SessionJuggler: Secure Web Login From an Untrusted Terminal Using Session Hijacking

Elie Bursztein
Stanford University
elie@cs.stanford.edu

Chinmay Soman
Stanford University
cpsoman@stanford.edu

Dan Boneh
Stanford University
dabo@cs.stanford.edu

John C. Mitchell
Stanford University
jcm@cs.stanford.edu

ABSTRACT

We use modern features of web browsers to develop a secure login system from an untrusted terminal. The system, called *Session Juggler*, requires no server-side changes and no special software on the terminal beyond a modern web browser. This important property makes adoption much easier than with previous proposals. With *Session Juggler* users never enter their long term credential on the untrusted terminal. Instead, users log in to a web site using a smartphone app and then transfer the entire session, including cookies and all other session state, to the untrusted terminal. We show that *Session Juggler* works on all the Alexa top 100 sites except eight. Of those eight, five failures were due to the site enforcing IP session binding. We also show that *Session Juggler* works flawlessly with Facebook connect. Beyond login, *Session Juggler* also provides a secure logout mechanism where the trusted phone is used to kill the session. To validate the session juggling concept we conducted a number of web site surveys that are of independent interest. First, we survey how web sites bind a session token to a specific device and show that most use fairly basic techniques that are easily defeated. Second, we survey how web sites handle logout and show that many popular sites surprisingly do not properly handle logout requests.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

Keywords

Mobile, session hijacking, secure login, cookies

1. INTRODUCTION

It is well known that password authentication is vulnerable to malware on the client's computer and that users logging in from untrusted machines, such as a friend's computer, an airport terminal, or other public access machines, are especially at risk [23]. Moreover, we show that many of the Alexa Top 100 sites do not use HTTPS on their login pages, further putting users at risk by sending passwords in the clear. Among the risks encountered by users of an insecure terminal and WiFi network, password theft has arguably the greatest consequence on users, since stolen passwords have long period of validity, are difficult to detect as stolen, and are commonly applicable across multiple site.

WWW 2012, April 16–20, 2012, Lyon, France.
ACM 978-1-4503-1229-5/12/04.

To address this issue many elegant proposals have tried to improve user authentication by using a smartphone as a security token [6,7,27,29,30,34]. All these approaches, however, require either client-side or server-side changes. Prior to our work no solution provided a universal mechanism that supports secure login to every website from any untrusted terminal using an off the shelf phone.

We also mention that Google provides a clever solution where users, prior to going on a trip, request a number of one-time passwords that they then use to login at public terminals. Alternatively, the Google authenticator application can provide one-time passwords on the fly. With this system, a password stolen by malware is of no use to the attacker. While Google's solution works well for Google, it does not help the user login to other sites who do not implement this system.

In this work, our goal is to improve user authentication for all web sites and therefore we propose a solution that requires no server-side changes and uses an unmodified terminal. We only require that the terminal run a modern web browser with the ability to install a Javascript bookmarklet.

Our contributions. We demonstrate an architecture called *Session Juggler* (`http://sessionjuggler.net`) that enables users to login without ever entering their long term credentials on the terminal. In essence *Session Juggler* works as follows: the user first navigates the terminal to the desired web site. The resulting URL is then transferred to the phone and the user is asked to login to the site from her phone. Once the login phase is completed the entire session state is deliberately "hijacked" and copied from the phone to the insecure terminal.

Many solutions, including Google's one-time passwords, focus on protecting the user's long-term credential, but do little to prevent malware from hijacking the session after login. Similarly, sites that allow browsing over HTTP do little to prevent session hijacking by a network sniffer [9]. The reasoning is that a session hijack gives the attacker a short-term session token rather than a long term password and this short-term token is revoked as soon as the user logs out. Therefore, session hijacking is a lesser concern. One difficulty with this reasoning is that an infected terminal may replace the logout button by a "no-op." The user will be fooled into thinking that the session was terminated, where in fact, the attacker is free to continue abusing the user's account. Similarly, for cleartext sessions, an attacker can simply block the logout request.

Session Juggler mitigates this threat by providing a *secure logout*: because the phone and the untrusted terminal share the same session data, malware cannot prevent the user from logging out through the phone app. When the user logs out via the phone, the website logout process should, in theory, invalidate all session state associated with the session, thus invalidating all session state stored on the untrusted terminal.

Analyzing session management in the wild. To validate the session juggling concept we begin with a number of studies of web applications in the wild that are of independent interest. Our first study shows that *75%* of the Alexa Top 1000 sites still do not use HTTPS on their login pages and transmit user passwords in the clear. Users at Internet Cafes are consequently vulnerable to password theft. Since users heavily reuse password across sites [16], an attacker who learns the user password at one of these insecure sites can easily attack the user at other sites that implement proper login.

Session Juggler can help protect users on sites that use HTTP login. Users enter their long term credential on the phone which is then transmitted via 3G over the air. The session is then transferred to the user's laptop and continues over HTTP. Since 3G data is encrypted over the air, the user's credential is protected from snooping at the Cafe. When the phone is connected on a WiFi network *Session Juggler* requires the user to click through a warning before sending the credentials over HTTP.

Since *Session Juggler* transfers the session from one machine (the phone) to another (the user's laptop or terminal), we may inadvertently trigger the anti-hijacking defense at the site and thereby invalidate the session. Our second study verifies the effectiveness of *Session Juggler* by manually analyzing how popular web sites bind sessions to devices. In Section 2.2 we show that only a few sites bind the user session to specific browser characteristics, such as an IP address (e.g. Apple and Amazon) or the local computer time (e.g. eBay) [14]. Our evaluation shows that *Session Juggler* works on all but eight of the 64 Alexa Top 100 websites that offer a login mechanism [1]. Two of these failures were due to a bug that makes the Android webview unable to render those sites. The last failure was due to an aggressive browser fingerprinting that our prototype currently does not handle. Moreover *Session Juggler* works flawlessly with Facebook connect which ensures that it can currently be used on over 85000 web sites [36].

Our third study looks at how web sites handle logout requests. As discussed above, secure logout limits the window of opportunity that an attacker has to use a hijacked session token. Once the user suspects that hijacking is taking place he or she can use *Session Juggler*'s secure logout mechanism and the session is invalidated. To ensure that *Session Juggler*'s secure logout effectively protects users we tested how web sites handle logout requests. To our surprise we found that many popular websites including *Linkedin*, *Blogspot*, *IMDB*, *CNN*, *MSN*, *eBay* and *Yahoo!*, do not properly invalidate session tokens on logout (see Table 2 for a complete list). These sites instruct the browser to delete the cookie client-side, but do not invalidate the session server-side. As a result, even a passive attacker who eavesdrops on HTTP traffic (e.g. Firesheep) can continue to transact on the user's account after the user clicks logout. *Session Juggler*'s secure logout mechanism only works on websites that enforce logout request on the server side. We discuss this issue in Section 2.3.

[1]Some sites such as Google appear multiple times in the top 100 hence the "low" number of sites tested.

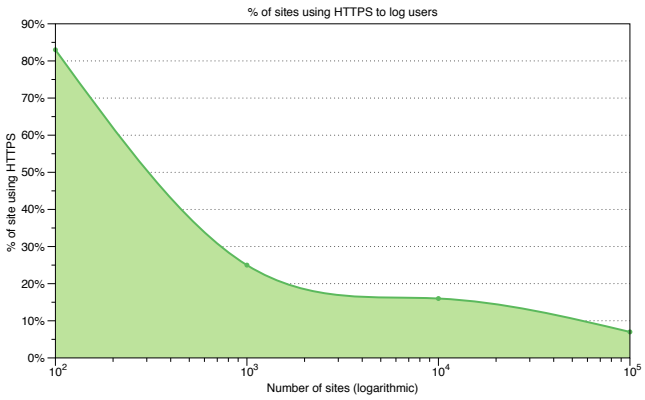

Figure 1: Fraction of sites using HTTPS login as a function of popularity. Sites listed from Alexa top 100,000

2. SESSION MANAGEMENT IN THE WILD

We begin with three studies that examine specific features of session management at popular web sites. These studies expose several widespread vulnerabilities, such as improper handling of logout requests. While running the studies we discovered that Google and Microsoft did not properly handle logout requests on their health records services (Google Health and Microsoft HealthVault). Both organizations quickly fixed the problem and acknowledged our findings in their hall of fame [19, 28].

2.1 Secure login pages

Our first study examines the number of sites that do not use HTTPS on the login page. Passwords at those sites are transmitted in the clear and can be easily sniffed at open WiFi hotspots such as Internet Cafes. Since people tend to reuse passwords across many sites, an HTTP login page may put all of the user's accounts at risk. *Session Juggler* helps mitigate these issues by transmitting user credential over 3G rather than the WiFI. Default 3G encryption makes sniffing harder.

To measure the number of sites that use cleartext login we inspected the HTTP login pages at the Alexa top 100,000 sites an look for sites that post the credentials over HTTP. The results are summarized in Figure 1. The horizontal axis orders site by Alexa popularity and the vertical axis shows the fraction of sites of that popularity that use HTTPS login. For example, only 25% of the Alexa top 1000 use HTTPS and the situation is worse after this: only 7% of the Alexa top 100,000 use HTTPS login.

2.2 Binding sessions to devices

Our second study examines how web sites bind sessions to devices. Recall that *Session Juggler* transfers sessions from one device (phone) to another (terminal or laptop). To ensure that *Session Juggler* will work properly with existing sites we had to determine whether web sites bind session tokens to client devices to prevent session hijacking. Recall that Panopticlick [14] is a system that is able to uniquely identify browsers. The question is whether web sites use Panopticlick-like techniques to monitor when sessions move from the originating browser to a new and unknown browser. Such a move could indicate a session hijack. We note that session migration may legitimately take place due to browser sync, where the user moves a session from a home computer to an office computer, but in these cases both machines will already be known to the web site and this would not trigger the site's hijack defenses.

To study how web sites bind sessions to devices we created accounts and manually logged in to all 64 Alexa Top 100 websites that have a login facility (we used Firefox under Windows). Once logged in, we clobbered various variables used by Panopticlick and then reconnected to the site to test if the session remained active. If not then the site was using the clobbered variable to bind to the device. To test session binding to IP address we used two different networks that had different address classes but belonged to the same AS. The results are summarized in Table 1.

Defense	% of Alexa100
Using HTTPS	83%
Using secure cookies	52%
Separating mobile and desktop sessions	6%
Binding session to IP address	8%
Checking local time	1%
Binding session to `user-agent` header	0%
Binding session to local language	0%
Logout over HTTPS	1%

Table 1: Anti-hijacking defenses at the Alexa top 100 sites

Of all the web sites we tested only `gmail.com` did not revert back to cleartext HTTP after the login page. This issue received considerable attention recently [10, 33] thanks to automated tools released last year. As a result, *Facebook*, *Twitter*, and *LinkedIn* now offer a full HTTPS version of their sites, but it is not enabled by default. More worrisome is the fact that only a fraction of the websites that use HTTPS use secure cookies (**83%** vs **52%**), which leaves them totally open to session hijacking by a network eavesdropper.

An even smaller fraction (**6%**) of websites separate desktop sessions from mobile sessions. That is, for most sites session tokens generated on a desktop can be used to transact on the mobile site and vice versa. On these sites one can transfer a session from a mobile device to a desktop without invalidating the session. For 6% of the sites, however, the session would be killed and therefore, *Session Juggler*'s phone application emulates a desktop browser so that the session transfer appears to be from desktop to desktop.

Surprisingly our survey reveals that only **9** websites out of the 64 most visited websites use non-basic session binding and only **5** of them use the client IP as the binding parameter (including *Amazon* and *Apple*). We discuss in which case *Session Juggler* works on websites that perform IP binding in section 5. Only *eBay* uses the local time on the client's machine for session binding. To use *Session Juggler* on a site like *eBay* we had to ensure that *Session Juggler*'s phone application emulates the desktop's local time. None of the websites we tested use more sophisticated browser fingerprints (user agent, local language, plugins) which we found surprising.

2.3 Logout procedures

Our third study examines the logout procedure at popular sites. We thought it was obvious that when a user clicks logout web sites should invalidate the active session server-side. To our surprise we found that many popular web sites remove the session cookie from the browser, but *do not invalidate the session on the server*. Consequently, an attacker who somehow obtains the user's session token can continue to transact on behalf of the user after logout.

This has two consequences:

1. Web sites that do login over HTTPS but then revert to cleartext HTTP can leak the session token to a network eavesdropper, as in the Firesheep attack [9]. With improper logout the eavesdropper can continue to transact even after the user explicitly logs out. The user has no way to invalidate the session token even if he or she suspect that the session was hijacked.

2. Consider a secure web site that operates entirely over HTTPS and requires a second factor to login, as is often the case in healthcare settings. If the site implements improper logout then malware on the client can steal the session token and continue to transact after the doctor or nurse logs out of the terminal. In effect, the session token becomes a *single factor* credential that cannot be revoked by the user.

As mentioned above, many web sites implement improper logout and this is a wide spread bug. Some web sites, including Twitter and Amazon, default to a partial logout state after the user clicks logout. In this state websites still accept the user's session token and provide basic information, but request re-authentication for sensitive actions. Our findings, listed in Table 2, show websites that allow users to perform sensitive operations after the user clicks logout. This listing summarizes all the websites from Alexa-top 100 that are not honoring logout requests on the server side. We also tested popular open-source software, such as *Wikimedia*, *PHPBB*, *Wordpress*, *Drupal*, *Cake*, but all implemented logout correctly. We also found that Google and Microsoft health record system where subject to logout issue. Finally we analyzed the following four open-source popular health record systems: OpenMR, FreeMed, ClearHealth, OpenClinic. Of these four, only OpenMR was subject to the logout issue.

The most interesting case we encountered was *Google*: while Google properly implements logout for core services such as GMail and Reader, many "side" services including *Youtube*, *Blogger*, *Orkut*, and, *Health* do not. The reason is that each of these services retained their custom session management system and added the standard Google authentication system as a secondary mechanism. This double authentication leads to problematic logout issues. For example, after logout the session token could still be used to edit the user's blog posts on Blogger but could not be used to access Gmail data.

Another interesting case is *Twitter*. As discussed earlier, *Twitter*'s mobile site is different from its main site. The two sites have different session management systems: while the main Twitter site correctly implements logout requests, the mobile site does not. This is another example in a general theme showing that mobile sites have weaker security than their desktop counterparts [32].

Is poor logout management a security vulnerability? For sites that login over HTTPS but then move to HTTP, poor logout is a significant risk. For other sites there is no clear position on this as was apparent from our discussions with affected companies. Google and Microsoft treated their respective health service logout issue as a vulnerability and quickly fixed the problem. As far as we know, Microsoft did not fix the Hotmail issue. Other companies who responded to us acknowledged the issue, but said that it was not a priority because of the lack of attack surface and the heavy cost of changing their session management infrastructure. From the *Session Juggler* perspective, logout issues are vulnerabilities because it prevents *Session Juggler* secure logout functionality to work properly.

Site	Sensitive Action Allowed After Logout
health.google.com	View and edit health record
healthvault.com	View and edit health record
Linkedin	Editing and saving profile
Yahoo	Accessing and sending emails
Hotmail/MSN	Accessing and sending emails
Blogger.com	Posting a blog post
Ebay	Bidding on an auction
Flickr	Uploading photos
wordpress.con	Posting a blog post
IMDB	Editing and saving profile
ask.com	Editing and saving profile
cnn.com	Editing and saving profile
conduit.com	Editing and saving profile
megaupload.com	Uploading files
mediafire.com	Uploading files
4shared.com	Uploading files
cnet.com	Editing and saving profile
weather.com	Editing and saving profile
imageshack.us	Uploading photos
OpenMR	Accessing, changing medical records

Table 2: Sites with improper logout

3. JUGGLING

We now describe how *Session Juggler* works. We start by describing step by step how the session transfers occur from the user's perspective. Then we describe in detail how the juggling protocol works and how its participants are implemented. Finally we review the security benefits of using *Session Juggler*.

3.1 Threat Model

The goals of *Session Juggler* are two fold. First, it ensures that the attacker cannot learn the the user's long term credential. In particular, *Session Juggler* mitigates the risk of a man-in-the-middle attack against websites that transmit user login credentials in the clear. *Session Juggler* does so by transmitting the user's credentials over a 3G connection that is encrypted and harder to intercept. Second, *Session Juggler* provides a trusted logout to invalidate an ongoing session for sites that securely implement logout.

Session Juggler does not protect against post-login attacks other than by providing a trusted logout. It should be noted that since many web sites only use HTTPS for the login page and then drop back to HTTP, session hijacking is currently easily done by a network attacker using standard tools [10] without the need for client-side malware. Often users have no recourse against a session hijack, even if they are aware that their session was hijacked — clicking "logout" can be intercepted by a network attacker and blocked (on sites we tested that fall back to HTTP the logout action was also over HTTP). In contrast, our secure logout will immediately invalidate the session once the user detects the attack.

3.2 User Experience

A storyboard of the user experience with *Session Juggler* is shown in figure 2[2]. As shown there, users login as follows:

1. **Initiate the Juggling**: First the user navigates on the untrusted terminal to the site she wishes to log on. In our example our user goes to `http://www.facebook.com`. Then instead of entering her credentials on the untrusted terminal, she clicks on the Session Juggler bookmarklet. In the figure 2, the bookmarklet button is located on the bookmark bar and is highlighted in green. Clicking the bookmarklet creates a div which contains a QR code and an input box as shown in the figure (highlighted in green).

2. **Selecting the Juggling Method** Once the bookmarklet UI is displayed on the targeted web site, the user starts the Session Juggler application on her phone. There she can choose between the two ways of initiating the juggling: the *visual login* mode that will use the QR code or the *pin login* mode that will use the bookmarklet input box. In the *visual mode*, the phone camera is used to read the QR code from the page to get all the necessary information to initiate the juggling. As we will see in more detail in the next section, the information exchanged includes the user agent, the url, and an AES key. With the *pin login* mode, the phone generates an AES key that the user then must input in the bookmarklet input box. While generating the key on the phone and inputting on the browser creates one more round of exchanges in the juggling protocol, it is generally easier to type a string on the terminal keyboard than on the phone. During our tests, we didn't notice any visible difference between using the *visual login* and the *pin login* mode in terms of latency.

3. **Giving consent** Once the juggling is initiated, before going to the site, the session juggler app displays a confirmation dialog showing the website favicon and domain name and asks the user to confirm that it is the website where she wants to logon. This confirmation dialog is used to mitigate phishing attacks and in particular a phishing attack where the malware changes the URL requested by the bookmarklet. *Session Juggler* also checks the URL against the google blacklist as an additional layer of defense.

4. **Login to the site** Once the user has given her consent, the Android app navigates to the login page and the user logs in on the phone. At this point, the Android native password manager will ask the user if she wants to login. This step is only required during the first login since the password manager saves the credentials. On subsequent logins the interaction with the phone this step is skipped.

5. **Transferring the Session** After successful login on the phone, the last step is to click on the transfer session button (highlighted in green on the figure) to transfer session state to the insecure terminal. We need to ask the user to click the transfer button since some web sites, such as *bankofamerica.com* and *sfcu.org*, have a multi-step login process where the user enters her ID on the first page and the password on a subsequent page. Consequently, only the user can tell when login is complete.

[2]The screenshots used in this storyboard were made using Firefox and our open source Android implementation of Session Juggler.

Figure 2: Session Juggler workflow illustrated

6. **Enjoying the Site** Less than a second after the transfer session button is clicked, the bookmarklet resumes the session on the insecure terminal and the user can enjoy her session. The user's login and password were never entered on the insecure terminal. All the terminal sees is the ephemeral session token.

Setup phase. Setting up *Session Juggler* is straightforward: On the Android phone, installing the app is as easy as installing any other app from the Android Market. To install the bookmarklet on the untrusted terminal the user has simply to remember the bookmarklet URL or use the short url associated with it. As a reminder, a short URL to the bookmarklet is displayed in the setting menu of the Android app.

Logging out. Since the session was first initiated on the phone, the phone has the session token and can, at a later time, issue a logout request for the user when it is asked to do so. This logout request, if properly honored, will invalidate the session token server-side thereby terminating the session on both the phone and the untrusted terminal. Note that in practice, as discussed in section 2, this might not work as some websites, such as Linkedin, that do not properly honor logout requests.

Until a standard format to describe logout URL is adopted, *Session Juggler* has no way of automatically finding the logout URL. This means that we have to resort to requiring the user to logout manually. To make this process easier, we implemented two workarounds. First, for popular sites like Facebook, Google, Linkedin, and Twitter, we built a database of logout URLs that give a single-click logout. Second, *Session Juggler* learns the URLs that the user previously used to logout of a given website.

After the first time logout, *Session Juggler* provides a single-click logout button on the phone. Note that in either case we show the resulting page so the user can visually verify that logout succeeded.

3.3 The Juggling protocol

Now that we have explained what the user experience with *Session Juggler* is, we describe how the juggling if effectively implemented. We start by describing the participants involved in the juggling and then we describe the juggling protocol step by step.

3.3.1 Juggler Principals

Overall juggling a session involves three main principals, which are depicted in figure 3: a web service named "Blackboard" after the design pattern of the same name [40], a phone application, and a bookmarklet on the browser in the insecure terminal.

The Bookmarklet.

The bookmarklet is a small piece of javascript that allows us to perform computations on the untrusted terminal browser. Using a bookmarklet is commonly used by many websites, including Wordpress and Readable, to provide quick cross browser access to a feature that requires executing javascript without requiring the user to install anything on the browser. As mentioned earlier, when the user starts using an untrusted terminal, she has to visit the bookmarklet website and drag and drop a link to the bookmarklet on her bookmark bar. When the user clicks on the bookmarklet, the javascript code is executed in the context of the site, which allows the bookmarklet to read the page location and content, set cookies, and navigate to the logged-in page once the juggling is done.

Our bookmarklet uses the SJCL Javascript crypto library [37] to encrypt/decrypt requests with AES and can be downloaded from `http://ly.tl/sj`.

The Blackboard.
The blackboard is a web service that facilitates the exchange of information between the phone and the terminal. This "middleware" which obeys the "Blackboard" design [40] pattern is needed because often the phone can't be reached directly by the terminal. For instance, the phone and the terminal might be on two different networks (3G / WiFi). Accordingly, the bulletin board can be viewed as a convenient way to write and read encrypted data and therefore does not need to be trusted. Note that the bulletin board can be any web service that allows a user to read and write data (i.e. Twitter). It can even be an HTTP server on the phone if the terminal is able to connect to it.

The blackboard is a very simple piece of code that can be implemented in less than 200 lines. In our short PHP implementation, each juggling session is identified by a parameter called ID that allows our blackboard to know which data to store and retrieve. Having a unique identifier for each juggling session is all it is takes to allow our blackboard to handle simultaneous users and sessions. Juggling data are stored in a simple file that is wiped after 24h by a cron script. Note here that all the data stored in the blackboard are encrypted so the blackboard can't tamper with any juggling session data (except to block the session transfer by erasing files).

As a security precaution, we limit the size of the data written to the file, as the amount of information exchanged during the juggling is very small. Note that the bookmarklet javascript is subject to the same origin policy and therefore the blackboard needs to authorize cross-domain requests via pre-flight requests [39] to allow the bookmarklet to read content. For older browsers that don't implement cross-domain requests, it is possible to use "hacks" such as the external script include trick to circumvent the same origin policy [5].

The trusted phone application. The phone application is the trusted piece of software that is used to logon to the website requested by the user. In addition, the phone application is responsible for transferring the session data to the bookmarklet through the blackboard and for providing a secure logout system. We have implemented our trusted application on Android using the standard framework. However, because the application needs to impersonate the untrusted terminal browser (i.e spoofing the user-agent), the standard *webview* is not directly usable. We ended up doing the HTTP requests ourselves and then passing the returned content to the web view to render for the user.

3.3.2 Protocol

As illustrated in section 3.2 , juggling can be initiated using a QR code (visual login) or by entering a pin code (pin login). Figure 3 describes the juggling protocol when the QR code is used, and the figure 4 depicts the protocol when a pin code is used. Because the visual login protocol requires one less exchange than the pin code protocol, we start by describing it.

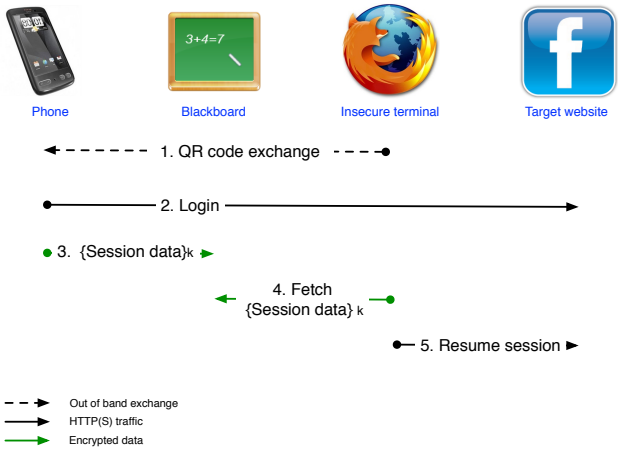

Figure 3: QR mode message flow

The Visual Login Protocol. As visible in figure 3, the visual login version of the juggling protocol is achieved in five steps:

1. **Transmitting Data via the QR Code**: When the user clicks on the bookmarklet, it generates a 128 bit AES key k using the crypto library (SJCL). k will eventually be used to encrypt the session data. Then the bookmarklet computes the id required to use the blackboard as follows: $id_i = \text{HMAC}_k(0)$. When this is done the bookmarklet encodes the key k, the browser version, the untrusted terminal OS version, and the target URL into a QR code. Finally the bookmarklet displays the QR code on top of the web site in the browser. The bookmarklet will keep polling the blackboard via XHR requests until it receives session data from the phone via the blackboard. On the phone side, the user input to the application the QR code content by taking a picture of the untrusted terminal screen.

2. **Login to the Site**: Using the site URL supplied by the bookmarklet via the QR code, the application navigates to the login page. Our application uses the browser and OS version supplied by the bookmarklet to spoof the user-agent header. This is mandatory because if we don't set the `user-agent` appropriately, the target web site might respond with the mobile version of the site or may later refuse the transfer of the session because of the site unlikely use of anti-hijacking defenses (See section 2). Once the user has confirmed she wants to login, the page is loaded and the user's password manager asks the user if she wants to supply her credentials.

3. **Session Acquisition** Once the user has successfully supplied her credentials, the web site redirects the mobile browser to the logged-in page and sets the cookies pertaining to the user's session. Once the logged-in page is loaded and the user has clicked on "transfer session," the session data are gathered by the application, encrypted with the key k and posted to the blackboard under the id id_i. The session data contains the URL with all its arguments and the cookies. Note that the Android framework does not offer a method to read all the cookies' information, such as the cookie path, so we had to use the `CookieSyncManager` class to force Android to store all cookies present in RAM into the "`webview.db`" sqlite database and then run a SQLite query to retrieve them.

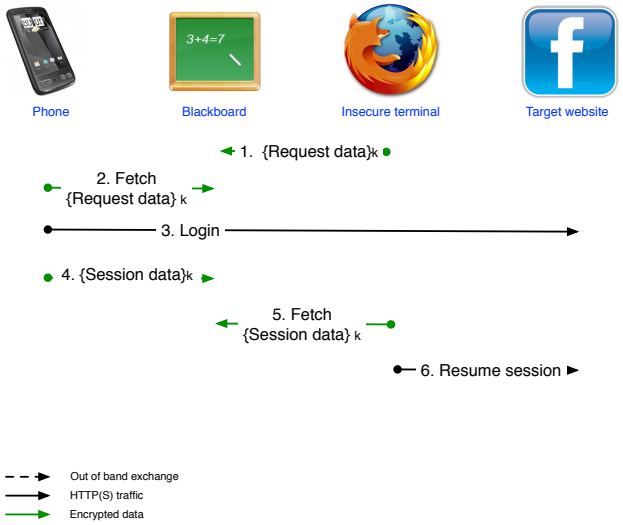

Figure 4: Pin-code message flow

4. **Session Transfer** A few milliseconds after the phone has successfully posted the encrypted data on the blackboard, the bookmarklet polls the blackboard by supplying the id id_i, and fetches the data.

5. **Resuming the Session** The bookmarklet decrypts the data polled from the blackboard using the key k, sets the cookies in javascript using the *document.cookie* object, and then navigates to the page by setting the *document.location* to the logged-in url page. The browser reacts to this by navigating to the logged-in page and the session is successfully transferred to the untrusted browser/terminal.

The pin login protocol. As visible in figure 4, the pin code juggling protocol is very similar to the visual mode except that it requires an extra exchange (steps 1 and 2) at the beginning. When the pin code protocol is used the AES key k is generated on the phone and then entered via the keyboard (128bits in hex) on the untrusted terminal's browser.

Accordingly, after the key is exchanged, the phone has no idea which website the user wants to login to and therefore the extra exchange is needed so the bookmarklet can tell the phone application which website the user wants to log into. The content of this exchange is pretty straightforward: the bookmarklet takes all the information that was encoded into the QR code and encrypts it using the key k. Then it posts it on the blackboard using the id id_i. A few milliseconds later the phone polls the request data from the blackboard by requesting data for the id k_i. From there the protocol is similar to the visual login protocol.

3.4 Security Analysis

Let's now analyze why *Session Juggler* improves login security.

First and foremost, *Session Juggler* improves the security by limiting malware's effectiveness by allowing it only to capture short-lived session credentials rather than long term ones. During the juggling the user's long term credentials are never inputted nor transferred to the insecure terminal, so even if the terminal is infected with malware, the attacker learns nothing about these credentials.

Similarly *Session Juggler* effectively mitigates the risks of using an insecure network (Sniffing, SSL strip) by transmitting user credentials over a 3G network which is encrypted.

The blackboard does not know the key k and does not see the key exchange, as it is done through an out of band exchange (QR code/pin code input). Thus the blackboard is unable to decrypt the session data. Accordingly an attacker has no incentive to create a rogue blackboard since it will only contain useless encrypted data. The only advantage that a rogue blackboard offers to the attacker is that it allows him to prevent the user from logging in by erasing all data. However since nothing prevents the user from switching blackboards, there is little incentive to do so. On the other hand, the owner of the untrusted computer's malware has no incentive to prevent the session from being be executed as it will still get him the short terms credentials.

In order to steal the long term credentials, the malware owner can resort to phishing the user by replacing the url sent by the bookmarklet with a malicious url. This kind of phishing is actually less dangerous than the regular phishing attacks because *Session Juggler* has three phishing defenses– a user consent popup that emphases the domain request, a domain blacklist check, and a password manager that only discloses passwords to the right url– already built-in on Android phones, leaving them out of harm's way.

Finally, because the phone and the untrusted terminal share the same session data, the malware can't prevent the user from logging out through the phone app, which leaves the malware owner with a smaller window of opportunity. Note that many websites, including *Amazon* and *Twitter*, require users to input their passwords on significant changes, which limits even further the effectiveness of hijacking a session.

4. EVALUATION

Our evaluation shows that *Session Juggler* can be successfully used to login on **87%** of the Alexa Top-100 websites. Additionally we also have a **100%** success while using *Session Juggler* to login on websites that use *Facebook connect*, which implies that *Session Juggler* can be used to login on more that **85000** sites [36].

We manually tested if we were able to use *Session Juggler* and Firefox to login on the 64 Alexa Top-100 websites that have a login system. The relatively low number of sites that have login capabilities is partially explained by the fact that *Google* appears multiple times in the Alexa top-100 rankings due to its various local versions. We only counted all these versions as one site so as not to skew the results in our favor.

We chose the visual mode to verify that websites' designs do not interfere with the QR code reading process. Overall we can report that *Session Juggler* is a viable solution as we were able to successfully perform a session transfer for **87%** of the most popular websites in the world. Moreover *Session Juggler* works perfectly with *Facebook connect* which implies that, according to the latest statistics, *Session Juggler* can be used to login successfully on more than **80000** websites [36]. The three case where *Session Juggler* fails (msn.com, megaupload.com and rapidshare.com) provide a couple of interesting observations that are discussed below.

Dealing with Mobile Session Separation. As mentioned in section 2, websites like *twitter.com* and *eBay* differentiate between authenticated sessions depending on whether the client is a smartphone or a regular web browser. Because they use the *user-agent* header to distinguish between the two, we had to modify our early prototype so the terminal browser's information is encoded into the transfer request. On the Android side we had to write custom a HTTP request code to emulate as closely as possible the terminal side headers, thus concealing the fact that the login was performed by a smartphone.

Finding Webview Limitations. Unfortunately, 2 out of the 3 *Session Juggler* failures are not directly fixable by us as they stem from the fact that the Android WebView is unable to render the full version of msn.com and rapidshare.com. Our other failure is due to the fact that megaupload.com automatically gets redirected to the website's mobile version, preventing session transfers. It is likely that Megaupload performs some kind of browser fingerprinting in javascript that requires us to develop a way to clobber the webview javascript variables. As discussed below, dealing with javascript browser fingerprinting is part of the extensions we propose for this work. Note that the megaupload fingerprinting behavior was not detected during our survey of the website hijacking defenses because megaupload does not use it for session binding.

Handling Secure Cookies. The first version of our Android application extracted cookies from the WebView object directly using the `CookieManager` interface. During our evaluation we found out that this mechanism failed to retrieve cookies marked 'secure'. We observed this odd behavior while testing authentication on *bing.com*, where certain cookies like `KievRPSSecAuth` were missing on the client browser even after the session transfer. In its current version, our application fetches the cookies directly from the cookie repository via the Sqlite interface as explained in section 3.

The Third Party Login Challenge. While testing we observed that, for *flickr.com* and *bing.com*, the actual authentication is done in a different domain (for example *yahoo.com* for *flickr.com*). While *Session Juggler* works with third-party login, we unfortunately had to enter our credentials twice, once on the domain and once an third party domain, to make the transfer work. Finding a more elegant solution to this problem is an open question. Note that this issue does not affect Facebook connect.

5. DISCUSSION

Before concluding, we discuss some extensions we made to improve *Session Juggler* and discuss our plan to deal with more aggressive session binding mechanisms if such mechanisms are put in place in the future.

Improving Usability via Long Term Pairing. One of the drawbacks of our architecture is that it requires the user to exchange a new key between the phone and the desktop every time the user want to login. If appropriate, this process can be made easier by using a long-term pairing between the phone and the desktop. Due to security concerns, the pairing key K cannot be stored in the bookmarklet [3], but can be stored in a browser extension, if one can be installed. At setup the key is copied to the phone via a QR code. Once the setup is completed, the phone app runs a service that polls the blackboard regularly and prompts the user to login as soon as it polls a new request. This might prove for computers that the user use frequently but do not fully trust such as a family's computer or a corporate one.

Improving Security via Cookie Editing. As sessions are transferred from the phone to the terminal, our phone application rewrites the transferred cookies so they become session cookies by removing the expiration date. Removing the expiration date improves user privacy as the cookies will not be written to disk. It also slightly improves the user's security because if the user forgets to logout but still closes the browser, the cookies are removed from memory.

Dealing with IP Binding. As mentioned in section 2.2, some websites bind the session to a specific IP. In this case the main options is to have the phone connected on the same network. However connecting the phone to the same network might not always possible. Another potential issue with this approach is the case where the website doesn't use HTTPS to transmit the credentials. In this case, the benefit of using session-juggler to protect against man-in-the-middle attack disappears as the 3G connection ceases to be used as an alternative-encrypted channel. We plan to add to *Session Juggler* a confirmation dialog when the user tries to send credential over HTTP on a WiFI connection.

Dealing with Aggressive Fingerprinting. While none of the web sites we tested use aggressive browser fingerprinting to bind a session to a browser, this may change in the future. More aggressive browser fingerprinting (such as testing the screen resolution, installed plug-ins, etc.) will make it harder to move the session from the phone to the terminal. Dealing with this kind of fingerprinting will require a significant additional development effort so the phone can impersonate an arbitrary version of Flash or Java. An alternative option would be to turn Java or Flash off on the terminal side. However since none of this is needed today — web sites do not fingerprint the browser for session binding — we leave this as interesting future work.

6. RELATED WORK

6.1 Previous Approaches

In this section we present the previous work pertaining to using a trusted device such as a PDA or a phone to login from an untrusted terminal and explain why it is not sufficient for our purposes. Overall, as visible in the summary table 3, as far as we can tell all the proposals to date require at least a server-side or terminal-side modification. It is only by leveraging the advance in mobile browsers and using session hijacking attack principle in an unexpected (and benevolent!) way the that *Session Juggler* is able to use a smartphone to login with no terminal-side or server-side modifications.

In a seminal work [7], the authors propose to use the Palm Pilot as a second authentication factor by implementing a one-time password generator on it. This work, as well as all related subsequent works requires server modifications. In [29], the authors propose using a PDA as a trusted third party by connecting it to the insecure terminal via USB. This scheme requires terminal-side modifications and server-side modifications. In [35], the authors propose using a mobile phone as a trusted input device to input data on untrusted terminal. The phone and the terminal are connected via a so-called "thin-client server" that acts as a relay. Installing this thin-client requires a terminal-side modification to be installed and a server-side modification to be used.

In [27] the authors propose the *MP-Auth* protocol that works by storing the user's long-term secret on a mobile device. The *MP-Auth* protocol requires server-side changes and needs the untrusted browser to communicate with the phone either via bluetooth or USB,

	[7]	[29]	[35]	[27]	[17]	[34]	[38]	*Session Juggler*
Year	1999	2004	2006	2007	2008	2008	2009	2012
Trusted device	Palm Pilot	PDA	Phone	Phone	Phone	Phone	Phone	Phone
Server-side modification	✓	✓	✓	✓		✓		
Terminal-side modification	✓	✓	✓	✓	✓	✓	✓	
Connection type	USB	USB	Net	USB/BT	USB	Net	NFC	3G/WiFi
Hardware needed					TPM		TPM/NFC	

Table 3: Comparison to previous approaches

which requires a client-side modification that involves installing a binary and a Firefox extension.

In [17] the authors combine the use of the terminal Trusted Platform Module (TPM), virtual machine, and phone to convince users they can trust the executed virtual machine. While this approach involves a huge-client side modification (using a specific hypervisor) and the presence of specific hardware, it is one of the first approaches, if not the first approach, to not require server side changes. The idea of using phone-attested virtual machine on the terminal-side was extended in 2009 [38] to use NFC (Near Field Communication) as a direct channel for the TPM's identity proof.

In [34], the authors propose splitting the display between the untrusted terminal and the phone. Their scheme requires a terminal-side modification that involves installing a Firefox extension. The aforementioned extension acts as a RDC (Remote Desktop Client) agent that forwards data to the phone. This scheme also requires a server-side modification so the web site is "separation aware." This scheme additionally requires an HTTP daemon on the phone which implies that there is a direct connection between the phone and the untrusted terminal.

Another approach, followed by some websites (such as Google), is to allow users to generate a list of passwords that are only usable one time. This approach requires a server-side modification and forces the user to carry a list of passwords for each site he or she wants to use. This approach also does not address the issue that a malware can prevent the user from logging out.

Finally, while three-leg protocols such as OAuth [11] are becoming the standard to delegate permission to "semi"-trusted applications, it is not a suitable approach to deal with untrusted terminals as the pairing process involves supplying the long term credentials at some point during the pairing process. This approach also requires server-side and terminal-side modifications.

6.2 Further Related Work

The Phone as a Second Factor. In [13] the authors survey the type of devices that can be used as second factors. In [6] the authors propose to use the phone as a one-time password generator. This work proposes a solution that is similar to the SecurID one time password solution [26]. In [20] the authors perform a formal analysis of protocols that use a phone as a second factor.

Session Hijacking. Session hijacking by a network attacker can be mitigated in part by HTTPS and marking all cookies as secure [25]. Other ideas include [24] which proposes to defend against session hijacking using a proxy, [15] which advocates end-to-end security, [22] which uses client-side policies to defend against session hijacking, and [2] which proposes an elegant idea for embedding a session token in the URL fragment identifier. In [18] the authors study methods to detect passive WiFi session hijacking by leveraging physical layer properties. Defenses against session hijacking by malware are based on "transaction confirmation" and such systems are presented in [21, 30, 31] and used by `authentify.com`.

Browser Sniffing. Browser fingerprinting is a very active field that includes a wide set of techniques. For instance Panopticlick [14] uses installed fonts, plug-ins, time zone, and many other parameters to identify the browser. Companies such as *41st Parameters* [1] sell this kind of technology to websites for fraud detection. In [4], the authors showed that web browsers can still be tracked even with the private mode enabled.

Storing Passwords on Phones. Storing passwords on phones is an active area. In [8] the authors propose to hide passwords in plain sight by creating thousands of decoys that are indistinguishable from the real password set. They also propose to use "honey words" to defend against online attacks. A related concept is mentioned in [12].

7. CONCLUSION

In this paper we present *Session Juggler* which is the first universal solution to perform a secure web login on an untrusted terminal. *Session Juggler* is universal as it does not requires any website modification and do not requires any specific software on the terminal beside a modern browser. Our evaluation shows that *Session Juggler* works with every Alexa Top 100 websites except eight. *Session Juggler* also works flawlessly with Facebook connect, which allows users to use it on more than 85000 websites. Finally *Session Juggler* is the first solution that provides a trusted logout mechanism that ensures that the user session will be killed as soon as the user is finishing using the website.

Acknowledgments

The authors thank Charlie Reis and Collin Jackson for many helpful discussions about this work. This work was supported by NSF, DARPA, and an AFOSR MURI grant.

8. REFERENCES

[1] 41st Parameters. Deviceinsight. http://www.the41.com/land/DeviceID.asp.

[2] B. Adida. Sessionlock: Securing web sessions against eavesdropping. In *World Wide Web*, 2008.

[3] B. Adida, A. Barth, and C. Jackson. Rootkits for javascript environments. In *Proc. of 3rd USENIX Workshop on Offensive Technologies (WOOT 2009)*, 2009.

[4] G. Aggarwal, E. Bursztein, C. Jackson, and D. Boneh. An analysis of private browsing modes in modern browsers. In *Usenix Security*, 2010.

[5] O. Alliance. Ajax and mashup security. Technical report, OpenAjax Alliance, 2008.

[6] F. Aloul, S. Zahidi, and W. El-Hajj. Two factor authentication using mobile phones. In *Computer Systems and Applications, 2009. AICCSA 2009. IEEE/ACS International Conference on*, pages 641–644. IEEE, 2009.

[7] D. Balfanz and E. Felten. Hand-held computers can be better smart cards. In *Proceedings of the 8th conference on USENIX Security Symposium-Volume 8*, page 2. USENIX Association, 1999.

[8] H. Bojinov, E. Bursztein, X. Boyen, and D. Boneh. Kamouflage: Loss-resistant password management. In *Proc. of ESORICS'10*, 2010.

[9] E. Butler. Firesheep. http://en.wikipedia.org/wiki/Firesheep.

[10] E. Butler and I. Gallagher. Hey web 2.0: Start protecting user privacy instead of pretending to. ToorCon 2010, 2010. sandiego.toorcon.org.

[11] O. community. Request for comments: 5849 the oauth 1.0 protocol. http://tools.ietf.org/html/rfc5849, 2010.

[12] D. Dasgupta and R. Azeem. An Investigation of Negative Authentication Systems. In *Proceedings of 3rd International Conference on Information Warfare and Security*.

[13] D. de Borde and S. Consulting. Two-factor authentication. *Siemens Enterprise Communications UK-Security Solutions*, 2008.

[14] P. Eckersley. How unique is your web browser? In *Proc. of PETS 2010*, number 6205 in LNCS, pages 1–18, 2010.

[15] U. Erlingsson, B. Livshits, and Y. Xie. End-to-end web application security. In *Proceedings of the 11th USENIX workshop on Hot topics in operating systems*, pages 1–6. USENIX Association, 2007.

[16] D. Florencio and C. Herley. A large-scale study of web password habits. In *Proceedings of the 16th international conference on World Wide Web*, pages 657–666. ACM, 2007.

[17] S. Garriss, R. Cáceres, S. Berger, R. Sailer, L. van Doorn, and X. Zhang. Trustworthy and personalized computing on public kiosks. In *Proceeding of the 6th international conference on Mobile systems, applications, and services*, pages 199–210. ACM, 2008.

[18] R. Gill, J. Smith, and A. Clark. Experiences in passively detecting session hijacking attacks in IEEE 802.11 networks. In *Proceedings of the 2006 Australasian workshops on Grid computing and e-research-Volume 54*, pages 221–230. Australian Computer Society, Inc., 2006.

[19] Google. Google hall of fame. http://www.google.com/about/corporate/company/halloffame.html.

[20] A. M. Hagalisletto. Analyzing two-factor authentication devices.

[21] C. Jackson, D. Boneh, and J. Mitchell. Transaction generators: Root kits for the web. In *Proc. of the 2nd USENIX Workshop on Hot Topics in Security*, 2007.

[22] M. Johns. SessionSafe: Implementing XSS immune session handling. *Computer Security–ESORICS 2006*, pages 444–460, 2006.

[23] N. Johnston. Scareware haunts airport internet terminals, 2010. symantec.com/connect/blogs/scareware-haunts-airport-internet-terminals.

[24] F. Kerschbaum. Simple cross-site attack prevention. In *Security and Privacy in Communications Networks and the Workshops, 2007. SecureComm 2007. Third International Conference on*, pages 464–472. IEEE, 2008.

[25] V. Khu-smith and C. Mitchell. Enhancing the security of cookies. *Information Security and Cryptology—ICISC 2001*, pages 197–230, 2002.

[26] R. Laboratories. One-time password specifications (otps). http://www.rsa.com/rsalabs/node.asp?id=2816.

[27] M. Mannan and P. Van Oorschot. Using a personal device to strengthen password authentication from an untrusted computer. In *Proceedings of the 11th International Conference on Financial cryptography and 1st International conference on Usable Security*, pages 88–103. Springer-Verlag, 2007.

[28] Microsoft. Security researcher acknowledgments for microsoft online services. http://technet.microsoft.com/en-us/security/cc308589, Oct 2011.

[29] A. Oprea, D. Balfanz, G. Durfee, and D. Smetters. Securing a remote terminal application with a mobile trusted device. 2004.

[30] S. N. Patel, J. S. Pierce, and G. D. Abowd. A gesture-based authentication scheme for untrusted public terminals. In *UIST '04: Proceedings of the 17th annual ACM symposium on User interface software and technology*, pages 157–160, New York, NY, USA, 2004. ACM.

[31] H. Qian, C. Surapaneni, S. Dispensa, and D. Medhi. Service management architecture and system capacity design for phonefactor: A two-factor authentication service. *Integrated Network Management, 2009. IM '09. IFIP/IEEE International Symposium on*, pages 73 –80, jun. 2009.

[32] G. Rydstedt, B. Gourdin, E. Bursztein, and D. Boneh. Framing attacks on smart phones and dumb routers: tap-jacking and geo-localization attacks. In *Proceedings of the 4th USENIX conference on Offensive technologies*, pages 1–8. USENIX Association, 2010.

[33] E. Security. Sidejacking. http://erratasec.blogspot.com/2008/01/more-sidejacking.html, 2008.

[34] R. Sharp, A. Madhavapeddy, R. Want, and T. Pering. Enhancing web browsing security on public terminals using mobile composition. In *Proceeding of the 6th international conference on Mobile systems, applications, and services*, pages 94–105. ACM, 2008.

[35] R. Sharp, J. Scott, and A. Beresford. Secure mobile computing via public terminals. *Pervasive Computing*, pages 238–253, 2006.

[36] Shishir. Top 30 interesting facebook figures. http://www.shishirk.com/2011/02/interesting-facebook-figures/, Feb 2011.

[37] E. Stark, M. Hamburg, and D. Boneh. Fast symmetric cryptography in javascript. In *Proc. of ACSAC 2009*, 2009.

[38] R. Toegl. Tagging the turtle: local attestation for kiosk computing. *Advances in Information Security and Assurance*, pages 60–69, 2009.

[39] A. van Kesteren. Cross-origin resource sharing. Technical report, W3C, July 2010.

[40] Wikipedia. Blackboard system. http://en.wikipedia.org/wiki/Blackboard_system.