibboleth and OpenID use a default name on the session cookies that can be used to identify the session cookies. The SSO servers do not force the use of the default names. If a conflict arises in the network with the cookie names, it is avoidable with a name change. Thus, the default name is not a definite way to identify the session cookies.

The domain attribute contains the URL of the host of the cookie. If the application knows the SSO systems servers, the domain name can be used to distinguish the session cookies. Some Shibboleth IdP servers contain the IdP letters in the address of the server. These IdP servers are thus identifiable with these letters. The domain name identification identifies all the cookies of the server and thus other cookies that are not authentication session cookies are also included.

A combination of name and domain attributes identifies most of the session cookies but it is not a sure way. The identification can distinguish cookies that are not authentication session cookies or leave authentication session cookies out of the identification. The session cookie identification needs a coherent naming policy for the session cookies. Without a coherent policy, the cookies cannot be distinguished from the other HTTP cookies.

7.3 Identifying the IdP the user wants to use

Different cookie-based SSO systems SPs in different domains have no means by themselves to know which IdP the user wants to use even if the user has already authenticated himself to the SSO system in the IdP. The SSO system can have a separate service to direct the user to the right IdP depending on the users choice. This service stores the chosen IdP to a cookie for further redirects to the IdP. The session migration application needs also to transfer

this cookie to migrate all the SSO systems functionality. If the service is not in use, the user has to provide the right IdP on every authentication to different SPs.

Identifying the cookies that tell what IdP to use is as hard or harder as identifying the session cookies. The IdP redirect cookies are persistent and do not have to be transferred through secure connections.

7.4 Distinguishing transferred session from a replay attack

Browsers have cookie handling vulnerability discussed in section 3.4 called the cookie harvesting vulnerability that gives cookies to the wrong recipient. The SSO session cookies can be harvested and used in a replay attack with this vulnerability. Thus, the SSO servers need a protection method against cookie use in replay attacks. If no protection is implemented, the authentication session in SSO system can be captured. The usual way to protect the cookie is to check that the cookies always come from the IP they were issued to. This prevents the authentication session migration from working. Some other way to protect the cookies has to be used.

First method to allow the cookies from different IP address but protect against a replay attack is to record the allowed IP addresses of individual users in a list in the SSO system. The SSO checks that the cookies come from a safe IP address if the IP address of the user changes during a session. This method needs additional management with the IP address lists. The IP address of the user is not always the same on the same device. Thus in some cases, a wide range of IP addresses would have to be accepted. Attacks from within the range are possible.

Second method is to tie the cookies to the SSL session protecting the connection. A Secure Cookie Protocol introduces a protocol to attach the cookies to the specific SSL session [30]. If the SSL session is migrated with the SSO authentication session migration, the cookies can be tied to the SSL session and the attacker would not be able to use the harvested cookies from a different SSL connection.

7.5 Migrating the whole internet session

The migration of only the authentication session is not practical in some cases. The authentication session migration needs only to transfer the session cookies. The whole internet session migration needs all the cookie data and the browsing history. The cookies include both the session and persistent cookies. Persistent cookies are stored in the file system either in one or several files. Browsing history is usually also in a single file. The persistent cookies and browsing history files are easy to import in to the browsers with methods they offer. This way works in the client-side approach of the session transfer implemented in this thesis. If the proxy approach is used the proxy has to monitor the HTTP connections and store the information itself.

The amount of data in the whole internet session transfer is considerably larger than the information for the authentication sessions transferred in this thesis. We transfered the session authentication cookies as text strings through a SPP connection. This way does not work with larger amounts of data. The larger amounts are transferred in Bluetooth with OBEX. OBEX is a file transfer protocol and transfers the files in binary format without alterations. OBEX was not used in this thesis because a library implementing OBEX in Python on all the platforms used in the thesis was not found.

7.6 Browsers and extensions with cookie handling

The web server uses cookies for state handling of the HTTP connection. Otherwise the HTTP connection is stateless. The cookies are thus intended for the web server and not the client. The start assumption is that the client does not need access to the cookies as default. However, many users want to save important cookies and delete the cookies that are used to track the users browsing in the net. Thus, on some platforms it is possible for the users to manipulate the cookies.

Allowing everyone to access the cookies also opens the door for malicious attackers. An attacker can inject an application to the browser and access the cookies. Even our implementation of the authentication session migration prototype can be used as such injected program with some modifications. The modified prototype could migrate the authentication session to a target device without the user knowing it. The users must only install trusted applications from the internet to prevent malicious attackers from gaining

access to the web browser.

On some operating systems, such as Symbian and Mac OS, the operating system handles the cookies. These operating systems offer a HTTP connection component for the applications. As HTTP is a standardised protocol, it is same for all the browsers. It removes redundant work from all the browsers but at the same time removes some flexibility. It is harder to use components such as the cookies for tasks they are not planned to be used. On the other hand, the implementation of cookies on different browsers have not always strictly followed the official definitions. Placing the HTTP connection handling to the operating system forces all the applications to use that implementation of HTTP. This forces the applications to use the correctly implemented HTTP and protocols affiliated with it because sometimes the applications do not implement HTTP as it is defined.

7.7 From prototype to real application

The implementation of this thesis is only a prototype for testing the authentication session migration. If an application is done based on this prototype, certain issues have to be considered. This section covers all the parts of the prototype and discusses what have to be taken into account when developing them.

The cookie extraction of the prototype works on Mozilla-based browsers Firefox, and Fennec. It was not possible to do the prototype on other browsers because access to the cookies was not possible with their free development tools. Though, about third of the users in the internet browse with different Firefox versions [9] and especially if Fennec becomes popular on mobile devices, it is worthwhile to do the application just for Mozilla-based browsers.

Bluetooth is a viable solution for the information transfer between devices. It offers flexibility with its different protocols. Also, security of the transfer is good because all the parts of the transfer stay under the user's control. Bluetooth does not offer an official protocol stack but different manufacturers have implemented their own protocol stacks. The Python library PyBluez only supports Microsoft and Widcomm Bluetooth stacks on Windows and Bluez stack on GNU/Linux. A usable application needs a better support for different Bluetooth protocol stacks on different platforms. Also, C++ programming language is worth to consider instead of Python as it is native programming language on many mobile devices.

Third part of the migration that has to be considered is the SSO systems. If

the application can migrate authentication sessions from OpenID and Shibboleth, it is probably enough because they seem to be the two most popular SSO systems in web usage. The authentication sessions of the IdPs migrate without problems. The problem with the SSO systems is the protection against replay attacks on the SPs. The SP sessions are protected on Shibboleth by checking that the cookies always come from the same IP address. This prevents the migrated SP session from working but the SSO system creates a new SP session with the migrated IdP session. Therefore, user usually does not see any difference in the migration.

Biggest problem for the application is identifying the cookies that hold the authentication session information. As section 7.2 tells, it is not possible to know in a sure way what cookies hold the authentication session information. Thus, the SSO systems should have a coherent naming policy for the authentication session migration application to work automatically. Other possibility is that the user tells the application which cookies have the session information. Federated SSO system make it possible that user has his identity information only on a few SSO IdPs. Therefore, the amount of the information that would have to be gathered is fairly small. Problem with this approach is that most users do not know anything about the session cookies.

Chapter 8

Conclusions

The implementation in this thesis is a client-side authentication session migration prototype. It migrates the authentication session of the SSO system by transferring the session cookies corresponding to the authentication to a target device. The migration works without problems if the SSO system fulfills following conditions:

- The authentication session cookies must be distinguishable from other cookies in the cookie storage.
- The SSO systems must not have a replay attack protection that rejects the cookies from the different IP address than the originating device of the migration.

The authentication session cookies need a standard for naming on all the cookie-based SSO systems. Standard naming allows the cookie migration application to know which cookies are the authentication session cookies without knowing anything else about the SSO systems the user has authentications on.

Using the HTTP state management protocol is not completely secure [35]. An attacker can harvest the cookies and use them in replay attacks. The cookie-based SSO systems requires a way to protect itself against replay attacks done with these harvested cookies. A common way to protect the SSO system is to check that the cookies come from the same IP address they were given. This prevents the migrated session from working. Some other way to protect the SSO system must be deployed.

Three types of approaches exist for session transfer: client-based, server-based, and proxy-based [26]. The client-based approach was used in this

approach. It is the simplest approach to implement because the platform provides mechanism for the cookie extraction and importation. Also, the client-based approach does not need authentication to the systems same way as an internet proxy does. The client-based approach needs different implementation for all the different browser clients.

The client-based authentication migration needs to access the session authentication cookies. Not all the major web browsers and operating systems that handle the HTTP connection give a free access to the cookie manipulation interface. This prevents implementing the migration application easily on all browsers and platforms. Allowing all developers access to the cookie interfaces would make the development of an application that use the cookies easier. Also, in general, if the cookies and other session information is easy to access, migrating different sessions is easier in the modern mobile environment.

The authentication session migration prototype implemented in this thesis successfully transfers SSO authentication sessions between devices. Thus, the approach used in the prototype is a viable alternative for the authentication session migration. The prototype was tested with OpenID [5] and Shibboleth [8] and it worked without problems.

8.1 Further work

The implementation in this thesis transfers only authentication sessions of cookie-based SSO systems. SSO systems with other session management approaches exist, for example the Higgins identity framework [2]. To be able to transfer all the user's authentication session, also the sessions of these SSO systems using other approaches need to be migrated. Thus, further work on different these different SSO systems is needed.

Bibliography

[1] Central Authentication Service (CAS). URL: http://www.jasig.org/cas, referred December 15th, 2009.

[2] Higgins open source identity network. URL: http://www.eclipse.org/higgins/, referred November 29th, 2009.

[3] Jalimo project (Java for Maemo). URL: https://wiki.evolvis.org/jalimo/index.php/Main_Page, referred December 13th, 2009.

[4] Livejournal (OpenID service provider). URL: http://www.livejournal.com, referred January 16th, 2010.

[5] OpenID (SSO system). URL: http://openid.net, referred January 15th, 2010.

[6] Pubcookie. URL: http://www.pubcookie.org/, referred December 15th, 2009.

[7] PyBluez (Bluetooth library for python). URL: http://code.google.com/p/pyBluez/, referred December 15th, 2009.

[8] Shibboleth (SSO system). URL: http://shibboleth.internet2.edu/, referred January 15th, 2010.

[9] W3Counter - Global Web Stats. URL: http://www.w3counter.com/globalstats.php?year=2010&month=1, referred February 9th, 2010.

[10] Adeyeye, M., Ventura, N., and Humphrey, D. A sip-based web session migration service. In *Proceedings of WEBIST 2009 - Poceedings of the 5th International Conference on Web Information Systems and Technologies* (march 2009), pp. 39–46.

[11] Bolla, R., Rapuzzi, R., and Repetto, M. Handling mobility over the network. In *Proceedings of the 4th International Conference on Future Internet Technologies (CFI '09)* (2009), pp. 16–19.

[12] Bolla, R., Rapuzzi, R., Repetto, M., Barsocchi, P., Chessa, S., and Lenzi, S. Automatic multimedia session migration by means of a context-aware mobility framework. In *Proceedings of The 6th International Conference on Mobile Technology, Application & Systems (Mobility '09)* (September 2009).

[13] Canfora, G., Di Santo, G., Venturi, G., Zimeo, E., and Zito, M. V. Migrating web application sessions in mobile computing. In *Proceedings of WWW '05: Special interest tracks and posters of the 14th international conference on World Wide Web* (may 2005), pp. 1166–1167.

[14] Canfora, G., Santo, G. D., Venturi, G., Zimeo, E., and Zito, M. V. Proxy-based hand-off of web sessions for user mobility. In *Proceedings of MobiQuitous 2005: Second Annual International Conference on Mobile and Ubiquitous Systems: Network and Services* (jul 2005), pp. 363–372.

[15] Carlsen, U. Cryptographic protocol flaws. In *Proceedings of IIIE Computer Security Foundations Workshop VII* (jun 1994), pp. 192–200.

[16] Chalandar, M., Darvish, P., and Rahmani, A. A centralized cookie-based single sign-on in distributed systems. In *Proceedings of ITI 5th International Conference on Information and Communications Technology (ICICT 2007)* (Dec. 2007), pp. 163–165.

[17] Cui, Y., Nahrstedt, K., and Xu, D. Seamless user-level handoff in ubiquitous multimedia service delivery. *Multimedia Tools and Applications 22*, 2 (February 2004).

[18] De Clercq, J. Single sign-on architectures. In *Proceedings of the International Conference on Infrastructure Security (InfraSec '02)* (2002), pp. 40–58.

[19] Dierks, T., and Rescorla, E. RFC 5246: The Transport Layer Security (TLS) Protocol Version 1.2, aug 2008. URL: Http://tools.ietf.org/html/rfc5246, referred September 3rd, 2009.

[20] Diniz, J. R. B., Ferraz, C. A. G., and Melo, H. An architecture of services for session management and contents adaptation in ubiquitous

medical environments. In *Proceedings of the 2008 ACM symposium on Applied computing (SAC t08)* (2008).

[21] Fu, K., Sit, E., Smith, K., and Feamster, N. Dos and don'ts of client authentication on the web. In *Proceedings of the 10th conference on USENIX Security Symposium* (aug 2001), vol. 10, p. 19.

[22] Geneiatakis, D., and Lambrinoudakis, C. An ontology descriptionfir sip security flaws. *Computer Communications 30* (mar 2007), 1367–1374.

[23] Hager, C., and Midkiff, S. An analysis of bluetooth security vulnerabilities. In *Proceedings of IEEE Wireless Communications and Networking (WCNC 2003)* (March 2003), vol. 3, pp. 1825–1831.

[24] Hardy, G. The truth behind single sign-on. *Information Security Technical Report 1*, 2 (1996), 46–55.

[25] Housley, P., Polk, W., Ford, W., and Solo, D. RFC 3280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL Profile), apr 2002. URL: http://tools.ietf.org/html/rfc3280, referred December 20, 2009.

[26] Hsieh, M., Wang, T., Tsai, C., and Tseng, C. Stateful session handoff for mobile www. *Information Sciences 176*, 9 (may 2006), 1241–1265.

[27] Klingenstein, N., and Cantor, S. NativeSPSessions, Jul 2009. URL: https://spaces.internet2.edu/display/SHIB2/NativeSPSessions, referred Septemper 22nd, 2009.

[28] Koponen, T., Eronen, P., and Särelä, M. Resilient connections for ssh and tls. In *Proceedings of USENIX '06 Annual Technical Conference* (may-june 2006).

[29] Kristol, D., and Montulli, L. RFC 2965: HTTP State Management Mechanism, Oct 2000. URL: Http://tools.ietf.org/html/rfc2965, referred June 29th, 2009.

[30] Liu, A., Kovacs, J., Huang, C., and Gouda, M. A secure cookie protocol. In *Proceedings of International Conference on Computer Communications and Networks* (oct 2005), pp. 333–338.

[31] Mate, S., Chandra, U., and Curcio, I. Movable-multimedia: session mobility in ubiquitous computing ecosystem. In *Proceedings of the 5th international conference on Mobile and ubiquitous multimedia* (dec 2006), pp. 8–es.

[32] Morgan, P. nsIFile (Mozilla extension reference), may 2009. URL: https://developer.mozilla.org/en/nsIFile, referred December 15th, 2009.

[33] Neuman, C., Yu, T., Hartman, S., and Raeburn, K. RFC 4120: The Kerberos Network Authentication Service (V5), Jul 2005. URL: Http://tools.ietf.org/html/rfc4120, referred August 11th, 2009.

[34] Park, J., and Dicoi, D. Wlan security: current and future. *Internet Computing, IEEE 7*, 5 (Sept/Oct. 2003), 60–65.

[35] Park, J., and Sandhu, R. Secure cookies on the web. *Internet Computing, IEEE 4*, 4 (Jul/Aug 2000), 36–44.

[36] Parker, T. Single sign-on systems-the technologies and the products. In *Proceedings of European Convention on Security and Detection* (May 1995), pp. 151–155.

[37] Pashalidis, A., and Mitchell, C. A taxonomy of single sign-on systems. In *Proceedings of 8th Australian on Information Security and Privacy (ACISP 2003)* (jul 2003), pp. 249–263.

[38] Rosenberg, J., Shulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and Schooler, E. RFC 3261: SIP: Session Initiation Protocol, jun 2002. URL: http://tools.ietf.org/html/rfc3261, referred December 20, 2009.

[39] Salowey, J., Zhou, H., Eronen, P., and Tschofenig, H. RFC4507: Transport Layer Security (TLS) Session Resumption without Server-Side State, May 2006. URL: http://tools.ietf.org/html/rfc4507, referred Oct 21st, 2009.

[40] Samar, V. Single sign-on using cookies for web applications. In *Proceedings of IEEE 8th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE '99)* (jun 1999), pp. 158–163.

[41] Shepherd, E. nsICookie (Mozilla extension reference), jul 2009. URL: https://developer.mozilla.org/en/nsICookie, referred December 15th, 2009.

[42] Shepherd, E. nsICookieManager (Mozilla extension reference), may 2009. URL: https://developer.mozilla.org/en/nsICookieManager, referred December 15th, 2009.

[43] Shepherd, E. nsICookieManager2 (Mozilla extension reference), aug 2009. URL: https://developer.mozilla.org/en/nsICookieManager2, referred October 7th, 2009.

[44] Shepherd, E., and Smedberg, B. nsIProcess (Mozilla extension reference), may 2009. URL: https://developer.mozilla.org/en/nsIProcess, referred December 15th, 2009.

[45] Song, H., Chu, H., Islam, N., Kurakake, S., and Katagiri, M. Browser state repository service. In *Proceedings of the First International Conference on Pervasive Computing (Pervasive '02)* (2002), pp. 253–266.

[46] Song, H., Chu, H., and Kurakake, S. Browser session preservation and migration. In *Proceedings of Poster Session of WWW 2002* (may 2002), p. 2.

[47] Sullivan, R. K. The case for federated identity. *Network Security 2005*, 9 (2005), 15–19.

[48] Syverson, P. A taxonomy of replay attacks. In *Proceedings of IEEE Computer Security Foundations Workshop VII* (jun 1994), pp. 131–136.

[49] Ye, R., Chan, A., and Zhu, F. Efficient cookie revocation for web authentication. *International Journal of Computer Science and Network Security 7*, 1 (jan 2007).