Authentication Session Migration

Sanna Suoranta, Jani Heikkinen, and Pekka Silvekoski

Aalto University, School of Science and Technology, Konemiehentie 2, 02150 Espoo

Abstract. Consumers increasingly access services with different devices such as desktop workstations, notepad computers and mobile phones. When they want to switch to another device while using a service, they have to re-authenticate. If several services and authenticated sessions are open, switching between the devices becomes cumbersome. Single Sign-on (SSO) techniques help to log in to several services but re-authentication is still necessary after changing the device. This clearly violates the goal of seamless mobility that is the target of much recent research. In this paper, we propose and implement migration of authentication session between a desktop computer and a mobile device. The solution is based on transferring the authentication session cookies. We tested the session migration with the OpenID, Shibboleth and CAS single sign-on systems and show that when the authentication cookies are transferred, the service sessions continue seamlessly and do not require re-authentication. The migration requires changes on the client web browsers but they can be implemented as web browser extensions and only minimal configuration changes on server side are sometimes required. The results of our study show that the client-to-client authentication session migration enables easy switching between client devices in online services where the service state is kept in the cloud and the web browser is acting as the user interface.

1 Introduction

During the last ten years, there has been steady increase in the number of web-based applications and cloud services have become widespread. Often, the services require authentication. As the number of applications has increased, the burden of authenticating to each one of these services has become unbearable to the user. Several single sign-on (SSO) techniques have been developed to help users to cope with their accounts in the various services. The problem, however, further aggravated by the fact that people have many devices such as smart phones, laptops, and notepad computers, and they alternate between these devices depending on the context. This context can be determined by several factors, namely the purpose of the use, time, and location. As a result, the number of devices and accessed web-based applications can create a considerable amount of work for a mobile user since there can be a number of sessions on different devices, each of which require separate authentication. In particular, when the user wants to switch to use another of her devices, for example, from a desktop workstation to a notepad computer, she has to re-authenticate. In order to mitigate this problem and to extend SSO to service access from multiple devices, we have developed techniques for authentication session mobility between personal devices.

Different mobility types include mobility of people, services, session state and terminals [7]. All these are necessary for a ubiquitous computing environment. Many solutions and techniques have been developed for terminal mobility, for example, Mobile IP [33] and Session Initiation Protocol (SIP) [40]. Service mobility means mainly consistent network connection establishment — that devices can connect to different kinds of networks seamlessly. In order to provide personal mobility, which means that a user can use any device and switch devices during a task, session mobility or session migration becomes essential. Historically, session migration has meant the migration of processes or virtual machines mainly in homogeneous server farms, and it has been difficult to implement anything similar in heterogeneous client systems. Fortunately, most new services are accessed with web browsers and the session state information is stored in the server or the cloud. Thus, session mobility in many modern services means simply moving the authentication session, which is the only part of the session that has a state stored on the client device.

Web-based applications have the concept of a session for indicating an authenticated user. The session information is typically stored on the server and the client only stores a session identifier in a cookie. When a user returns to the service during the same session, the web server gets the client identity information from the cookie that is delivered together with the service request. Also, many SSO and federated identity management (FIM) techniques, for example, OpenID [35] and Shibboleth [25], use cookies to indicate the authenticated user. In FIM, the service and authentication have been separated to two distinct providers. The user contacts first the service provider, which then redirects the user to a separate identity provider for authentication. When the user has successfully authenticated herself, the identity provider informs the service provider, and the service provider can then decide whether the user has rights to access the service. In the process, an authentication session is created both between the user device and the identity provider and between the user device and the service provider. The user can reuse the authentication session for another service since she has a cookie from the identity provider that shows who she is, or the identity provider remembers that she has already authenticated herself.

In this paper, we implement client-side migration of the authentication sessions. Our goal was to create a system that requires no changes to the identity provider or service provider. In our prototype implementation, the user can continue using a service after transferring the authentication session cookies from one device to another one. We tested our system using the Shibboleth, OpenID and CAS [26] single sign-on mechanisms.

The paper is organized as follows. First, we describe session migration technologies from the literature in Sec. 2. We introduce federated identity management systems and explain how they use cookies in web browsers for sessions in Sec. 3. Then, Sec. 4 and 5 present the design of our solution for client-side migration of authentication session and how we have implemented it on Firefox. In Sec. 6, we discuss how the implementation can be extended to work on other platforms. In Sec. 7, we evaluate the proposed techniques by testing them. In Sec. 8, we discuss what should be done in order to make the session migration work in all browsers and devices and, finally, Sec. 9 concludes the paper.

2 Related Work on Session Migration

Virtual machine process migration was a widely studied subject already as early as in the 1980s. Milojicic et al. [32] survey the most important process migration implementations before year 2000, for example MOSIX [4], Sprite [16], and Mach [1]. They list reasons why these have not gained wide adoption: complexity, costs, non-transparency, homogeneity, lack of applications and infrastructure, and that users did not need the migration. Later, virtual machine process migration has become essential on server side to guarantee higher performance and shorter out-of-service time, to enable load-balancing and to facilitate fault management [12]. For example, Clark et al. [12] describe how to migrate an entire live virtual operating system and all of its applications in Xen virtual machines. Also other virtualized operating systems provide migration. For example, OpenVZ has an extension called CheckPoinTing (CPT) that allows OpenVZ kernel to save a virtual environment and restore it later even on a different host [37], and another Linux based solution, Kernel-based Virtual Machine (KVM) has similar functionality [29]. Mostly virtual operating systems are used on server side for hosting several services on one physical server and for load balancing. Nevertheless, also client side solution exists: MobiDesk virtualizes the user's whole computing session allowing transparent migration of sessions from one computer to another [5].

A whole virtual operating system is easier to migrate than a single application because all memory and state dependencies are handled inside the kernel as one packet. However, moving only application sessions takes less capacity on the communication path and the participating devices may be able to use different operating systems. A stand-alone application is of course easier to move than an application client that communicates with an external service and needs also connection and session state information on the server side. In this paper, we are more interested in the communicating applications. Communication service sessions can be migrated in many ways in different layers of the protocol stack. Some techniques migrate the session directly between two devices, others use proxies where the session is stored during the migration.

Allard et al. [3] have presented a solution for transferring IPsec context using Context Transfer Protocol (CXTP) [31]. The solution is targeted for mobile nodes that move between networks but it works also for switching between devices. The mobile node has a secure connection using IPsec VPN tunnel through an access router with its Mobile IPv6 home agent. In the context transfer, the access router end of the IPsec tunnel is moved to another access router. The IPsec context consists of IP addresses, security parameter indexes that identify the used security associations (SAs), and other SA information telling the used algorithms and modes etc.

On the transport layer, Secure Socket Layer (SSL) [36] and Transport Layer Security (TLS) [14] allow caching of sessions since creation of cryptographic keys can be heavy. Caching is not always enough on the server side where load-balancing is used in addition. Hatsugai et al. [23] present a way for servers to migrate SSL sessions from one server to another one dynamically when the servers form a cluster but their solution is working on the server side and it is not for the client. Koponen et al. [28] extend the TLS protocol so that sessions can survive changing IP addresses, which means that the client can move in the network. Newer transport layer protocols, such as Stream Control Transmission Protocol (SCTP) [48], which is originally designed for transferring telephone signaling messages over IP networks, provides transport layer mobility by multihoming: the connection can have multiple source and destination IP addresses [9].

Many studies present how multimedia sessions or browser based communication sessions can be migrated. Hsieh et al. [24] introduce three approaches for the browser session migration: client-based, server-based and proxy based. Several implementations for these approaches exist. For example, Basto Diniz et al. [15] introduce session management for the ubiquitous medical environment where sessions can be migrated between devices or even suspended by storing them into a server. Cui et al. [13] have developed a middleware solution for user mobility where the client host uses service discovery to locate the services and store state information and handoff manager moves the session when the user changes the device. Bolla et al. [6] approach the problem of multimedia session migration from different starting point: they introduce a Personal Address to identify users and their sessions instead of the network dependent IP addresses. Moreover, many web service solutions are based on SIP. For example, Shacham et al. [42] have created a SIP based session migration for multimedia content. Their solution has two security features: authentication of the device user with a secure token or close proximity, and privacy features where the participants of a communication session can deny session transfer to less trusted devices. Adeyeye et al. [2] present another SIP based solution that allows transferring session data between two web browsers.

RFC3374 [27] lists Authentication, Authorization and Accounting (AAA) information context transfer as one facilitator of seamless IP mobility. For example, Bournelle et al. [8] extend the above mentioned CXTP protocol for transferring network access authentication sessions that use the PANA protocol [19] from one device to another one in order to speed up handover by avoiding re-authentication. Also, Georgiades et al. [20] added AAA context information to Cellular-IP protocol messages in order to improve handover performance.

Nevertheless, many of these above mentioned solutions, especially the AAA context transfer mechanisms, are targeted mainly for device mobility and changing the access network technology or improving server performance, not for application session migration between devices. Nowadays many applications and services works on top of the HTTP protocol to form the communication channel with the client part that uses web browser as user interface. Even though the basic HTTP is stateless, the service can have session state in the server, and the client only has a session identifier in form of a cookie. This means that there is no reason for complex application state transfer between the client devices. Moreover, underlying communication sessions, e.g. TCP connections, can fail and are re-established often. Thus, there is no reason to migrate communication state either. Only authentication session remains to be migrated in the client side.

3 Federated Identity Systems and Web Session Cookies

Web browsers have become the widely used client platform for services on the Internet. Many web services still have their own user account databases and use password-based authentication but new means for identity management are now available. In Federated Identity Management (FIM) systems, the user account management is separated to its own provider: when a user want to authenticate herself to a service, the service provider forwards the request to an identity provider that verifies the user's identification. The core idea of FIM is that the user needs to log in only once in order to use several services and all the services do not need to maintain user account databases. Moreover, some FIM systems allow the user to choose which identity provider they use.

Two common FIM technologies are OpenID [35] and Shibboleth [25]. OpenID is, as its name says, open for anyone to establish their own identity provider, and the OpenID community provides free implementations and instructions for both identity and service providers. The identity verification methods of identity providers vary from strong smart-card-based authentication of legal persons to weak methods where the proof of identity is that the user can receive email using an address. Contrary to the original idea of openness, OpenID allows service providers to choose which identity providers they accept and many organizations that have several online services use OpenID for account management but accept only their own OpenID identity provider.

The other technology mentioned above, Shibboleth, is based on SAML [39] that is also a public standard and free implementations for it are available. Unlike OpenID, SAML requires formal agreements between the participating organizations, which are usually organized as federations. In Finland, the institutions of higher education have formed a federation called HAKA [18] where the universities can provide common services using their own user accounts for access management. The HAKA federation provides schemas and instructions for both the identity and service providers. Fig. 1 depicts how a service authenticates a user with the help of an identity provider in Shibboleth. The user first opens the webpage of the service. Her connection is redirected to the identity provider. If the service accepts several IdPs, a list is provided for the user before the redirection. The IdP authenticates the user and redirects the connection back to the service provider with information that the authentication was successful. Then, the service can decide if the authenticated user has right to use the service or not.

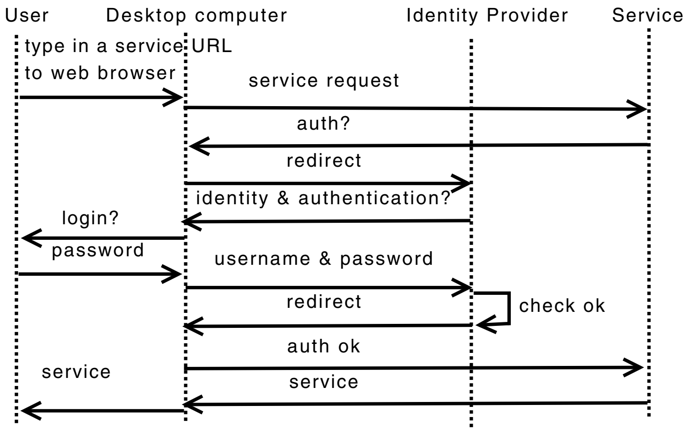

Fig. 1. Shibboleth authentication

In its basic form, an HTTP session is stateless and can consist of many short TCP connections [17]. A web server handles the stateful sessions by sending cookies to the client side web browsers. Samar [41] presents three approaches for cookie based SSO systems: centralized cookie server, decentralized cookie server and centralized cookie login server. In centralized SSO, for example, authentication is done by a centralized entity that gives cookies to services telling the state of the user [10].

Both OpenID and Shibboleth use cookies for storing the authentication session with the identity provider and also possibly for storing the session with service provider. In OpenID, an authentication session is formed between the client and identity provider. Service providers do not necessarily have a session with the client at all. The user must always type in the OpenID identifier since the service providers do not even know if the user is already authenticated to some identity provider. Shibboleth client, on the other hand, creates sessions with both identity and service providers and both of them send their own cookies to the client side. A third cookie maybe created when the user has chosen an identity provider for a service. This means that the user does not need to identify herself while re-authenticating to the service since the service provider knows, based on the cookie, with which identity provider to check that the user still has an active authentication session. The cookies are local to the browser at the client device and neither OpenID nor Shibboleth has any support for sessions that involve multiple client devices or browsers.

4 Design

In this paper, our goal is to design and implement a system that allows the user to switch between devices while using a service that requires authentication and uses single sign-on. Overall, the implementation of SSO migration consists of cookie extraction, creating cookie file, transfer between the devices, importing the cookie and opening the web browser using the same webpage where the user was before the migration. In this section, we describe in detail how all these parts were designed. Silvekoski [46] gives an even more detailed description.

Fig. 2 depicts how a Shibboleth authentication session is migrated from a desktop computer to an Internet tablet device in our implementation. First, the user starts the migration by choosing it from the web browser menu. This starts a browser extension that first extracts the Shibboleth IdP and service cookies and then transfers them to the target device. The target device opens a web browser with the URL of the service that the user was accessing. Since the authentication cookies have been transferred from the other device, the user does not need to re-authenticate and can continue using the service with the mobile device browser. The migration works similarly in the other direction, when moving the session from the mobile device to the desktop computer.

In some cases, however, the service cookies cannot be transferred. If the transfer at the service cookies fails or the service provider does not, for any reason, accept them, the authentication session transfer still succeeds but another step is needed. When a web browser is opened on the target device after the cookie migration, the service redirects the connection to the identity provider. Since the user is already authenticated, the identity provider does not ask her password again. It redirects the connection back to the service provider with the user authentication information.

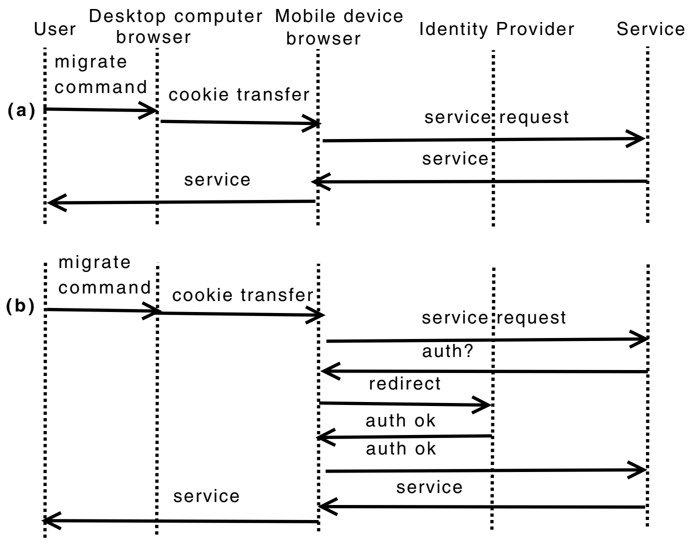

Fig. 2. Authentication session migration with (a) all cookies (b) only authentication cookies

As described above, many SSO systems use cookies to store information about sessions in the client-side web browsers. The session information tells, for example, which user has logged in to the service and how long the session is valid. Moving the cookie from one device and web browser to another one should migrate the session since all the client-side session information is stored in the cookie. Next, we present the design of our cookie based session migration for single sign-on.

Migrating the SSO session requires three steps: extracting the cookies from the original device and web browser, transferring the cookies from the original device to the target device, and importing them into the target device browser. Fig. 3 depicts the SSO cookie migration. When the user chooses to migrate the session, a browser extension first extracts the cookies from the browser and writes them into a cookie file. Then the browser extension starts a transfer module and gives it the location of the cookie file and the URL of the current page on the browser that tells the service location. The transfer module creates a connection to transfer module on the target device and sends the cookie file and URL over to it. The transfer module on the target device imports the cookie into the web browser and starts the browser with the given URL. The migration works similarly both ways between the two devices.

The method for extracting and importing the cookies depends on how they are stored on the original and target device and which web browsers are used. Either the browser or the operating system handles the cookies, but usually an interface for cookie management is offered. The extraction application fetches the cookie information, stores it into a file, and passes the file to the transfer module. Usually, SSO uses session cookies which are stored in the memory rather than on the disk. For this reason, the cookies cannot be simple read from a file and an API for accessing them is needed. If the browser manages the cookies, the extraction is done with a browser extension. Otherwise, the

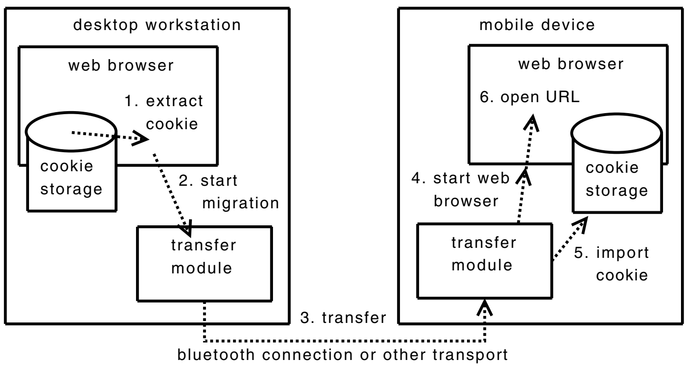

Fig. 3. Shibboleth authentication

operating system provides the cookie information but also a browser extension is needed since that allows the user to start the migration and gives the URL of the current page.

The cookies are transferred between devices in files. If the cookies are stored in the memory, the cookie extractor creates a file and stores the cookie information into it in the format in which they were stored in the memory. SSO cookies are encrypted and thus the exact byte values are essential so that the cookie data does not change. However, it is not always necessary to transfer all cookie data, just the name, value, domain and path must be transferred. The cookie domain and path tell the owner of the cookie, namely the service, whereas name and value give the purpose of the cookie and session data. The cookie information is stored in a file where each cookie is four lines long with following content: name, value, domain and path. Several cookies can be stored into one file.

The transfer module takes care of moving the cookie file from the original device to the target device. It has two behaviors: a client that sends the cookie from the original device and a server that receives the cookie on the target device. In our implementation, when the user wants to migrate her session, she starts the transfer server on the target device and clicks a start button in a menu of the web browser on the original device. The target device shows a dialog that tells where the connection is coming from so that the user can be sure the cookies are coming from the correct original device. In addition to the cookie, also the URL of the current page on the web browser is sent. The transfer client reads the cookie file, establish a connection with the transfer server that waits for the connections on the target device, and transfers the cookie and URL over to it. After successful copying the cookie to the target device, the transfer client removes it from the original device. Otherwise, the session might remain open on the original device, and this might confuse some services. Also, ending the session on the target device does not remove the cookie on the original device but only on the target device and the session might accidentally stay open even when the user thinks she has logged out and closed the browser. On the target device, the corresponding transfer server receives the cookie file, stores it, and starts the local web browser with the URL it received. After the transfer, also the transfer server closes itself. In future implementations, the transfer module could always run as a daemon process, which would slightly simplify the user experience. Next, we describe the implementation in more detail.

5 Firefox Implementation

The SSO technologies chosen for the implementation were Shibboleth and OpenID. Both of these are freely available open source systems and thus easy to take into use. Shibboleth is used in Finnish universities and there are several services available. All students and staff members have their own user accounts. OpenID has several identity providers and services available on the Internet. We chose to use Mozilla Firefox on the desktop computer running Windows XP operating system and Fennec on the Nokia N810 Internet Tablet running Maemo OS. The Fennec browser was a beta, which caused some problems that we describe later.

We used Bluetooth for transferring the cookies. It provides encrypted connections between the devices and the devices are identified with their unique addresses. The devices can be found using Bluetooth service discovery. If the connecting device is unknown, the user is asked to approve the connection. The devices can also be paired to remember each other. Bluetooth is designed for personal area connections and transferring data between one user's devices, which means that the pairing usually needs to be only once for new devices. Bluetooth has built-in encryption and its security is generally considered adequate [22]. We used the Python language to implement the transfer module that receives the connection at the target device since the only Java edition, Java micro edition (JME), that support Bluetooth, does not work on N810.

Both of the used web browsers save the session cookies in the memory of the device but offer a possibility to fetch the cookies using scripting. Moreover, same extensions such as the cookie handling extension work on both Firefox and Fennec since both are Mozilla-based web browsers. The extensions are cross-platform component object model (XPCOM) components than can use cross-platform libraries. Mozilla extensions can be done with JavaScript. We used nsICookie [43], nsICookieManager [44], nsIFile [34] and nsIProcess [45] interfaces and components. The first difference between the browsers is that Fennec does not have drop down menus. Thus, the user starts the migration by choosing it from Firefox drop-down menu on the desktop computer or by clicking a button in the Fennec menubar on the mobile device.

First, in the authentication session migration, nsICookieManager is used to extract the cookie data from the web browser memory. The cookies are in nsICookie format as UTF-8 text, which consists of the cookie name, value, host and path, and additional information. In the extraction process, all cookies in the browser memory are enumerated and the needed SSO cookies are chosen based on their name. Neither of our example FIM systems, OpenID and Shibboleth, has strict instructions for naming the cookies. OpenID uses usually a combination of the identity provider name and openid tags, for example _exampleidp_openid. Shibboleth names usually the service session with _shibsession_ and application code and the authentication session with _idp_session but both of these can be changed using system attributes. The cookie names also depend on the used authentication method in Shibboleth. The recognized cookies are stored in a file in the root directory of the browser extension using the nsIFile interface and the UTF-8 encoding.

For transferring the cookies, nsIProcess starts the python application for Bluetooth and gives it the location of the cookie file and the current URL of the web browser. We used an external Python library called PyBluez [21] for the Bluetooth operations.

It works both on Windows XP and Maemo, so that the same code can be used on both devices. Since cookies are small text files, we used the Bluetooth RFCOMM serial port profile (SPP) to transfer them. First, the Bluetooth client starts the device and service discovery. Of course, the Bluetooth server on the target device must be already waiting for the connections. The client asks the user if the correct target device is found by showing a dialog that gives the device identifiers of discovered devices. The dialog is done with native graphical library of N810, GTK+, since it was harder to find a browser-integrated UI library that works on the internet tablet. When the target device has been selected, the client opens the connection and transfers the cookie information file as a string. Before client closes, it deletes the cookie from the original device using nsICookieManager.

On the target device, the transfer server module receives the message. It stores the URL and writes the cookie into a file. Then, it starts the web browser that uses nsIFile to read the cookie and nsICookieManager to import it into the browser memory. When the authentication session is migrated from a mobile device to the desktop computer, the Firefox web browser is started on the desktop computer with the Python web browser library. In the other direction, this could not be done since Fennec is a new browser still under development. As an intermediate solution, we started the Fennec browser with the subprocess command in Maemo by executing a shell script and the user must browse to the right page herself. Next, we discuss how to extend the same process to work in other web browsers and devices.

6 Porting to Other Browsers and Operating Systems

The way the cookies are stored depends on the device, operating system and web browser. In addition, there are two kinds of cookies, session and persistent cookies, whose storage differs. For example, both the identity and service cookies are session cookies in Shibboleth but the “Where are you from” (WAYF) cookie that allows user to store chosen identity provider into the web browser is a persistent cookie. Different web browsers on different platform handle the cookies in their own way. Usually, persistent cookies are stored in the file system while session cookies exist only in the browser memory. Therefore, accessing the cookies differs between devices and browsers. In order to migrate the authentication session, full access to session cookies is necessary since the cookies must be extracted from the web browser on the original device and entered to the web browser on the target device.

Table 1. The session cookie handling in different browsers

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

Table 1 summarizes the handling of cookies on different browsers. In Windows environment, persistent cookies are stored in the file system and the session cookies in the device memory in many popular web browsers, namely Internet Explorer (IE), Mozilla Firefox, and Opera. The location and format of the stored persistent cookies differs between browsers. IE stores the persistent cookies into separate files in its cookie directory but it does not give developers opportunities to manipulate the session cookies in the device memory. Mozilla Firefox offers wide possibilities to extend the browser, and one existing extension offers programmers full control of the stored cookies. Opera offers an editor to the user for manipulating both session and persistent cookies manually but only persistent cookies can be extracted. Thus, the user must first change a session cookie into a persistent one before it can be transferred, and scripting cannot do this. In Mac OS, cookies are handled differently: the operating system offers an HTTP package that handles all the cookies and offers the possibility to add and manipulate cookies freely.

Mobile devices are even more heterogeneous with respect to their operating systems and browsers. A browser that works on all mobile operating systems does not exist, and even all programming languages are not available on mobile devices. Many popular browsers have their own mobile version that has a lighter graphical user interface than the desktop computer version. For example, of the Mozilla based browser, MicroB works on the Nokia Maemo operating system and Fennec has a beta for Nokia N810 and an alpha version for Window mobile. Most Firefox extensions should work on Fennec. Opera Mini, on the other hand, has a completely different approach to mobile browsing: it uses a proxy that compresses and preprocesses the web pages for the device, and the proxy handles also the cookies. Moreover, the operating system handles the cookies for the mobile version of Apple Safari that works on the IPhone. Similarly, the Symbian OS handles the HTTP connections and cookies for Browser, which is the mobile version of Safari on Symbian operating systems.

7 Experimental Evaluation

For testing the authentication session migration, we performed three experiments using different SSO technologies. First, we tested OpenID authentication session migration with Livejournal [30] as the service provider since it accepts other OpenID identity providers than its own. We used claimID [11] as the OpenID identity provider since it allows creating new accounts easily. Migrating the session cookie named _claimid_openid migrates the authentication session into the target device where the user could continue using the service. OpenID service provider differ in the ways the cookies are implemented and, in order to migrate Livejournal SP, two cookies were needed: ljloggedin and ljmastersession. We cannot be sure if we migrated also other information than the authentication session with these two cookies. In Livejournal, the user can choose whether the service provider will check that the client IP address for the session remains constant. Insisting that the IP address does not change prevented the session migration as was expected since the devices have the different IP addresses.

The second test was done using Shibboleth, which is used to authenticate users at our university. Thus, we had one identity provider, the university, and we tried several different services. Unfortunately, many of the services required that the connections are from the same IP address, which prevents the session migration. The address is stored in the cookie and it cannot be changed. Since we did not have possibility to reconfigure the services, we tested the migration only with those providers that allowed the client to change its IP address. For them, the migration works fine with the Shibboleth SSO.

In addition to the two federated solutions, we tested another centralized authentication mechanism that is used in social media services in the OtaSizzle project [47] at the university. These services use the Central Authentication Service (CAS) [26] to authenticate users, and the experiments showed that transferring its cookies successfully migrates the authentication session. The service session could be continued after migration without re-authentication.

Our main goal was to create system where authentication session can be migrated from one device to another in a way that re-authentication is not necessary and no or minimal modifications for the server side are required. Of course, the session migration should be faster and require less input from the user than re-authentication. Moreover, the user should be able to continue her browser session from the same URL and logical state on the target platform.

From the session migration point of view, our prototype fulfills the requirements. Transferring the authentication session cookies were enough and no additional information was required for the session migration on the tested services. Migration of cookies on the client side did not require changes on server side. OpenID worked directly with its default settings. Shibboleth, which has replay attack prevention on by default, did not work since it checks that also the authentication session cookies come from the same IP address. This means that service providers should be configured not to check the IP address in order to allow user to migrate the authentication between her devices. This configuration enables also mobile computers to continue their sessions after moving between access networks.

From the usability point of view, the migration is faster than typing the passwords on the mobile device. However, the Bluetooth device discovery sometimes took a long time. For example, Windows repeated the service discovery of PyBluez devices four times in order to be sure that all devices were found. To speed up the discovery, the searching for services can be restricted to already paired devices. The other requirement, that the user can continue from the same URL and state of the service, is usually met since web browsers can be started with a command line or shell script with the starting URL as a parameter. In our prototype implementation, the continuing on the same location worked only when the session was migrated from mobile device to a desktop computer, not vice versa, since the Fennec browser used on mobile device was only a beta version and did not have the required feature of starting on the given URL. This will be easily fixed when the browser becomes more complete.

Our main goal was to move only the authentication session, not the whole user session, because the user session is maintained by the web server, and by the cloud services in the future, and only the authentication session binds the user connection to the services on the client side. From the server point of view, the migration is tantamount to the client moving to a different IP address and the user pressing the refresh button on the browser.

8 Discussion

In our tests, we showed that transferring the authentication session cookies migrates the authentication session and, in most cases, the entire user session between devices. In order to take the authentication session migration into wider use, following steps are required:

– Standardization of the API for accessing authentication cookies in web browsers,
– Standardization of the naming of the authentication cookies in SSO systems,
– Recommendation not to bind the cookies to IP addresses but to use some other replay attack protection technology,
– Defining standard ways to transmit the cookies over Bluetooth, IP and other channels, and
– Definition of a cookie file format for cookie transfer.

Technically these changes are fairly easy to do, as shown is this paper, but the hard part is the interoperability between many devices and browsers. Thus, standardization is needed.

The methods for *accessing cookies* differ between web browsers and devices as described in Sec. 6. Nowadays, accessing the session cookies is not always possible at all or requires actions from the user. In order to enable universal authentication session migration, the session cookies must be available through an API in all operating systems and web browsers. Then, the migration extension can extract the cookies for transfer.

Naming cookies in a standard way in SSO specifications helps identifying them for the migration. The FIM specifications should give stricter guidelines for naming the cookies.

The IP address of the client is often stored in the cookies to prevent connections from other client hosts than the original one. This is a historical feature to prevent sniffing of the authentication cookies in services that do not use SSL/TLS to protect the cookies. In such services, the cookies may be transferred as plain text. An attacker can record the cookie and send it to a server pretending to be the original communication partner and thus hijack the connection [38]. The service provider mitigated this threat by accepting cookies only from the current IP address of the user. In OpenID, the administrator of the identity provider must take the protection into use. Shibboleth, on the other hand, checks by default that, when the connection is redirected from the service provider to the identity provider and back, the IP address of the client remains the same, and that the client sends the cookies always from the same IP address during the following session. Session migration requires such controls to be disabled. Some other means to prevent replay attacks with stolen cookies should be used. For example, SSL/TLS with client certificates prevents the attack.

A secure connection for the cookie transfer between the user's devices must be easy to take into use. We used Bluetooth that provides easy way to securely pair the devices, but the devices offer many other possibilities. For example, a mobile devices could use a WLAN connection in peer-to-peer mode to connect to other devices without external gateways, or the connection could be created through the Internet using an access point. WLAN has its own security mechanism called Wi-Fi Protected Access (WPA). An Internet proxy could also be used to deliver the cookies and the connections from the two devices to the proxy protected with TLS. For authentication session migration, creating a secure connection must be easy and not to require active participation from the user after the initial setup. The device discovery and verifying the connection parties must happen transparently after the user initiates the session transfer. In this respect, our current implementation needs to be developed further: the user should not need to start the migration on both devices on every migration but only to pair the devices on the first connection.

Cookie transfer file format is the last important part of the migration. The devices may use different encoding for files and text but the cookie information must not change during the transmission or when storing it to the target browser. Cookies often contain special characters, and thus using the same encoding for the information on the original device and on the target device is important.

9 Conclusion

In this paper, we have experimented with authentication session migration based on the transfer of client-side cookies. Many web services use cookies for recognizing the users and for storing information about the service state. Also, the Shibboleth, OpenID and CAS SSO technologies use cookies to tell that a user has already authenticated herself. Moving the cookies that the identity provider created for the user enables her to continue using the service with another device without re-authentication.

The session transfer does not require any changes on server side if the authentication session cookies can be identified by standard names on the client side and the server is not configured to use IP-address-based replay attack protection. Our current implementation relies on Bluetooth device pairing for secure transfer of the session state and shows a dialog to the user for choosing the target device before migrating the authentication session cookies. We had minor performance problems with the Bluetooth service discovery when the devices were not continually connected but otherwise the migration was fast enough to be used regularly.

Client-side session migration works well in a situation where the only party that knows which services and which identity providers are in use is the client. A drawback of the client side implementation is that all different browsers on different devices and platforms need their own extensions since the cookies are handled very differently in these. Solving this problem requires standardization of some SSO features that have currently been left as configuration and implementation options. In the future, online services increasingly are such that the service state is kept in the cloud and the web browser is acting as the user interface, and only the authentication binds the service session to a device or a operating system. Based on the results of this paper, we believe that client-to-client session migration for such services is easy to implement and should become a regular feature of web browsers and SSO services.

References

1. Accetta, M., Baron, R., Bolosky, W., Golub, D., Rashid, R., Tevanian, A., Young, M.: Mach: A new kernel foundation for UNIX development. In: Proceedings of the Summer USENIX Conference (1986)

2. Adeyeye, M., Ventura, N.: A sip-based web client for http session mobility and multimedia services. Computer Communications 33(8) (2010)
3. Allard, F., Bonnin, J.M.: An application of the context transfer protocol: IPsec in a IPv6 mobility environment. International Journal of Communication Networks and Distributed Systems 1(1) (2008)
4. Barak, A., Laden, O., Yarom, Y.: The NOW MOSIX and its preemptive process migration scheme. Bulletin of the IEEE Technical Committee on Operating Systems and Application Environments 7(2), 5–11 (1995)
5. Baratto, R.A., Potter, S., Su, G., Nieh, J.: Mobidesk: mobile virtual desktop computing. In: MobiCom 2004: Proceedings of the 10th Annual International Conference on Mobile Computing and Networking (2004)
6. Bolla, R., Rapuzzi, R., Repetto, M., Barsocchi, P., Chessa, S., Lenzi, S.: Automatic multimedia session migration by means of a context-aware mobility framework. In: Mobility 2009, The 6th International Conference on Mobile Technology, Application & Systems (2009)
7. Bolla, R., Rapuzzi, R., Repetto, M.: Handling mobility over the network. In: CFI 2009: Proceedings of the 4th International Conference on Future Internet Technologies (2009)
8. Bournelle, J., Laurent-Maknavicius, M., Tschofenig, H., Mghazli, Y.E.: Handover-aware access control mechanism: CTP for PANA. Universal Multiservice Networks (2004)
9. Budzisz, L., Ferrús, R., Brunstrom, A., Grinnemo, K.J., Fracchia, R., Galante, G., Casadevall, F.: Towards transport-layer mobility: Evolution of SCTP multihoming. Computer Communications 31(5) (March 2008)
10. Chalandar, M.E., Darvish, P., Rahmani, A.M.: A centralized cookie-based single sign-on in distributed systems. In: ITI 5th International Conference on Information and Communications Technology (ICICT 2007), pp. 163–165 (2007)
11. claimID.com, Inc: claimID (2010), http://claimid.com (referred 2.8.2010)
12. Clark, C., Fraser, K., Hand, S., Hansen, J.G., Jul, E., Limpach, C., Pratt, I., Warfield, A.: Live migration of virtual machines. In: NSDI 2005: 2nd Symposium on Networked Systems Desgin and Implementation. USENIX Association (2005)
13. Cui, Y., Nahrstedt, K., Xu, D.: Seamless user-level handoff in ubiquitous multimedia service delivery. Multimedia Tools and Applications 22(2) (February 2004)
14. Dierks, T., Rescorla, E.: The transport layer security (TLS) protocol version 1.1. RFC 4346, IETF (April 2006)
15. Diniz, J.R.B., Ferraz, C.A.G., Melo, H.: An architecture of services for session management and contents adaptation in ubiquitous medical environments. In: SAC 2008: Proceedings of the 2008 ACM Symposium on Applied Computing (2008)
16. Douglis, F.: Process migration in the Sprite operating system. In: Proceedings of the 7th International Conference on Distributed Computing Systems, pp. 18–25 (1987)
17. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T.: Hypertext transfer protocol – http/1.1. RFC 2616, IETF (June 1999)
18. Finnish IT center for science (CSC): HAKA federation, http://www.csc.fi/english/institutions/haka (referred 10.2.2010)
19. Forsberg, D., Ohba, Y., Patil, B., Tschofenig, H., Yesig, A.: Protocol for carrying authentication for network access (PANA). RFC 5191, IETF (May 2008)
20. Georgiades, M., Akhtar, N., Politis, C., Tafazolli, R.: Enhancing mobility management protocols to minimise AAA impact on handoff performance. Computer Communications 30, 608–628 (2007)
21. Google: Pybluez (bluetooth library for python), http//code.google.com/p/pybluez/ (referred 15.12.2009)
22. Hager, C., Midkiff, S.: An analysis of bluetooth security vulnerabilities. In: Proceedings of IEEE Wireless Communications and Networking (WCNC 2003) (March 2003)

23. Hatsugai, R., Saito, T.: Load-balancing SSL cluster using session migration. In: AINA 2007: Proceedings of the 21st International Conference on Advanced Networking and Applications. IEEE Computer Society (May 2007)
24. Hsieh, M., Wang, T., Sai, C., Tseng, C.: Stateful session handoff for mobile www. Information Sciences 176(9), 1241–1265 (2006)
25. Internet2: Shibboleth (2006), `http://shibboleth.internet2.edu/` (referred 5.9.2006)
26. Jasig: Central authentication service (CAS), `http://www.jasig.org/cas` (ref. 15.1.2009)
27. Kempf, J.: Problem description: Reasons for performing context transfers between nodes in an IP access network. RFC 3374, IETF (September 2002)
28. Koponen, T., Eronen, P., Särelä, M.: Esilient connections for SSH and TLS. In: USENIX Annual Technical Conference (2006)
29. KVM: Kvm migration, `http://www.linux-kvm.org/page/Migration` (referred 27.7.2010)
30. Livejournal: Livejournal, `http://www.livejournal.com` (referred 16.1.2010)
31. Loughney, J., Nakhjiri, M., Perkins, C., Koodli, R.: Context transfer protocol (CXTP). RFC 4067, IETF (July 2005)
32. Milojicic, D.S., Douglis, F., Paindaveine, Y., Wheeler, R., Zhou, S.: Process migration. ACM Compuring Surveys 32(3), 241–299 (2000)
33. Montenegro, G., Roberts, P., Patil, B.: IP routing for wireless/mobile hosts (mobileip) (concluded ietf working group) (August 2001), `http://datatracker.ietf.org/wg/mobileip/charter/` (referred 26.7.2010)
34. Morgan, P.: nsIFile (mozilla extension reference), `http://developer.mozilla.org/en/nsIFile` (referred 15.12.2009)
35. OpenID.net: Openid.net (2008), `http://openid.net/`
36. OpenSSL: Openssl project (2005), `http://www.openssl.org/` (referred 17.10.2008)
37. OpenVZ: Checkpointing and live migration (September 6, 2007), `http://wiki.openvz.org/Checkpointing_and_live_migration` (referred 27.7.2010)
38. Park, J.S., Sandhu, R.: Secure cookies on the web. IEEE Internet Computing 4(4), 36–44 (2000)
39. Ragouzis, N., Hughes, J., Philpott, R., Maler, E., Madsen, P., Scavo, T.: Security assertion markup language (saml) v2.0 technical overview. Tech. rep., OASIS (February 2007)
40. Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Sparks, J.P.R., Handley, M., Schooler, E.: Sip: Session initiation protocol. RFC 3261, IETF (2002)
41. 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 1999), pp. 158–163 (June 1999)
42. Shacham, R., Schulzrinne, H., Thakolsri, S., Kellerer, W.: Ubiquitous device personalization and use: The next generation of IP multimedia communications. Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP) 3(2) (May 2007)
43. Shepherd, E.: nsICookie (mozilla extension reference), `http://developer.mozilla.org/en/nsICookie` (referred 15.12.2009)
44. Shepherd, E.: nsICookieManager (mozilla extension reference), `http://developer.mozilla.org/en/nsICookieManager` (referred 26.7.2010)
45. Shepherd, E., Smedberg, B.: nsIProcess (mozilla extension reference) (May 2009), `http://developer.mozilla.org/en/nsIProcess` (referred 15.12.2009)
46. Silvekoski, P.: Client-side migration of authentication session. Master's thesis, Aalto University School of Science and Technology (2010)
47. Sizzlelab.org: Otasizzle (April 2010), `http://sizl.org/` (referred 28.7.2010)
48. Stewart, R.: Stream control transmission protocol. RFC 4960, IETF (September 2007)