Security in Migratory Interactive Web Applications

Giuseppe Ghiani, Lorenzo Isoni, Fabio Paternò
CNR-ISTI, HIIS Laboratory
Via Moruzzi, 1
56124 Pisa, Italy
{giuseppe.ghiani, lorenzo.isoni, fabio.paterno}@isti.cnr.it

ABSTRACT
In ubiquitous environments migratory interactive applications allow users to perform their tasks continuously across various devices. Users can push and pull migratory Web applications from one device to another for various reasons. However, the flexibility of such pervasive applications raises various security issues, such as the risk of theft of private information from the migrated user interfaces or the intrusion of malicious versions of the applications replacing the original ones. In this paper, we analyse such risks and present a number of solutions to address them in a client/server-based solution for supporting secure migration of interactive Web applications.

Categories and Subject Descriptors
H.5.m. [**Information interfaces and presentation (e.g., HCI)**]: Miscellaneous. K.6.5. [**Management of computing and information systems**]: Security and protection *authentication, unauthorized access (e.g., hacking, phreaking).*

General Terms
Algorithms, Security, Human Factors.

Keywords
Migratory Interactive Web Applications, Security, Multi-device Environments.

1. INTRODUCTION
In recent years there has been an increasing interest in solutions able to exploit the technological offerings of the mass market in terms of variety of devices characterized by widely varying interaction resources (such as screen size, support for vocal interaction, touch-support, etc.). In current multi-device environments it is important to support flexible access mechanisms, which should consider that users often need to move and would like to opportunistically exploit the devices that dynamically become available. A recent study [15] highlighted that most of consumers' time is spent in front of a variety of interactive devices, which can be used both sequentially (i.e. by moving from one device to another) and simultaneously (i.e. using more than one device at the same time). According to that study, sequential usage prevails on the simultaneous, and there is a need for enabling users to preserve their interaction state when moving the task performance between devices.

Application migration is a type of multi-device support in which users can dynamically change device and still continue to perform their tasks from the point they left off in the source device. Environments supporting migratory interactive Web applications allow users to dynamically push and pull them from one device to another for various reasons. Web migration implies that the client-side part of a Web application is moved automatically and in real time from the browser of a source device to the browser of a target one. This is done without particular user intervention (i.e. the user is not requested to insert further information, such as a URL, on the target device). When the applications move to another device the state of their interactive part (i.e. the result of user interactions) is preserved as well without requiring any support from the server side of the application. Some benefits of Web migration are the task continuity when changing device, and the possibility of sharing contents and functionalities with other users (e.g., when migrating an interface towards one or multiple users). In addition, migratory environments can even allow users to interactively select the parts of the interactive application that they would like to migrate to the target device, thus enabling user-driven adaptation. In this paper we consider a migration platform independent of the Web application servers. It is dedicated to providing additional services (i.e. partial/total migration) to existing applications, through the support of a proxy server. Thus, due to the architecture of such platform, migrations across various devices can raise additional security risks, concerning the theft of private information in the interactive applications or the intrusion of bogus versions of the interactive migrating application to replace the original ones. For example, a user can enter personal information and confidential data while booking a room on a mobile device and then can migrate it to a desktop system with the risk that the data are stolen in some way. To address such issues, we present a set of techniques that aim to preserve security in this type of context.

In the paper, after discussing related work, we introduce the security issues in migration of Web applications across multiple devices. Next we briefly describe an architecture supporting migration of interactive Web applications by describing its functionalities, main components and the communications between them. We then analyse the possible security risks arising from that solution, by dividing them into theft of information and false input. We continue by presenting design and implementation of a number of techniques that allow the users to avoid security risks when

MUM '12, December 04 - 06 2012, Ulm, Germany.

accessing the Migration Platform functionalities and performing proxy-based navigation. We also report on how the proposed solution has been validated with some well-known existing Web applications. Lastly, we draw some conclusions and provide indications for future work.

2. RELATED WORK

Security and privacy in multi-device interaction of nomadic users has been tackled by Arthur and Olsen [2]. They proposed XICE, a toolkit for safely distributing UIs from devices to annexed displays, and dealt with potential risks related to the usage of (public) annexed devices. The goal is to protect user privacy and security from risks arising when application windows are shared with external devices that are considered untrusted. Although we also consider the main security and privacy issues, another difference with our work is that the XICE is a windowing toolkit for creating new applications, while the Migration Platform we consider is aimed at making existing applications migratory. XICE is thus a tool for developers, while we consider security issues for end users interacting with migratory Web applications.

Deep Shot [3] is a tool supporting migration that exploits the camera of a smartphone to detect the current application and its state but in this work the associated security issues were not addressed. In general, making the actual security support in Web applications access more user perceivable was addressed in [10], where the authors propose a Firefox plug-in to include a bar whose colour indicates the actual security level. However, that work did not consider the issues of security when Web applications are accessed in multi-device environments.

More in general, privacy and security concerns regarding interaction with public displays have been investigated as well as issues related to sharing applications among multiple users/devices.

User's behaviour towards public displays in public or semi-public spaces has been recently studied in [1]. The general findings of that study indicate that inputting personal information in shared displays is perceived as privacy affecting. Users are afraid of leaving personal information on a public display because it can be caught by people watching at the display while it is being inputted or can be subsequently found by other users of the display.

Some interesting issues concerning access to public devices were also addressed by [14], which supports the use of public fixed displays with the possibility of hiding private information while allowing mobile devices to show it and support user input. In our work we do not consider the issues of showing information in public devices but we rather focus on the security risks associated with dynamic migration of interactive Web applications. An architecture that affords mobile user greater trust and security when browsing the internet (e.g., when making personal/financial transactions) from public terminals at Internet Cafes or other unfamiliar locations is presented in [13]. This is achieved by enabling Web applications to split their client-side pages across a pair of browsers: one untrusted browser running on a public PC and one trusted browser running on the user's personal mobile device, composed into a single logical interface through a local connection, wired or wireless. However, their solution involving two devices for accessing a Web application can generate some usability issues. Thus, we have investigated different solutions for preserving both usability and security when accessing Web applications in pervasive environments.

Muse [16], a system for ubiquitous computing through mobile devices, relies on a safe approach for application sharing and migration in cloud environments. The underlying platform of Muse, named CyberLiveApp [8], provides privacy and security to the users that need to share applications with (or to migrate them towards) other users. A common issue between CyberLiveApp and our Migration Platform is maintaining the application state during migration/sharing, particularly in case of security concerns (e.g., when the application interface involves confidential data). The authors of CyberLiveApp claim to have achieved a fine-grained control of security and privacy by letting the user choose which application windows to share with others through a proxy-based filtering mechanism. Differently, we propose the possibility for the users to customize their protection by specifying the privacy level. In our support, the target device(s) will be able to access the functionalities of the original interface according to such parameters. The main difference with our approach is, however, that CyberLiveApp is based on services of virtual machines in the cloud, while our Migration Platform is able to support existing Web applications.

An agile process for developing secure Web applications is presented in [5], discussing methods that integrate security design within the development process. The authors state that security of Web applications must be tackled at design time, rather than after implementation. This approach, however, was not suitable with our case study, since our environment should make able to migrate existing applications, even if they were designed without addressing security risks.

Satoh and Tokuda [12] consider service compositions assuming that the atomic services comply with own security policies. When atomic services are composed, inconsistencies can arise between their policies and developers may have to perform consistency checks “by hand”. In order to avoid this overhead and to enable even developers without expertise in security issues to compose automatically services, the authors propose a logic-based architecture for policy composition. Process definitions, service descriptions and data protection policies are given as inputs. The output consists of a composite service description with security policies, which is then used to generate the concrete policies. In our approach we cope with existing Web applications (and associated services) that are accessed through a migration server, differently from [12] where the atomic services are all assumed to be secure.

To summarise, we note that the specific security issues that can be involved in emerging ubiquitous migratory interactive applications have been underexplored. This paper aims to describe possible solutions for such issues.

3. SECURITY ISSUES IN WEB APPLICATION MIGRATION

Web application migration allows users to change device and continue the performance of their tasks from the point they left off in the source device. This implies preserving the state of the Web application even on the client-side. Such state includes input entered in forms, cookies, sessions, etc.

An example Web migration is represented by users accessing an online store like Amazon: they log in, browse the site, and enter some input in forms to query the product database. In the case shown in Figure 1, the user has previously logged in on the desktop device and has added two items to the shopping cart (upper part). After migration to the mobile device (lower part), the page

is state-persistent: the partially filled in form (see 3 in Figure 1) has kept its previously inserted values and the user session is maintained (see 1, 2). Thus, if further navigation were performed from the migrated page (e.g., a search or the addition of an item to the shopping cart), the user session containing the items in the session chart would be preserved throughout the navigation.

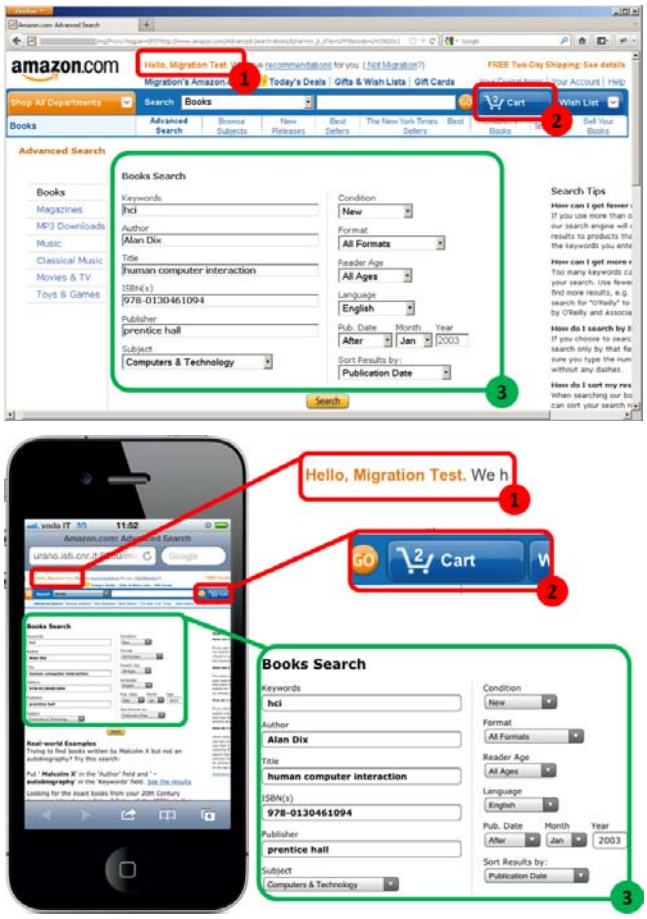

Fig. 1. Example of Migration involving Personal data.

Relevant security issues arise upon migration triggering, i.e. when the interface moves from the originating device to the target one. Indeed, what is actually transferred is not the simple source code of the page, but the current DOM (i.e. the representation of the current Web application within the originating browser), including the interaction state which might contain confidential data. For instance, a page with a login form may contain a username and a password, which the user would not like to be transferred "in clear" within the network.

A secure Migration Platform should then be able to detect security and privacy issues of the navigated Web pages to act accordingly when migration is triggered. In addition, security must be granted in general during the whole interaction session with the Web application through the Migration Platform: this includes the phase before migration triggering, and the post-migration phase, i.e. during further navigation from the migrated page.

As a starting point to identify the specific security lacks that may affect the migration process, it is worth referring to the best known types of attacks that threaten user security and privacy. For For the sake of clarity, we have grouped the typical attacks into two main categories: those devoted to information theft and those aimed at illicitly impersonating the user through false input, though both attack strategies are often initiated by stealing some user data. The following two subsections discuss both types of security problems and how they can be originated. The next sections, instead, describe first the architecture of the proposed Migration Platform, and then its intrinsic security risks and how they have been tackled to improve the security of the environment.

3.1 Information theft

Personal information, such as credentials or navigation chronology, can be easily stolen if data are exchanged transparently between client and server. The strategies of listening to the packets exchanged between client and server, known as *eavesdropping*, are often aimed to get the user confidential data, such as access credentials. Eavesdropping is among the most serious threats of the systems that exchange data without encryption.

The access credentials can be stolen even by repeatedly querying the Web application server. A similar strategy, known as *brute force*, can be performed by iteratively trying the login with different credentials, until the right combination is found.

3.2 False input

In order to act as an authorized user, an intruder may initially perform an authentication attack. The goal of an authentication attack is to skip the system authentication procedure in order to perform unauthorized operations, such as access to information or functionalities to which the attacker would not be entitled (e.g., user's privacy settings). This type of attack can lead to the possibility of performing false input, in which false information is input into the computer with the aim to cause a fraudulent output.

Such kind of attack can be carried out through diverse strategies, such as the previously mentioned eavesdropping or the brute force. Another, more refined, way to trick the authentication mechanism is the *code interpretation* of scripts within a Web page. When the decision of what information to display, according to user privileges, or even the authentication mechanism is delegated to the page scripts, an authentication attack can be performed by simply interpreting the scripts. Scripts interpretation, in this case, reveals the system communication logic. Any modifications can then be made to the page/scripts in order to skip the controls, and to access the system without any authentication.

False input can be even performed by skipping the authentication mechanism and directly acting as an authorized user. A similar strategy is known as a session management attack, often referred to as *session hijacking*, and aims to steal the user session identifier (usually sent as a cookie) in order to act as the user. Session hijacking is currently among the most common threats, and can be carried out by the following mechanisms: *Cross site scripting* (*XSS*), which is done by injecting malicious scripts into the navigated page with the aim to steal data (such as cookies); *Session sidejacking*, that consists in listening to the network packets in order to extract the session cookie(s) (similarly to eavesdropping); *Replay attack*, performed by sending copies of packets previously "stolen" from the network to the application server. Replay attack

is aimed to commit frauds (e.g., repeated transactions) by re-sending packets that contain the original cookies.

Session fixation is done by forcing a user to use a known identifier to query the system. This type of attack usually exploits the forwarding of the identifier as a URL parameter.

Systems typically perform controls on user privileges before allowing the access to functionalities and resources. Techniques aiming to trick such controls are often referred to as *authorization attacks*.

A malicious code fragment can be injected into a system by passing it as a parameter through some exposed method. This technique is known as code injection and can be extremely dangerous if the input parameter is not properly validated. A widely known code injection technique consists in passing a query string to the form for querying a database (e.g., a search form). Indeed, the code injection applied to the system database could return personal data of the system subscribers.

4. A CLIENT/SERVER ARCHITECTURE FOR WEB APPLICATION MIGRATION

The Migration Platform architecture considered in this paper allows any browser-enabled fixed or mobile device to access any interactive Web applications through a proxy, which injects on them some JavaScript code that makes them migratory. The proxy also annotates all the URLs within the page and its associated resources (e.g., CSS) in a way to convert them into absolute links under the domain of the Migration Server. In addition, any device involved in the migration environment must execute the Migration Client, which is a separate Web application able to provide information on the other devices available in the environment. The information is gathered and circulated through the execution of an Ajax-based discovery protocol. The Migration Client running on each active device periodically queries the Migration Platform server for the available information (e.g., list of possible target devices and their activities). At the same time, every device updates information about its situation to the Migration Platform. The Migration Client is also used to trigger the migration and select the target device. A Web application is launched from a client device through the Migration Proxy, which annotates it with JavaScript excerpts that are subsequently exploited to support migration. When migration is triggered by the user from the Migration Client, the DOM (Document Object Model) and the state of the interactive Web application are sent to the Migration Server, which updates the DOM with the whole interaction state and uploads it to the target device. The upload is carried out in two steps: an incoming migration message is first sent by the Migration Server to the Migration Client of the target device containing a reference to the target page URL within the Migration Server, then a new window/tab is activated with the target URL (which is relative to the Migration Server) so that the target page is shown.

In general, migration can be: single-user, when a mobile user decides to change device and continue to interact with the same application in the target device; multiuser, when an interactive application accessed by a user is moved (either through push by the user on the source device or through pull by the user on the target device) to a device accessed by another user. The considered Migration Platform architecture has been discussed in [6], however that work did not tackle security issues arising from application migration and how to address them.

In this type of architecture the risks of the pre-migration phase are mainly due to the proxy, which is an additional entity between the client and the original application server. The proxy-based navigation should then assure at least the same security level of the application server.

The post-migration phase implies maintaining the state of the interaction session on the destination device, which can be owned by same the user that triggered the migration or by another user (e.g. a friend or a colleague). Here the point is preventing other devices (inside or outside the Platform) from accessing such session.

5. MIGRATION PLATFORM SPECIFIC ISSUES

The potential security lacks of the Migration Platform are due to the way in which data exchange takes place internally and externally.

Figure 2 shows an overview of the double-side communications across the Platform. Those referred to as internal side issues are related to the communication between the migration server and the clients (e.g., Proxy navigation, cookies exchange, migration involving multiple users). The external side comprises instead the issues that spring from data exchange between the migration server and the servers that host the original applications (i.e. the ones to which the clients are accessing).

In the following, the main problems and the proposed solutions are discussed in detail.

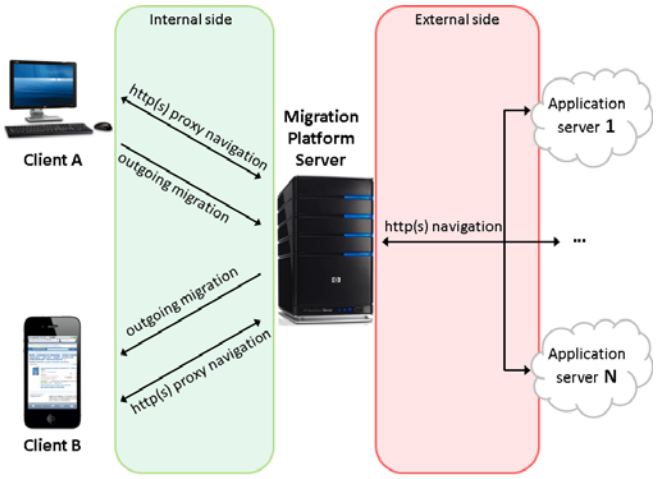

Fig. 2. Internal and external side communications across the Migration Platform.

5.1 Internal side security

5.1.1 Proxy navigation and security

An extreme strategy to provide navigation security would consist in always accessing the proxy via HTTPS. This solution might technically provide the highest security, but is considered to be

inefficient because the usage of HTTPS tends to decrease data communication rate, with an overall decreasing of the usability of the migration environment. A more recent investigation, discussed in [7], has proposed techniques for speeding up cryptographic algorithms. However, this solution is still at a preliminary stage for general adoption. In general, HTTPS still impacts on the performance and thus we have adopted a solution in which the proxy uses the HTTPS protocol at run-time according to the protocol used by the application server. The proxy is accessed via HTTPS whenever the application server should be accessed through it. For instance, a home banking Web site can be initially accessed via HTTP, since the main page is originally available via HTTP. As soon as the user logs in, then a HTTPS request is sent by the client browser to the proxy and, accordingly, this request is forwarded through HTTPS by the proxy to the application server, and likewise the answer from the application server is transmitted to the client through the proxy still in HTTPS protocol.

5.1.2 Cookies management and security

The proxy introduces some issues concerning the management of the application cookies. The application server considers the proxy as its client, but multiple users can concurrently access their applications through the proxy. Such users should thus be considered differently by the application server. A possible strategy consists in allowing the proxy to forward the cookies received from the Web application server to the client. The proxy annotates the name field of each cookie by appending the application server domain before forwarding it to the client. This is done in order to keep trace of the original cookie domain, so that the proxy is subsequently able to recognize which Web application each cookie belongs to. The client browser can then locally store each cookie, while the proxy does not keep any of them. For the client browser, all the updated cookies are actually referred to the domain of the proxy. Such a strategy is completely decentralized as the cookies are locally stored by the clients, and does not require any additional storage space on the proxy. However, the strategy is also highly inefficient because the client browser appends all the previously saved proxy cookies to every subsequent request to the proxy. It is worth noting that the Migration Platform allows any user to navigate several pages at the same time, and thus manages an arbitrary number of cookies. Within this strategy, most of the forwarded cookies are useless, as the proxy restores the original cookie names and only forwards the cookies belonging to the domain of the original application server.

An improved and more efficient mechanism for cookies management has thus been developed and is completely proxy-centered. The application cookies are not sent to the client but are instead saved in an entity, named *cookie store*, in the proxy. The cookie store keeps the cookies name/value/domain and the reference to the owner client (which is always a subscriber to the Migration Platform). Each client is bound to its previously stored application cookies by a proxy session cookie, which is a token generated by the proxy Web server and stored on the client browser. The header of every client request to the proxy contains the proxy session cookie. The proxy uses its session cookie as an index to the cookie store section of the user. In the following, we refer to the application server cookies as application cookies, and to the proxy session cookies as proxy cookies.

The theft of a proxy cookie would let an intruder access all the application cookies of the user. As a consequence, the intruder could exploit the application cookies to reproduce the user session interacting with the application server of banks, email, etc. A possible option to protect the proxy cookie would be to always exchange it via HTTPS. However, as already mentioned, this strategy impacts the navigation efficiency. Several pages can be navigated at the same time on the client, and such active pages would forward their requests to the proxy in a nondeterministic manner. Thus, the usage of a unique session id is also unsuitable for the proxy. We have excluded the possibility to use the client IP address as unique user identifier, since the device IP can change dynamically in some cases. The Migration Platform is indeed accessible by mobile devices, which often switch from one network to another (thus changing their address).

For the previously mentioned reasons, the navigation proxy cookie, which identifies the user session, has been chosen to be HttpOnly (i.e. it cannot be read by scripts) and constant, but additional controls are performed by the platform. For every request, the proxy checks whether the browser user-agent is consistent with the one stored at login time and, if not, the request is rejected. After this first check, in the event a resource is requested via HTTPS, the proxy performs an additional control on the JSECURE cookie set up at login time. Details on the JSECURE cookie generation are provided in the next section.

HTTPS navigation via proxy occurs, as previously stated, only when the application server requires the SSL usage (e.g., when private data are going to be exchanged). Thus, sensitive data can be safely exchanged via proxy.

5.1.3 Shared resources in migration

The possibility of sharing information and Web interactive applications among several devices, is a major feature of the Migration Platform. When a Web page is migrated, a copy of the page is created and stored on the Migration Platform server. As soon as the migrated page is available, the Migration Client of the target device is notified in order to open the page in the browser. A way for notifying the target device is to send back, via Ajax, the reference of the migrated page, which is actually the URL within the Migration Platform server. This solution was adopted by the first Migration Platform prototype, because it was considered to be simple and effective. In detail, the URL string was contained in the response of the Ajax request periodically performed by the Migration Client, which updates the device presence and checks for incoming migration pending. An initial security analysis revealed the risk of passing a URL unencrypted through an HTTP response: an attacker could get the URL by simply listening to the network communications, and then use it to access the migrated page on the Migration Platform server.

As previously discussed, the proxy navigation is considered to be secure. Indeed, when HTTPS connections are performed via proxy, the secure cookie created during user login is used. Thus, even if intruders got access to the migrated page, they would not be able to perform secure navigation, because they do not own the right secure cookie. However, the consequences of a similar attack lie in the possibility of the intruders to perform unauthorized operations or acquire sensitive data even from the single migrated page. It should be considered that the migrated page could contain the balance of a personal bank account. It is thus desirable for the Migration Platform to protect the Web pages during and after (as well as before) the migration process.

To protect the migrated pages from unauthorized access (i.e. from being accessed by unentitled users), we have developed a dedicated functionality that provides the target page content as response. In detail, instead of the migrated page URL, a special command is

passed back to the Migration Client. Such a command triggers the invocation of a special service, named loader, which loads the target HTML document and writes its content on the response. The Migration Client of the target device gets the response content and shows it on a new window. Such a procedure is carried out only after successful authentication of the client, as the loader service will actually provide the page content only if the JREG cookie of the client is consistent with the one related to the actual target. The target device is indeed registered and active (i.e. the user has already logged in). More details on the JREG cookie are provided in the next section.

5.1.4 Migration and HTTPS

Migration is usually performed via HTTP. The HTTPS protocol is used only when the user explicitly chooses to rely on the secure connection, or when the Platform automatically detects security implications on the page. As for the discovery protocol, the reason why we have chosen not to perform always via HTTPS lies on its performance limitations.

The automatic detection of security implications consists of checking whether: (1) the page protocol is HTTPS, (2) the document contains at least one form which "action" field is under HTTPS or (3) the document contains at least one input of type "password". The first condition reveals whether the page was originally considered to deal with sensitive data. The second and the third, instead, stem from the peculiarities of the Platform architecture, and in particular from the strategy of forwarding the page DOM and its state from the originating to the destination device. The serialization, in order to ensure state persistence, includes the values contained in form fields. If a page is under HTTP but contains a form which "action" is under HTTPS, then it will send the form content via HTTPS when the "submit" is triggered in order to protect the form field values (e.g., userID and password during a login). The Migration Platform, besides providing navigation (proxy) security, has also to assure the confidentiality of the values within the forms with HTTPS "action", since those values were originally designed to be exchanged via HTTPS.

If one or more of the three mentioned conditions are verified, then migration is performed via secure protocol. In this case, the DOM serialization and the state of the page are forwarded to the platform through an HTTPS POST. The target page is then opened via HTTPS and the values of the JavaScript variables are also loaded via HTTPS (this is because sensitive data might be contained by script variables which have been "frozen" at migration time).

It is worth pointing out that the user can in any case require that the secure protocol is used for migration, just checking the associated option when the migration is triggered. This is because, even if the page is under non-secure protocol and has not HTTPS forms, it could contain information that the user might not want to be captured (e.g., parameters for advanced search forms, results of an online item search, or anything s/he considers as confidential), and this is particularly true for dynamic pages.

5.1.5 State persistence and security

A requirement of the migration process is the state persistence: when the migrated page is opened in the target device, the result of the interactions performed in the source device has to be maintained. The page state includes form fields content, JavaScript variables value and the cookies related to the Web page domain. The problem is that users can change device and still would like to benefit from the associated cookies for the current application.

As already discussed, although the application cookies are not stored in the client devices, the user can indirectly access them through the Migration Platform. The proxy is indeed aware of the correspondence between the user and the original cookie(s) value for the requested domain. When a migration is triggered, the external cookies of the source internal session are bound to the target internal session. This is implemented by creating a reference, in the cookie store, from the internal session of the target device to the internal session of the source device.

From the point of view of the users involved in the migration process, two cases can occur: the single user and the multiuser. Security drawbacks could arise from totally copying the cookies in the multiuser migration, when the application can migrate to a device owned by another user. Thus, in this way, the target device can access all the application cookies owned by the user of the source device. In detail, the user accessing the target device not only accesses the application cookies related to the migrated Web page, but all the cookies that were bound to the user accessing the source device since the beginning of the session.

There is more than one possible alternative solution to manage source cookies in multiuser migration, according to the level of protection. An option provides the target with all the cookies within the domain of the migrated page. An example scenario involves the subscriber of some service (user A) that needs to temporarily allow a friend/colleague (user B) to access the service. User A performs authentication on the application and migrates the resulting page to the device of user B. It is then possible, for user B, to navigate throughout the pages of the service, since s/he has got all the application cookies for that domain. User B would not be entitled to modify user A credentials, as s/he would be requested to insert the original ones (which s/he does not know). A more restrictive option consists on providing only the cookies related to the current path of the domain to the target device. This would be aimed to allow the target user to exploit functionalities or access information restricted to the page(s) in the path of the source one. The main limitation of this option is that the cookies of common Web applications usually do not define the "path" field. An extreme possibility is not to copy any cookie.

We adopted a solution where the privacy level is specified at migration time by the user. The user can choose to share with the target user(s): i) all cookies, ii) only those related to the domain, iii) only the domain ones specific to the current page path, iv) none.

5.2 External side security

The security issues in the communication between the platform proxy and the Web application servers were the first ones to be tackled. It was initially clear that the proxy had to be able to perform secure connections in compliance with the applications requirements. Indeed, Web applications often exchange data with clients via the HTTPS protocol, even if their domains are usually accessed via HTTP. This is done by the application server in order to protect sensitive user data against third party sniffing. For instance, the user login credentials as well as credit card details are typically sent via HTTPS.

The proxy of the Migration Platform is able to connect to any server via HTTPS, as well as via HTTP. The connection is man-

aged by a library that fully implements all HTTP methods and supports encryption with HTTPS protocol [11]. In practice, the level of security in the external communication is determined by that requested by the application.

The cookie transfer from the proxy to the application server is another aspect that involves the external side communication security. However, as previously discussed the Platform is able to keep the correspondence between users and application cookies. Thus, given a Website X for which user A has got a cookie (which is saved in the cookie store of the Platform), another user B would in general not be able to exploit that cookie. The only situation that implies cookies sharing is the multi-user migration, where the originating user explicitly chooses to copy her cookies into the session of the destination user.

6. OTHER SECURITY ISSUES OF THE MIGRATION PLATFORM

The main Migration Platform operations, such as login, profile update, migration request, are explicitly performed by the user through the Migration Client. Other operations, such as the exchange of device discovery messages (e.g., user/device presence) with the platform, are periodically and automatically done by the Migration Client.

The aim of the login is to protect user personal data, and information and resources of other subscribers who are in relationship with the user (and to which intruders would not be entitled to access otherwise). It would be possible for anyone, especially in public networks, to sniff the packets exchanged between the Migration Client and the login servlet. Thus, if the data were exchanged “in clear”, the login credentials could be stolen. This is why the Migration Platform exploits the Secure Socket Layer (SSL) of the HTTPS protocol during the main user operations, so as to reduce the risk of eavesdropping.

The login on the platform is carried out via HTTPS, thus protecting the user credentials against sniffing. However, as discussed in the previous sections, intruders may also exploit session sidejacking by listening to the network packets and extracting the session cookies from the headers. A possible way to overcome this problem could be to perform all the operations via HTTPS. However, differently from the login operation (which is performed once per session), the device discovery protocol operations are repeatedly performed in real-time. The discovery basically consists on sending periodical “update” requests to the Platform. The Migration Client of each logged device announces its state every few seconds. As a consequence, the solution of using the HTTPS protocol for device discovery is technically feasible but would have heavy drawbacks on efficiency.

Although the discovery is performed through non-secure protocol (HTTP), we have defined mechanisms for controlling the access to the Platform functionalities. Regarding the operations allowed to registered users, we have opted for a non-secure and unique session cookie, referred to as JREG. The uniqueness of JREG is aimed at improving the security and refers to the possibility of using the cookie only once. Thus, an intruder is very unlikely to capture the cookie and reuse it to act as the owner user since a second usage of the unique JREG cookie would not be allowed by the Migration Platform. The uniqueness is obtained by appending an increasing timestamp to the cookie. This is achieved by the Platform that, upon user login, calculates the JREG by concatenating the deviceID and the output of the HMAC (Hash-based Message Authentication Code) function. The usage of a hash encoded function is necessary to assure the following features: confidentiality, since data sent are private and thus it is better not to send them unencrypted; integrity, the hash function applied to the cookie returns a different value if the cookie is modified by third parties. The deviceID modification is an example of possible attack attempt; uniqueness, only one encoded string refers to a specific character sequence.

A similar strategy for improving cookie security is also reported in [9]. It is worth pointing out that there is the possibility for an intruder to use the stolen cookie before the owner. As a consequence, if the cookie were used for authentication the system would reject the access of the real user and would authenticate the intruder. In order to limit such risk, the HttpOnly attribute is set to the JREG cookie, thereby making it inaccessible to malicious scripts. This solution avoids the cookie theft by means of Cross-site scripting (XSS) attacks

An additional strategy, aimed to limit the consequences of cookie theft, exploits a complementary cookie named JSECURE. The Migration Platform login servlet, at login time, generates the JSECURE cookie with secure and HttpOnly attributes set as true. In addition, the secure attribute allows the cookie to be exchanged only via HTTPS (thus providing additional protection against thefts). Operations involving the exchange of personal user data are always performed via HTTPS (e.g., profile update), and for each of those sensitive operations, the system checks the consistency of the incoming JSECURE cookie. Thanks to such a check, the intruder is unable to perform any malicious operation because s/he does not know the actual JSECURE value. The real user would instead be able to re-login at any time, gaining full access to the system.

In order to achieve uniqueness and unpredictability, the session cookies are generated as message digests. The SHA-256 algorithm (from the javax.crypto package) is applied to a random number generated through SHA1PRNG algorithm of the Java SecureRandom class (from the java.security package).The JSECURE cookie instead, being secure and HttpOnly, is highly protected against thefts, thus the intruder is unlikely to know it.

The above mentioned strategies, as already stated, do not exclude the possibility of attacks but limit them and their consequences.

A protection against the brute force attack is also achieved by limiting the number of access attempts that can be performed within a specified amount of time.

The proxy servlet for navigation could be affected by code injection if an inconsistent URL were passed as a parameter of the GET/POST method with the aim to exploit some control weakness. For instance, if a bad string is passed as URL to the proxy and the exception stack is printed (e.g., in the error page), the attacker can even obtain information about the system source code. In the current prototype, the proxy is protected against code injection by enhanced robust controls on the input URL and by safe handling of the server-side exceptions (stack trace printing is disabled). Regarding code injection within other user operations (e.g., profile update), such risk is implicitly avoided as the subscribers’ data are stored in an XML database (which is not subject to code injection).

If authentication controls were performed client-side, i.e. by the JavaScript, a potential script interpretation attack against the Migration Platform could be done by creating a modified version

of the Migration Client. The modified Migration Client would skip the authentication controls with the aim to perform forbidden operations or to get sensitive information (e.g., monitoring the activity of other users, for which the intruder would not have the privileges). Such an intrusion, aimed to trick authorization controls, would lead to a series of authorization attacks: the intruder could be able to interfere with the user device functionalities by even overriding the controls on incoming migration privileges specified by a user. Thus, the intruder could migrate a malicious Web page towards the user device. However, such risks are avoided in the Migration Platform because all the authentication and, consequently, privilege checks are performed server-side.

The Migration Platform does not allow a session id (e.g., a session cookie) to be passed through URL, thus session fixation attacks do not concretely threat the Platform.

7. AN EXAMPLE OF APPLICATION OF SECURE WEB MIGRATION

Personal information is often contained, in different ways, on a Web page that is being migrated. Such data can lie in the document source code, such as in the shopping cart summary page, or can have been explicitly entered by the user during the interaction with the page.

As mentioned in Section 4, migration is carried out by serializing the Document Object Model (DOM) available in the source device browser. The DOM actually includes the values of the form fields filled in by the user. In order to understand how this affected the security of the preliminary Migration Platform, it is sufficient to consider a simple scenario: the user is creating a personal account on a social network, through non-secure connection (see Figure 3), while s/he decides to trigger migration towards another personal device. The migrated page contains the partially filled form, including the personal email and password. Thus, although the form action value is "secure" (i.e. an HTTPS address), sensitive data would have been exchanged without any encryption between the client device and the Migration Platform. This would happen because the form page was accessed through HTTP, even if the form submission would have forced the browser to switch to a HTTPS connection.

Figure 3. An example of registration form filled with personal data within a page under HTTP.

In the security-enhanced version of the Platform, at migration time, the Migration Platform is able to automatically detect the presence of sensitive data and to act properly. If the page is navigated through HTTPS, then migration is performed through HTTPS. If the page is navigated through HTTP, a search for secure forms is done within the document. If the page has one or more secure forms and if at least one input field is filled, and/or if it contains an input of type "password", then a secure migration is performed.

The security control on the migrated page is performed when migration is carried out by the scripts injected in by the proxy at navigation time. The motivation of distinguishing among secure and non-secure migration lies in the performance: secure migration is indeed more time-consuming, thus it is reasonable to migrate via HTTPS only in the case of security concerns.

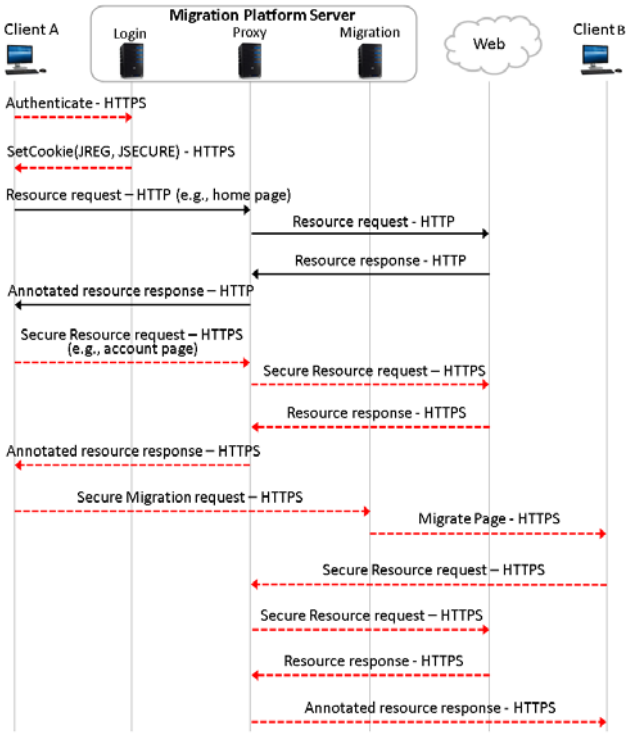

Figure 4. A sequence diagram showing the main communications involved in secure migration.

Figure 4 summarizes the main communications that take place when a Web application is navigated via proxy through HTTPS, and when it is migrated. Black continuous arrows indicate HTTP communications, while red dotted arrows refer to HTTPS ones. Client A initially authenticates in the Migration Platform and requests the Website home page. While performing login in the home page, the user is redirected to a HTTPS address and continues navigating the personal pages via proxy. Upon request, the platform performs a secure migration, thus acquiring the source DOM from the source device and forwarding it to the target device (Client B) through HTTPS. It is assumed that Client B had already performed authentication, in the same way as described for Client A. The user then continues the navigation through the target device, still via proxy within HTTPS connection.

A technological issue, known as *cross-domain exception* arose when a secure migration was triggered for a non-secure page. The reason is in the different domains between the page and the secure address of the migration servlet (HTTPS protocol, e.g., 8443 port): the page is indeed accessed via the conventional platform address (HTTP protocol, e.g., 8080 port). In order to solve this issue, we have enabled the CORS (Cross-Origin Resource Sharing) support [4] in the Migration Platform. CORS is a W3C standard specification for authorizing a browser window to exchange messages (e.g., through Ajax) with different domains.

8. VALIDATION OF SECURE MIGRATION

In order to validate the secure migration, we have carried out two tests. A first one was aimed to check that the Platform is able to provide interaction continuity when accessing pages migrated via HTTPS. A second test was devoted to quantify the overhead of a migration performed via HTTPS with respect to HTTP.

8.1 Preliminary tests

The aim of the initial tests was to ensure that the Platform provides interaction continuity, i.e. full state persistence. The continuity is assured only if the results of user interaction are preserved, and these include content of filled forms, session and other cookies. With this first test session we thus checked that, after migrating a Web page via secure protocol, the user was actually able to continue the navigation from the target device without losing the original session (i.e. without having, for instance, to re-log in the Web application or to re-add the items to the shopping cart).

We considered some of the most visited Web pages (according to the ranking of http://www.alexa.com) and tested the Migration Platform on them, with the aim of validating the secure support. The following list summarizes the operations performed on the tested Web applications:

- *google.com (Gmail)*: the login page and the email composition on the user personal page were migrated, preserving access credential, cookies and email form content (email title and body).
- *eBay.com*: the account creation page was migrated, and the user data (such as name, surname, username, password) were maintained.
- *amazon.com*: the shopping cart content page was migrated with cookies persistence (selected items were maintained).
- *wordpress.com, youtube.com*: the login page was migrated preserving user access credentials.
- *paypal.com*: the form for sending a payment was migrated, and the cookie as well as the form field content were maintained (payment amount and beneficiary).
- *facebook.com*: the login page was migrated with the user credentials form content.
- *ryanair.com*: the flight reservation form, including user personal details (name, surname and address) and credit card data, was migrated.

After each Web migration, navigation was performed in the target device, in order to verify the state persistence. Although many of the tested pages were in the cross-domain situation (i.e. pages in non-secure address but migration performed via HTTPS) the migration worked in all cases thanks to the use of the CORS support.

8.2 Performance tests

The main aim of this test session was to have an indication of the cost in terms of additional time, in the case the user chooses to perform a migration via HTTPS rather than HTTP. In this case we only consider the HTTP REQUEST through the post METHOD, which is carried out for the transmission of the DOM and the associated state.

Five international well-known Websites were chosen for quantifying the impact of the secure protocol on such migration performance. The pages considered were originally accessible via HTTP. It is worth noting that it would not have had sense to consider HTTPS pages in the test, since in that case the Platform would have automatically performed migration via HTTPS (i.e. the user is not allowed migrate a secure page in non-secure manner). The test took place in laboratory, the source device was a desktop PC (with wired connection) and the target was a laptop (with wireless connection).

Table 1 summarises the results of the technical test. For each input page, migration was tested 10 times in HTTP and 10 times in HTTPS. The average times for the two migration modalities and the average difference are reported in Table 1. The time measurement started when migration was triggered on the source device and ended when the redirect message was sent to the target device.

Input page			**Average time (s)**		
Domain	**Description**	**Size (KB)**	**HTTP**	**HTTPS**	**Overhead (%)**
Venere.com	Search hotels – 15 results	173	1,072	1,157	7,9
Bbc.co.uk	Business homepage	202	0,717	0,774	7,9
Ebay.com	50 items search results	478	1,524	1,550	1,7
Amazon.com	Search books – 12 results	550	1,534	1,638	6,8
Google.com	advanced search - 100 results	601	3,105	3,576	15,2
Mean		400	1,6	1,7	7,9
Standard Deviation		200	0,9	1,1	4,8

Table 1. Summary of the technical test results.

According to the tests results, the migration time varied between about 0,7 and 3,6 seconds, according to the page and the protocol used. As expected, the HTTP was always faster. However, the difference was not so high as the HTTPS overhead varied between 1,7% and 15,2% and was, on average, less than 8%. Thus,

in the case a user feels that a page under HTTP contains confidential data and chooses to perform migration via HTTPS, the impact of the secure protocol on the overall time is reasonably low.

9. CONCLUSIONS AND FUTURE WORK

We have presented an analysis and discussion of the security issues involved in client-server support for migration of interactive Web application and how they can be addressed. We have also reported on how such results were applied to an existing proxy-based migration platform, which did not address them previously. In addition, the techniques that we have proposed can also be applied to other platforms that are not specifically aimed at migrating Web applications but are characterised by a similar proxy-based architecture.

The results have also been validated by testing various widely known Web applications to check that they effectively migrate with state persistence while exploiting the security support.

Future work will be dedicated to further empirical usability evaluation of the security mechanisms introduced in the Migration Platform.

Alternative architectures for the described Migration Platform, such as cloud-based architectures, will be considered. For instance, the Platform Proxy could be transparently replicated by delegating its management to a third party (i.e. a provider). On the one hand, this strategy can lead to better scalability of the performance, as there could even be one proxy instance for each Platform user. On the other hand, security might be improved as well, since if a proxy were successfully attacked, the intrusion consequences would affect only the “owner” user of that single proxy.

10. REFERENCES

[1] Alt, F., Kubitza, T., Bial, D., et al. Digifieds: insights into deploying digital public notice areas in the wild. Proceedings of MUM’11, ACM Press, pp. 165-174.

[2] Arthur, R., Olsen, D.R. Privacy-aware shared UI toolkit for nomadic environments. Software – Practice and Experience, 2011, 42:601-628..

[3] Chang, T.-H., Li, Y. Deep Shot: a framework for migrating tasks across devices using mobile phone cameras. Proceedings of CHI 2011, ACM, pp. 2163-2172.

[4] Cross-Origin Resource Sharing. W3C Working Draft 27 July 2010. http://www.w3.org/TR/cors/.

[5] Ge, X., Paige, R.F., Polack, F.A.C., Chivers, H., Brooke, P.J.: Agile development of secure web applications. Proceedings of ICWE ‘06, ACM, pp. 305-312.

[6] Ghiani, G., Paternò, F. Santoro, C. Push and Pull of Web User Interfaces in Multi-Device Environments. Proceedings of AVI 2012, ACM New York, pp. 10-17.

[7] Kounavis, M., Kang, X., Grewal, K., Eszenyi, M., Gueron, S., Durham, D. Encrypting the internet. SIGCOMM ‘10, ACM, 2010, pp. 135-146.

[8] Li, J., Jia, Y., Liu, L., Woa, T. CyberLiveApp: A secure sharing and migration approach for live virtual desktop applications in a cloud environmentFuture Generation Computer Systems, August 2011, Elsevier.

[9] Liu, A. X., Kovacs, J., Huang, C.-T., Gouda, M. G. A Secure Cookie Protocol for HTTP. Proceedings of ICCCN 2005, IEEE, pp. 333-338.

[10] Maurer, M., De Luca, A., Stockinger, T. Shining Chrome: Using Web Browser Personas to Enhance SSL Certificate Visualization. INTERACT (4) 2011, pp. 44-51

[11] Org.apache.http.client library, http://hc.apache.org/httpcomponents-client-ga/

[12] Satoh, F., Tokuda, T. Security Policy Composition for Composite Services. Proceedings of ICWE '08, IEEE, pp. 86-97.

[13] Sharp, R., Madhavapeddy, A., Want, R., Pering, T. Enhancing Web browsing security on public terminals using mobile composition. Proceedings of MobiSys 2008, ACM, pp. 94-105.

[14] Sharp, R., Scott, R., Beresford, A.R. Secure Mobile Computing via Public Terminals. Proceedings of Pervasive 2006, Springer-Verlag Berlin, pp. 238-253.

[15] The New Multi-screen World: Understanding Cross-Platform Consumer Behavior. Google Research Report, 2012. http://services.google.com/fh/files/misc/multiscreenworld_final.pdf

[16] Yu, W., Li, J., Hu, C., Zhong, L. Muse: A Multimedia Streaming Enabled Remote Interactivity System for Mobile Devices. Proceedings of MUM’11, ACM, pp. 216-225.