Black Hat USA 2010

Bad Memories

Elie Bursztein, Baptiste Gourdin, Gustav Rydstedt, Dan Boneh

Stanford University **Security Laboratory**

http://seclab.stanford.edu

Preface

Bad memories summarize our latest research results on offensive web technologies. The Security Lab is a part of the Computer Science Department at Stanford University. Research projects in the group focus on various aspects of network and computer security.

While secure communication protocols have received a lot of attention and have been widely deployed over the last few years, the way their sensitive data is stored remains a weak link in practice. The purpose of this paper is to raise awareness of this fact and demonstrate that attackers can make such secure communication protocols irrelevant by targeting the data storage mechanism.

In this paper, we demonstrate the weakness of current storage mechanisms by showing the following attacks: first, we show how an attacker can remotely locate and break into a Wifi network by crafting a malicious web page that targets its access point. Secondly, we demonstrate how an attacker can inject a malicious library that is capable of compromising subsequent SSL sessions by leveraging the fact that websites trust external javascript libraries, such as Google Analytics. We then describe how to easily fool the user into accepting this malicious javascript library by exploiting browser UI corner cases. Next, we introduce frame leak attacks that are capable of extracting private information from the website (and not from the user) by leveraging the recent scrolling technique of Stone. Our frame leak attacks defeat click-jacking defenses that have previously been considered secure. In addition, we illustrate how a frame leak attack works by demonstrating how to use it to extract Facebook profile information, bypassing Facebook's framebusting defenses in the process. Finally, we develop a new attack called tap-jacking that uses features of mobile browsers to implement a strong clickjacking attack on phones. We show that tap-jacking on a phone is more powerful than traditional clickjacking attacks on desktop browsers, and thus imply smartphones should not be considered a secure form of data storage.

Table of Contents

1 Introduction

While secure communication protocols have received a lot of attention and have been widely deployed over the last few years, the way their sensitive data is stored remains a weak link in practice. The purpose of this paper is to raise awareness of this fact and demonstrate that attackers can make such secure communication protocols irrelevant by targeting the data storage mechanism. We demonstrate the weakness of current storage mechanisms by showing four different kind of attacks that every time exploit a storage mechanism weakness.

Breaking into a WPA network with a Webpage

We show that many popular wifi routers are vulnerable to framing and XSS attacks that can be used to steal the router's WPA secret key and to accurately locate the router on a map. Our attacks make use of new same origin bugs in current versions of Firefox and Chrome. We were able to carry this out as an automated attack on eight different brands of routers: *Belkin, Netgear, D-link, Linksys, Buffalo Zyxtel, SMC, TrendNet*. The end result is that an attacker can create an accurate world map of WPA keys needed to access private wifi networks where wifi is available.

Defeating HTTPS via cache injection

Every major browser implement caching in order to display webpage faster. We demonstrate how an attacker can exploit this caching mechanism by injecting a malicious library that is capable of compromising subsequent SSL sessions. Cache injection attacks work on the vast majority of popular websites because they are including javascript file in their pages. Moreover injecting a single malicious library allow an attacker to target multiples websites when they are using the same third party javascript library such Google Analytics.

Attacking Facebook with Frame leak attack

Facebook deploys an interesting frame busting defense — when framed the site places a dark transparent `div` over the page. This `div` lets the user see the page contents, but any click on the `div` causes the top window to navigate to Facebook's main site. We show in Section 4 that even though this frame busting defense might be reasonable against traditional clickjacking attacks, it is still vulnerable to a "frame leak" attack. Frame leakage leverages a recent scrolling attack [14] to learn private information about the victim. We demonstrate the effectiveness of this attack by showing how an attacker can use it to learn private information about a user's Facebook profile despite Facebook's frame busting defense.

Phone tapjacking

Web framing attacks take place in the browser and begin with a malicious page loading a victim page as an iframe. Once the victim page is framed the framing page can mount a variety of attacks. The most common example is clickjacking [7] where the victim page is loaded as a transparent frame over an innocuous page that tempts the user to click on buttons or links.

When the user clicks, the click event is sent to the transparent victim frame causing some unintended action to take place on behalf of the user (e.g. sending a tweet or purchasing an item). Other framing attacks include UI redressing [6] (where the framing page places frames on top of the victim page), drag-and-drop attacks [14], and scrolling attacks [14]. We discuss these attacks in more details later on in the paper. The standard defense against framing attacks, called frame busting, refers to code or annotation in a web page intended to prevent the web page from being loaded in a sub-frame [12]. The following simple frame busting code is a commonly used:

```
if (top.location != location)
   top.location = self.location;
```

A recent survey of clickjacking defenses on popular sites [12] shows that only 14% of Alexa Top-500 implement some variant of frame busting. The survey also shows that current methods can be easily circumvented and proposes better frame busting methods.

In this paper we study framing attacks on mobile sites and social sites, and framing attacks on sites embedded in consumer electronics, specifically routers. We develop attacks showing that smartphones and routers are highly vulnerable to framing attacks, much more than regular browsers and public web sites. Despite these significant vulnerabilities very few mobile and embedded sites defend themselves against framing. We also show that some framing defenses, such as those employed by Facebook, prevent standard clickjacking but can nevertheless lead to private information leakage. We found that 53% of Alexa-Top 500 sites have mobile alternatives to their primary site designed to render better on a phone. These are most often served in the `.mobi` domain, or in `m.*, mobile.*, wap.*` subdomains. A majority delivers a significant subset of their functionality to their mobile sites. While 14% of the top 500 sites do some form of frame busting on their main site, virtually none use frame busting on their mobile sites (we only found one site that frame busts both its main site and its mobile site). As a result almost all mobile sites are vulnerable to framing attacks on the phone. We show in Section 5 that framing attacks on the phone are even more effective than their desktop counterparts. We introduce *tap-jacking*, a mobile web attack that is far more dangerous than its clickjacking cousin on the desktop. These attacks show that essentially all current mobile sites are easily compromised.

At a minimum, tap-jacking requires a mobile browser with support for javascript and frames. We use extra features of mobile browsers to make the attack more effective. Both the iPhone and Android browser have all the features needed for tap-jacking. Surprisingly, the Opera mini browser seems to be immune to tap-jacking, despite its full support for Javascript and frames. We discuss this in more detail in Section 5.2.

Lessons

It is not clear why sites that frame bust on their main site fail to do so on their mobile equivalent. When discussing this issue with developers we predominately hear two arguments:

1. Older mobile browsers do not support the JavaScript needed for frame busting. The concern is that frame busting code will cause the mobile site to render incorrectly on older phones. However this concern is easily addressed by selectively rendering based on user-agent. That is, inject frame busting code when the user agent is an iPhone or Android and do not inject it if the browser is an older phone.

2. Clickjacking is not an issue on cell phones. We hope this paper demonstrates that this assumption is simply not true. In fact, we show that quite the opposite holds: framing attacks are more effective on smart phones than on desktops.

As an aside we note that most mobile sites do not check that the user agent is a mobile phone and happily render in a desktop browser. Even worse, most sites do not differentiate sessions between the main site and the mobile site (i.e. a user logged in at the main site is also logged in at the mobile site and vice versa). As a result, if a site frame busts on the main site but not on the mobile site, an attacker can frame the mobile site on a desktop client and mount a framing attack of its choice on the site. For example, if a user is logged into twitter on the desktop, an attacker on the desktop can frame the mobile twitter site and use the user's credential from the main site to post tweets on behalf of the user. Clearly mobile sites should frame bust if the user agent indicates a phone that supports frame busting. If this is not possible for some reason then at the very least sites should not share sessions between the main site and the mobile site.

2 Breaking into a WPA network with a Webpage

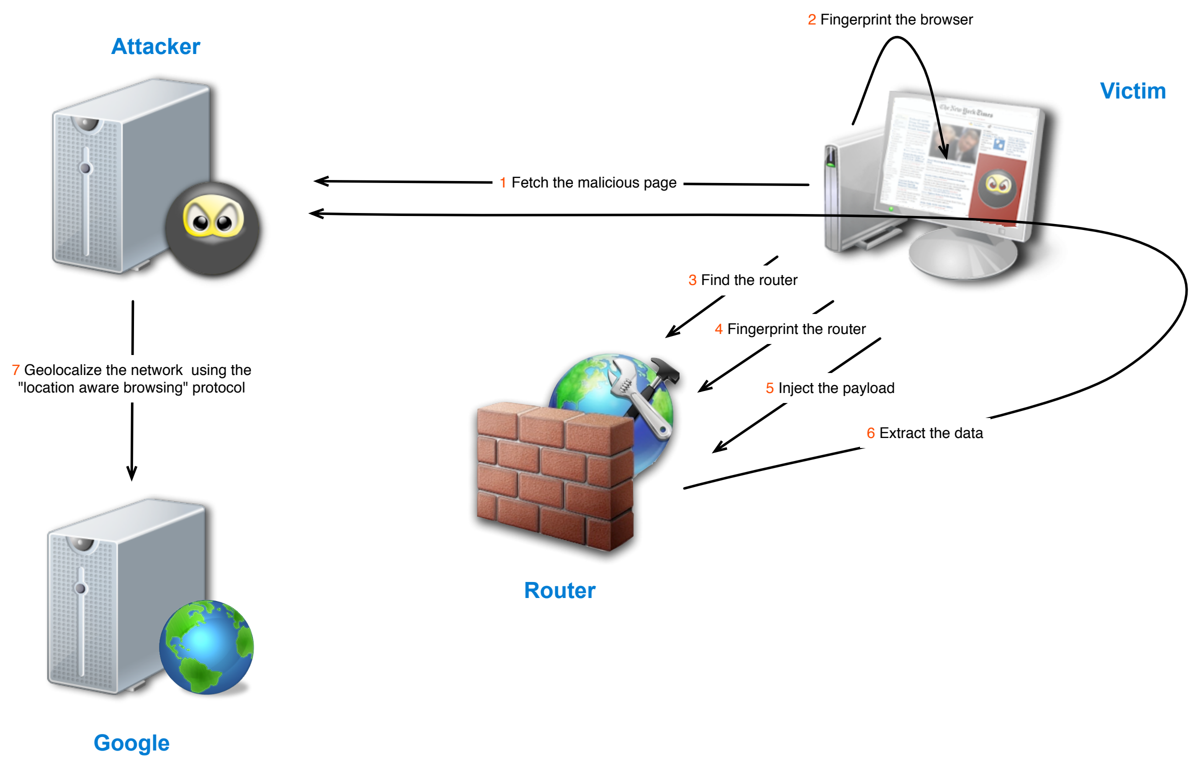
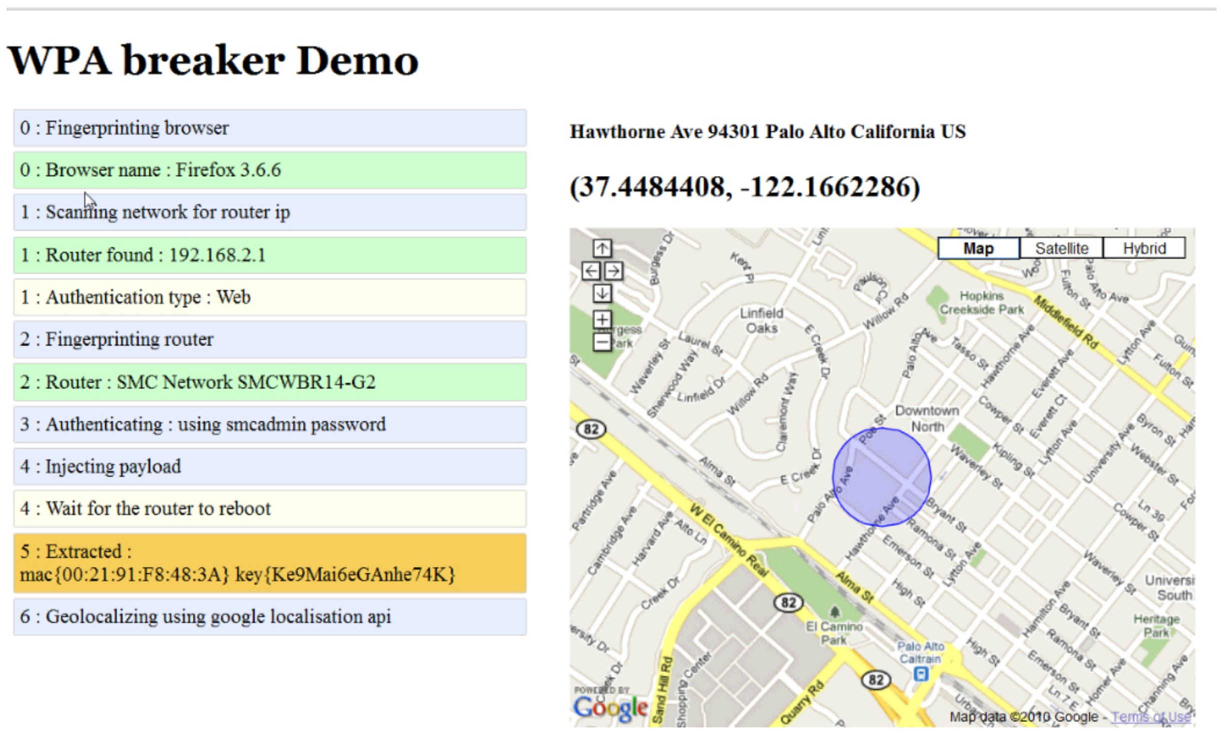

Attacker
2 Fingerprint the browser
Victim
1 Fetch the malicious page
3 Find the router
4 Fingerprint the router
5 Inject the payload
6 Extract the data
7 Geolocalize the network using the "location aware browsing" protocol
Router
Google

Figure 1: Router attack flow

Many routers provide configuration and monitoring web interfaces. In this section we show that using a multi-stage attack, including framing or XSS injection, an attacker can steal a router's WPA key and determine the physical location of the router. More precisely, this attack involves completing the 7 steps outlined in Figure 1. Since the specifics of the attack is highly dependent on browser behavior, the first step is to fingerprint the browser. In the second step the local network is scanned to find the router. The third step is to fingerprint the router to determine what type of authentication the router is using and what default password to try. Fingerprinting the router also lets us choose an XSS payload to inject or a page to frame. The fourth step is logging into the router. This is the most challenging step for both forms of authentication (HTTP authentication and web-form authentication): with HTTP authentication the browser pops up a dialog. With web authentication it is difficult to test if authentication succeeded. In the fifth step we inject the XSS payload or frame the victim router page. In the sixth step the WPA key and the wifi mac address is sent back to the attacker and finally, in the last step, the Mozilla "Location-Aware Browsing" protocol is exploited to geo-localize the router.



WPA breaker Demo

```
0 : Fingerprinting browser
0 : Browser name : Firefox 3.6.6
1 : Scanning network for router ip
1 : Router found : 192.168.2.1
1 : Authentication type : Web
2 : Fingerprinting router
2 : Router : SMC Network SMCWBR14-G2
3 : Authenticating : using smcadmin password
4 : Injecting payload
4 : Wait for the router to reboot
5 : Extracted :
mac{00:21:91:F8:48:3A} key{Ke9Mai6eGAnhe74K}
6 : Geolocalizing using google localisation api
```

Hawthorne Ave 94301 Palo Alto California US

(37.4484408, -122.1662286)

Figure 2: Our router attacking tool

A screenshot of our tool that implements all the steps above in an automated fashion is visible in figure 2. Additionally a video demo can be found here: `http://ly.tl/v1`.

We tested feasibility of the attacks on routers from the following brands: *Belkin, Netgear, D-link, Linksys, Buffalo Zyxtel, SMC, TrendNet*. We found that all of them are vulnerable to drag and drop framing attacks and that at least 4 of them can be exploited directly by XSS. Attacking routers in an automated fashion can be difficult due to strictly enforced same origin policies and router filtering rules. We tried numerous approaches and attacks for each of the seven steps we outlined in Figure 1; often with little success. Floods of security patches and fixes in modern browsers, Silverlight and Flash forced us to be more creative. For instance, the current version of Flash (10.1) makes it almost impossible to probe and attack a local network due to a security conscious socket and web service API. Similarly, we tested the idea of using the UPNP protocol to extract or change router passwords; it turns out that on many router we tested the UPNP component was disabled by default. Even when UPNP is enable, the services provided do not allow to access Wifi parameters.

2.1 Dealing with Browser Behavior

Knowing specific browser behavior of the victim is crucial in conducting this attack as it will determine how the attack need be conducted.

Firefox (FF)

FF is currently the best browser for attacking LAN routers (from the attacker's point of view). First, it allows for XHR requests to arbitrary domains. This lets us implement an efficient port scanner (rather than rely on the classic onError image approach). Second, we exploit two bugs we found which make steps 2, 3, and 4 much easier. More precisely, the first bug we found allows us to detect whether the router is using HTTP authentication or a web form authentication. The second bug allows us to authenticate on a router that have a basic authentication without triggering the classic HTTP authentication popup even if we are supplying the wrong password.

Internet Explorer 8 (IE)

IE appear to be the hardest browser to exploit since it strictly disallows cross-domain XHR unless the script is from *file://* origin. Obviously this forces us to implement port scanning using images, which is unreliable and slow. The real problem with IE is with routers that use HTTP basic authentication. If an incorrect password is supplied in the URL, browsers will prompt a login popup. For an automated attack this is inconvenient; and many of older tricks to circumvent this has been fixed (such as loading sites through a CSS background-image url). Add to this that since November 2007 every version of IE (6, 7, and 8) does not support URLs containing a username and password [9]. As far as we know there is no way to exploit IE to automatically log in to a router using HTTP authentication. In summary, we are only able to use IE for an automated attack only if the router uses web authentication.

Chrome

The Firefox bug which allowed us to fingerprint a router using images without triggering a popup works on Chrome as well. In fact, most of the methods we used on Firefox also work on chrome.

2.2 Finding the router

To locate a router, our port scanner looks for all the probable IP addresses in the 192.168.* range. More precisely our POC scans the addresses in 192.168.*.1 and 192.168.*.254. We can easily extend this to all the other private IP address ranges defined in RFC 1918, but routers use a limited set of default IP addresses. When *XHR*s (XMLHTTPRequests) are used to scan the network we take advantage of their asynchronous nature to span 100 requests at once.

On IE, the port scanning is performed by spanning 100 invisible images and timing how long it takes before each image event handler onerror is triggered. Every router tested takes less than 3 seconds to answers whereas the image timeout is 12 seconds which makes this technique possible. This is a rudimentary and slow, but it works.

2.3 Fingerprinting the router

When an IP is found, we perform a series of tests to identify the type of router. This tells us what default passwords to try. We start first by scanning the router to see if ports other than the web interface(80) are open. Due to the Firefox bug mentioned earlier we have the ability to conveniently tell if a router is using HTTP Basic authentication and if a candidate password succeeded. In Chrome we take the conservative approach of brute forcing the HTTP authentication with all known default passwords before testing to see if it succeeded. We can do so by sending authentication requests which will not notify the user of a failed attempt by exploiting a bug we found in Chrome.

2.4 Login to the router

We deal with HTTP authentication by either trying all the probable passwords (Chrome) or exploiting a bug (Firefox). For routers that do authentication using a we form, we found that It is possible to authenticate to all of them because they are susceptible to cross site request forgery (CSRF) attacks. However the real difficulty of dealing with web based authentication is not in sending the request but rather in detecting whether or not we are successfully authenticated because of the same origin policy. Until now the standard approach to detect if a login was successful was based on timing attacks [3]. There are other ways to do this more reliably. The first one, *Cross site Url Hijacking* (XSUH) [5], discovered recently (May 2010), leverages the fact that Firefox error catching discloses the URL responsible for the error. One SMC router we tested performs a redirection when the user is successfully logged in. Therefore, by triggering a fetching error and subsequently looking at the returned URL we can tell if we were able successfully log in. However, the most reliable technique to detect if the user is successfully logged in or not is a new technique we call ”frame leak attack” (FLA) (see section 4). This technique leverages Stone’s [14] observation that one can combine double framing and a hash-tag to detect if a page contain a specific element or not. To test if the user is logged in or not, we navigate the inner IFRAME to a page and hashtag only available while logged in. If the scrollbar exist the user is logged in, otherwise not. Note that this attack works on every router since none of them deploy frame busting defenses.

2.5 Stealing WIFI information

Once we are logged into the router, we have, as depicted in figure 1, two options of acquiring needed WIFI information: we can either use a drag-and-drop framing attack or an XSS injection attack.

XSS injection

If the router is vulnerable to XSS attacks, the most straightforward way to steal the Wifi information is to inject code via a CSRF and capture information. The XSS payload will do the following: First, open an IFRAME to a page containing WPA key or MAC address (or other useful information). Since the payload operates in the same origin as the framed page it can freely read script and DOM data from it. This data is then sent back to the attacker using a cross-domain form request.

Framing attack

When it is not possible to inject an XSS payload, we extract needed information using a drag and drop attack. This possible on all routers we tested due to the complete lack of framing defenses. The drag and drop framing attack introduced by Stone at the Blackhat 2010 [14] allows an attacker to extract data from a framed page by abusing the HTML 5 drag-and-drop capabilities[1]. This attack currently works in every major browsers. The downside of this attacks is that it requires some social engineering to get the user activate the drag.

2.6 Geolocalization

Once in possession of the MAC address, geolocalization can be done using Mozilla "Location-Aware Browsing" protocol. The idea of geo-localizing router using XSS was first demonstrated by Samy [8] in April 2010. Being able to geolocalize a network from its MAC is an important advance in offensive tools since it allows the attacker to know the exact location of the victim.

[1]A demo of the drag and drop attack is available from here : ly.tl/rt1

3 Defeating HTTPS via cache injection

In this section we demonstrate how an attacker can compromise multiple HTTPS sessions by injecting a malicious javascript library into the victim browser cache.

3.1 How cache injection works

A cache injection works as follows: In the first step the user connects to an insecure location such as a coffee shop. Then attacker executes a man in the middle attack against the user in order to inject a malicious external javascript in one of the webpage viewed by the user. The malicious javascript library is served with the correct HTTP headers to ensure that it will be cached by the victim browser. Once the library is successfully cached, every HTTPS connection to a page that includes the targeted library will be compromised until the victim clears his browser cache. The library will be fetched from the cache that contains the malicious version injected by the attacker. Moreover connecting to a page that includes the injected javascript library, will never trigger a browser warning because as the browser blindly trusts its cache content.

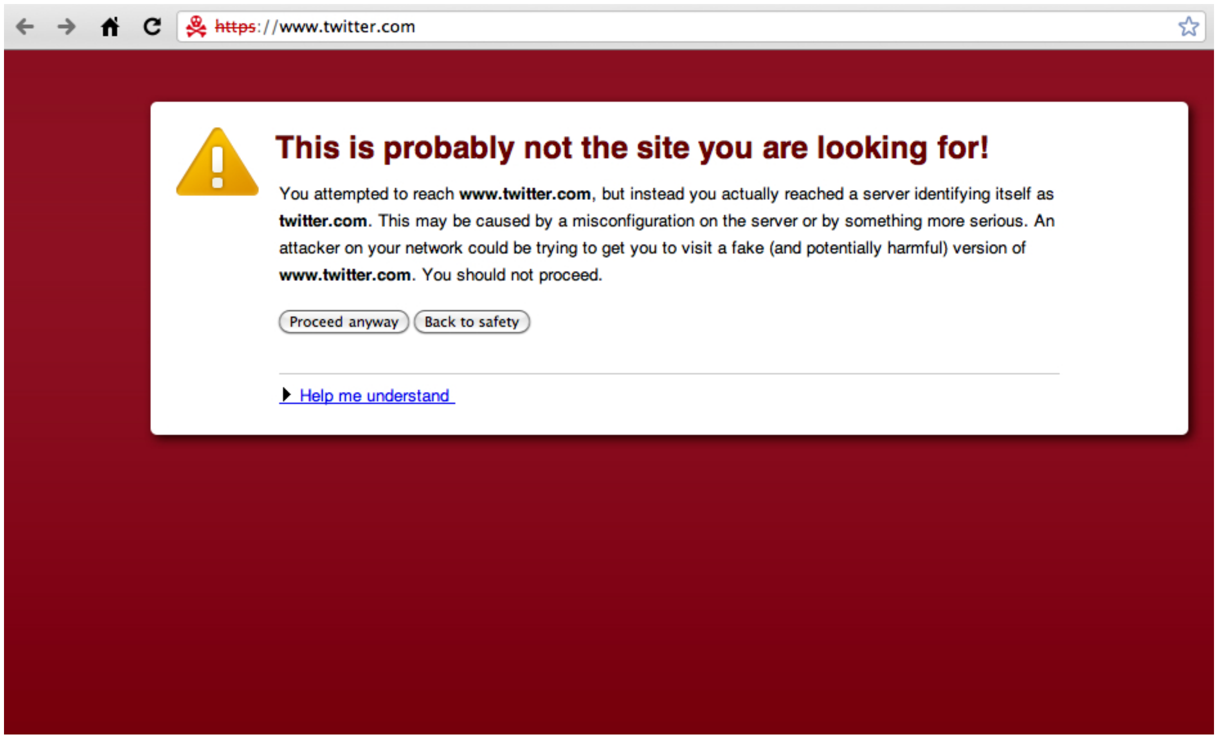
3.2 Why cache injection are dangerous

Injecting a malicious library is a very efficient attack because of the following three reasons: first cache injection attack works on every major browsers, as caching is one of the major mechanism used to display web pages faster. Even smartphone browsers obey at least to the expiration date headers directive[1]. Secondly it can be used to target the vast majority of web sites. While crawling the Alexa top 10000 websites, we found that among those who use SSL, at least 60% of them include at least one external javascript library in their main page which makes them vulnerable to caching attacks. Finally the attacker can leverage the fact that many web sites use external javascript libraries such as Google Analytics and jQuery to target multiple web sites at once.

3.3 Exploiting Browser UI Inconsistency

The only defense against cache injection attacks is the warning displayed by the browser when a SSL certificate is invalid (see figure 3). However this is not a big hurdle for the attacker because of the following two reasons: First as emphases by the recent measurement done by Comodo that shows that around 50% of websites have an invalid certificate[4], users are used to click through these warnings. Even popular websites such as `https://www.twitter.com` and `https://www.youtube.com` display a warning because of cname mismatch. Secondly and even more deadly, we were able to create situations where the standard SSL warning was not displayed properly.

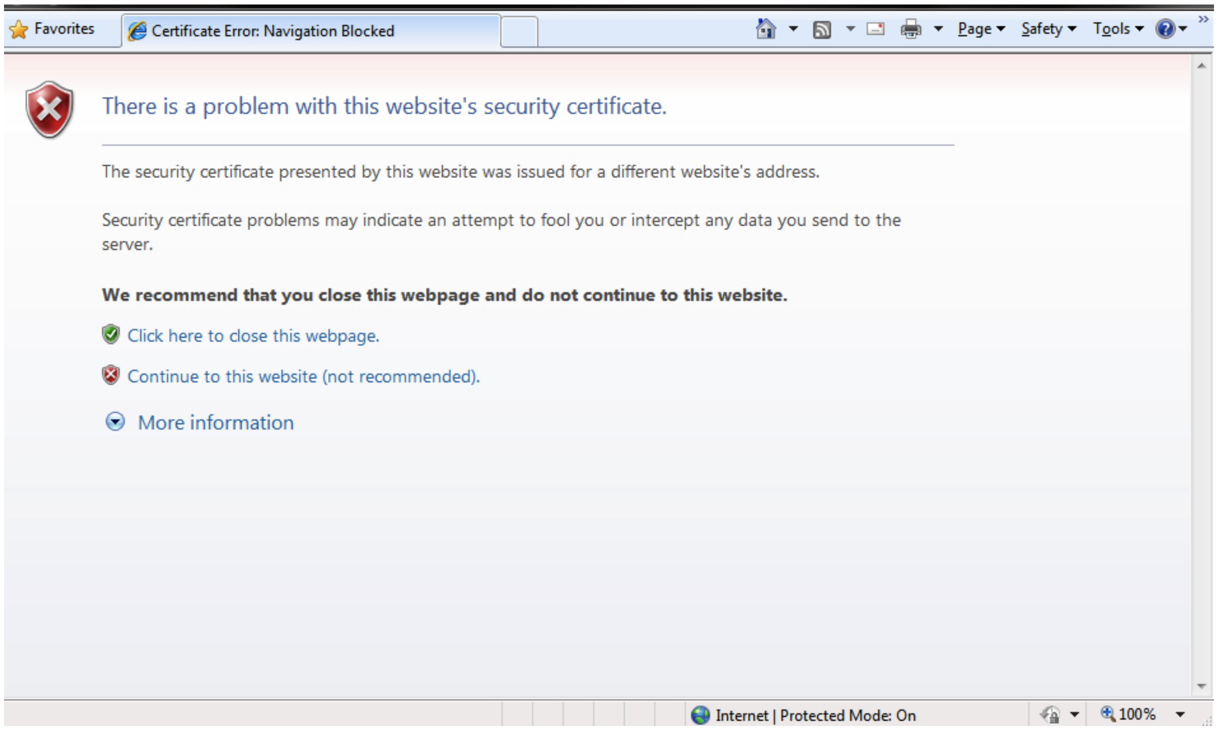
Internet Explorer.



This is probably not the site you are looking for!

You attempted to reach **www.twitter.com**, but instead you actually reached a server identifying itself as **twitter.com**. This may be caused by a misconfiguration on the server or by something more serious. An attacker on your network could be trying to get you to visit a fake (and potentially harmful) version of **www.twitter.com**. You should not proceed.

Figure 3: Chrome SSL warning



There is a problem with this website's security certificate.

The security certificate presented by this website was issued for a different website's address.

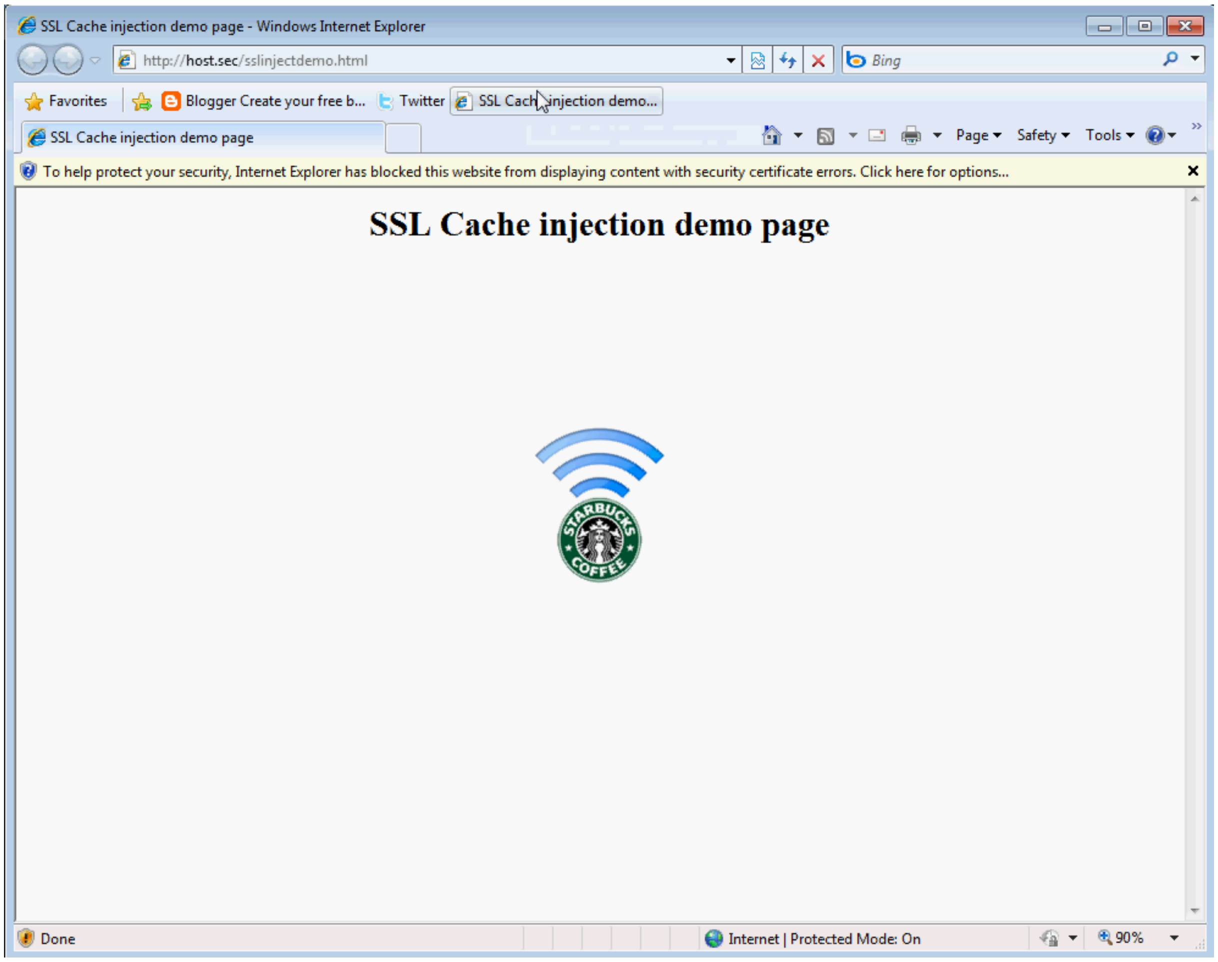
Security certificate problems may indicate an attempt to fool you or intercept any data you send to the server.

We recommend that you close this webpage and do not continue to this website.

Favorites
Certificate Error: Navigation Blocked
Page
Safety
Tools
Click here to close this webpage.
Continue to this website (not recommended).
More information
Internet | Protected Mode: On
100%

Figure 4: Internet Explorer 8 standard SSL warning

Figure 4 shows Internet Explorer 8 standard SSL warning . Figure 5 shows what happen when the bad certificate is embedded into an invisible iframe. As one can see in this case the user only see a small yellow warning bar on the top that provides no information about the content that has been blocked. It is likely that seeing this kind of warning while accessing the startup page of a hotspot won't trigger the user suspicion. To make the matter worst, accepting one time to display the blocked content will in fact cache multiple javascript libraries from different origins at once. This allows the attacker to poison all the most popular shared libraries at once which makes cache injection attacks even more deadly when the user is using Internet Explorer.


SSL Cache injection demo page - Windows Internet Explorer
http://host.sec/sslinjectdemo.html
Bing
Favorites
Blogger Create your free b...
Twitter
SSL Cache injection demo...
SSL Cache injection demo page
Page
Safety
Tools

To help protect your security, Internet Explorer has blocked this website from displaying content with security certificate errors. Click here for options...

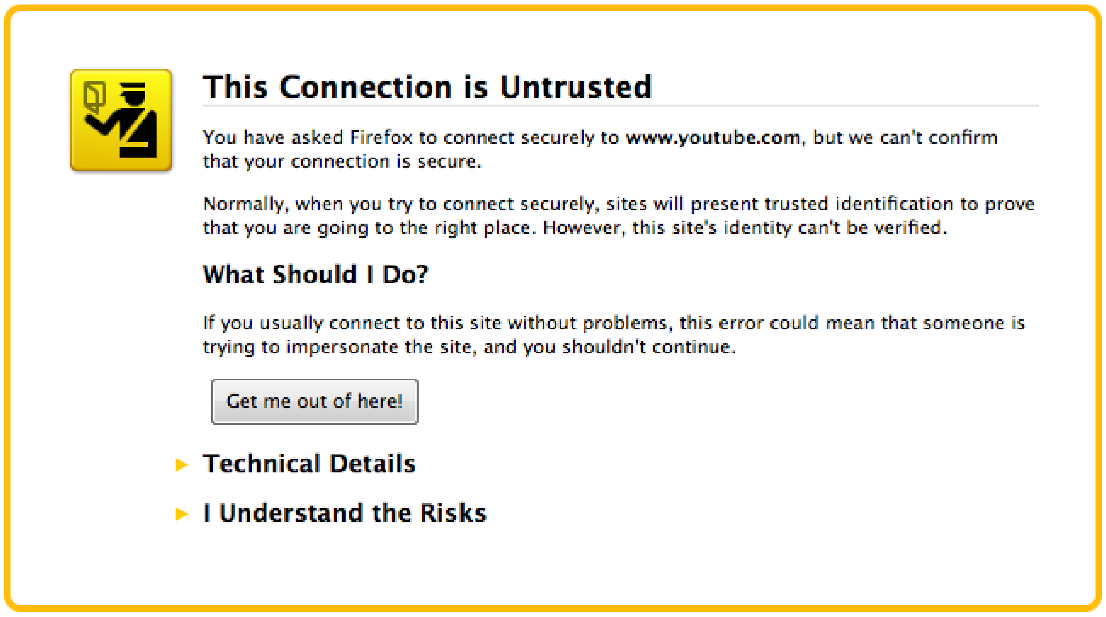
SSL Cache injection demo page

Done
Internet | Protected Mode: On
90%

Figure 5: Internet Explorer 8 corner case

Firefox.

Figure 6 displays what is the standard SSL warning for Firefox 3.6. While we weren't able to find a case where the warning was not displayed properly, there is still an interesting case that the attacker can exploit. As visible in figure 7 when a page with a bad certificate is loaded into an iframe, the warning is displayed inside the iframe as well. Since the warning has no clickjacking protection, the attacker can use a clickjacking attack to reduce the warning by pass from three



This Connection is Untrusted

You have asked Firefox to connect securely to **www.youtube.com**, but we can't confirm that your connection is secure.

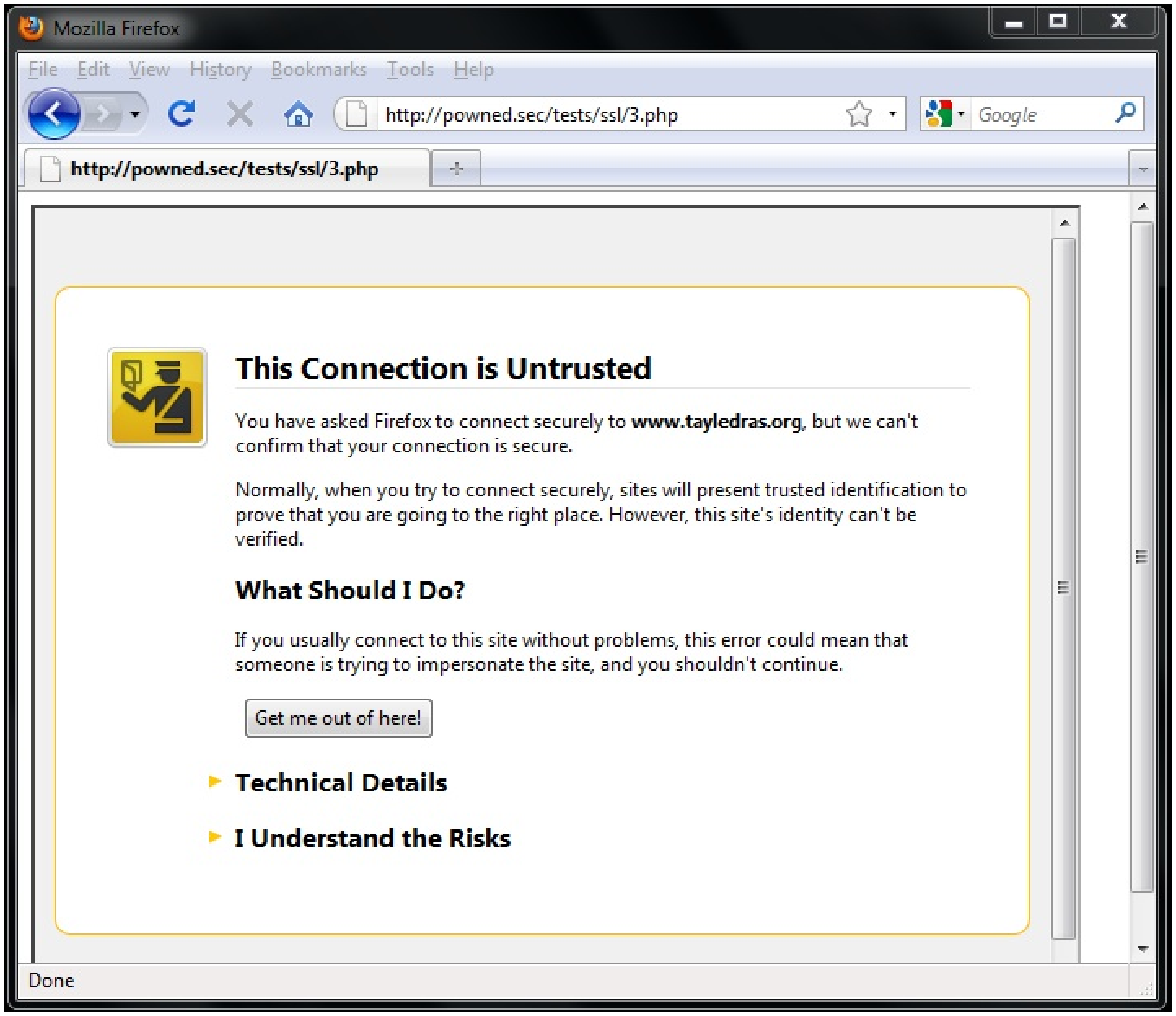
Normally, when you try to connect securely, sites will present trusted identification to prove that you are going to the right place. However, this site's identity can't be verified.

What Should I Do?

If you usually connect to this site without problems, this error could mean that someone is trying to impersonate the site, and you shouldn't continue.

Figure 6: Firefox standard SSL warning

clicks to one. The victim will only see the last popup window which only asks to accept a certificate without any warning (Figure 8).


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://powned.sec/tests/ssl/3.php
Google
http://powned.sec/tests/ssl/3.php
Done

This Connection is Untrusted

You have asked Firefox to connect securely to **www.tayledras.org**, but we can't confirm that your connection is secure.

Normally, when you try to connect securely, sites will present trusted identification to prove that you are going to the right place. However, this site's identity can't be verified.

What Should I Do?

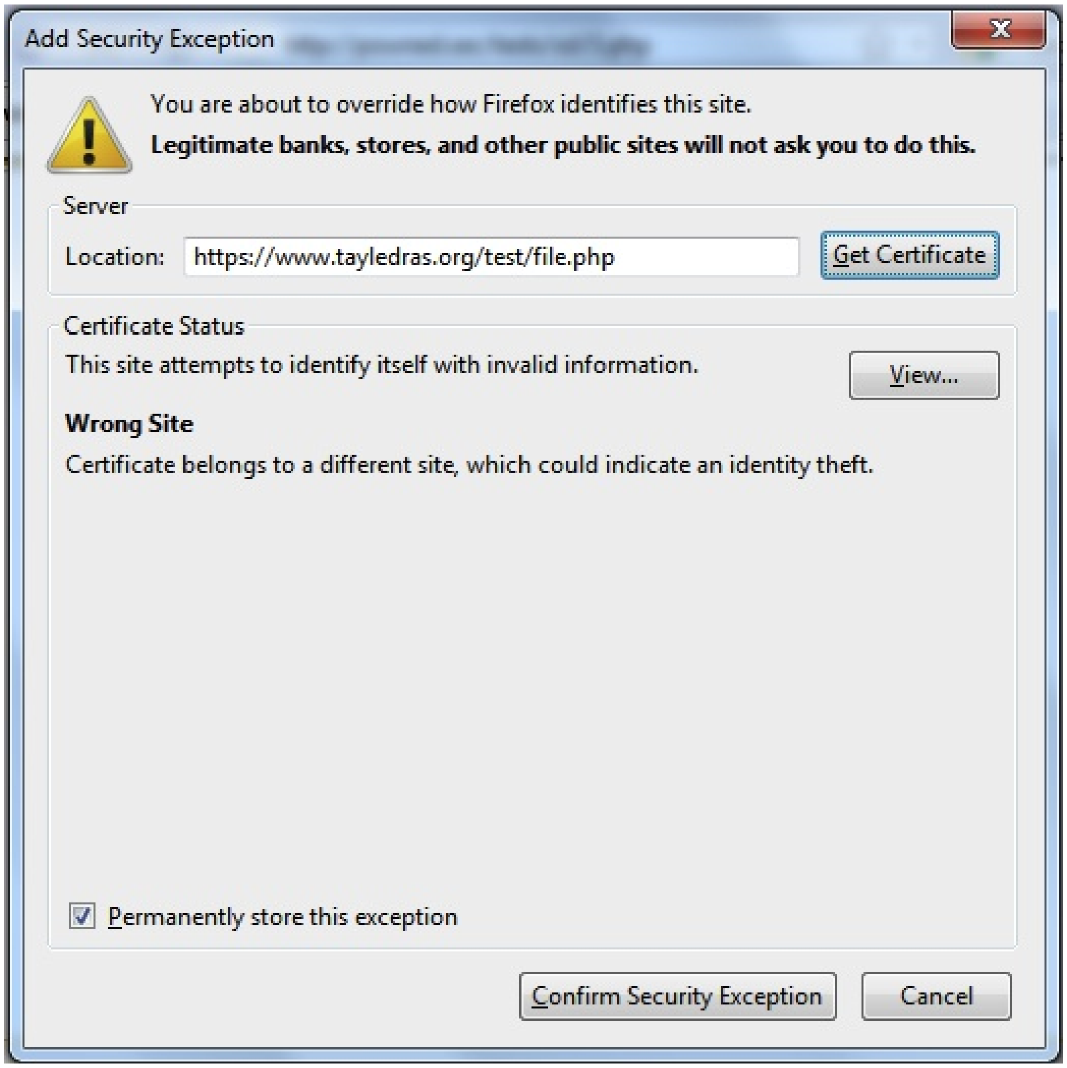
If you usually connect to this site without problems, this error could mean that someone is trying to impersonate the site, and you shouldn't continue.

Get me out of here!

Technical Details

I Understand the Risks

Figure 7: Firefox standard SSL warning Framed



Add Security Exception

You are about to override how Firefox identifies this site.

Legitimate banks, stores, and other public sites will not ask you to do this.

Server

Location: https://www.tayledras.org/test/file.php

Certificate Status

This site attempts to identify itself with invalid information.

Wrong Site

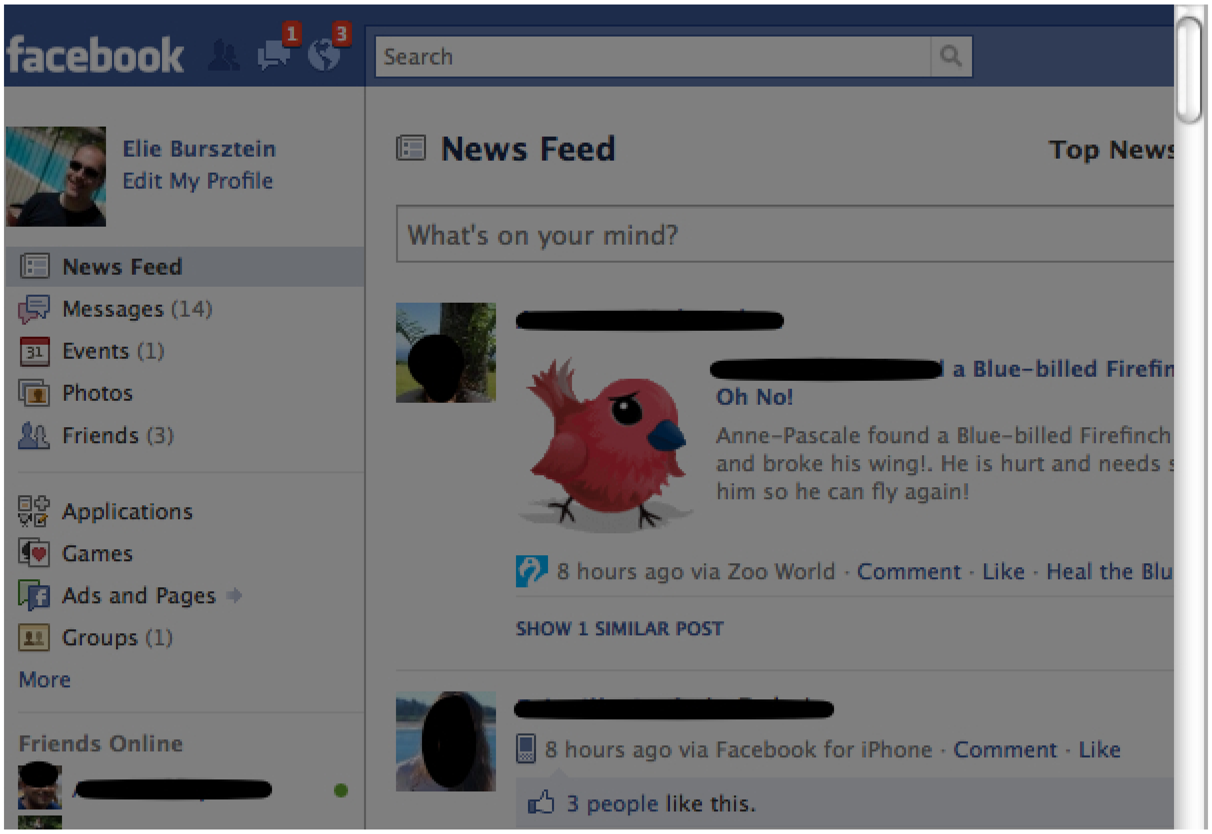
Certificate belongs to a different site, which could indicate an identity theft.

[✓] Permanently store this exception

Figure 8: Firefox: Remaining popup after the clickjacking attack warning

4 Attacking Facebook with Frame leak attack

A Frame Leak Attack (FLA) leverages the idea of reading the scrollbar position to know if a hashtag is present in a framed page presented by Stone [14] to conduct privacy attack: the attacker infers private information about the framed context by looking at the presence of certain hashtags. In the previous section we demonstrated how we used this scrolling attack to identify if we were able successfully login to a router or not. In this section we demonstrate that FLA has a broader and potentially more dangerous use-case. FLA can be used to mount privacy based attack when there is no frame busting code (or broken frame busting code). To illustrate how FLA can be used to defeat privacy, we show how w used it to attack Facebook.



[redacted]

[redacted] a Blue-billed Firefin

Oh No!

Anne-Pascale found a Blue-billed Firefinch and broke his wing!. He is hurt and needs s him so he can fly again!

8 hours ago via Zoo World · Comment · Like · Heal the Blu

SHOW 1 SIMILAR POST

[redacted]

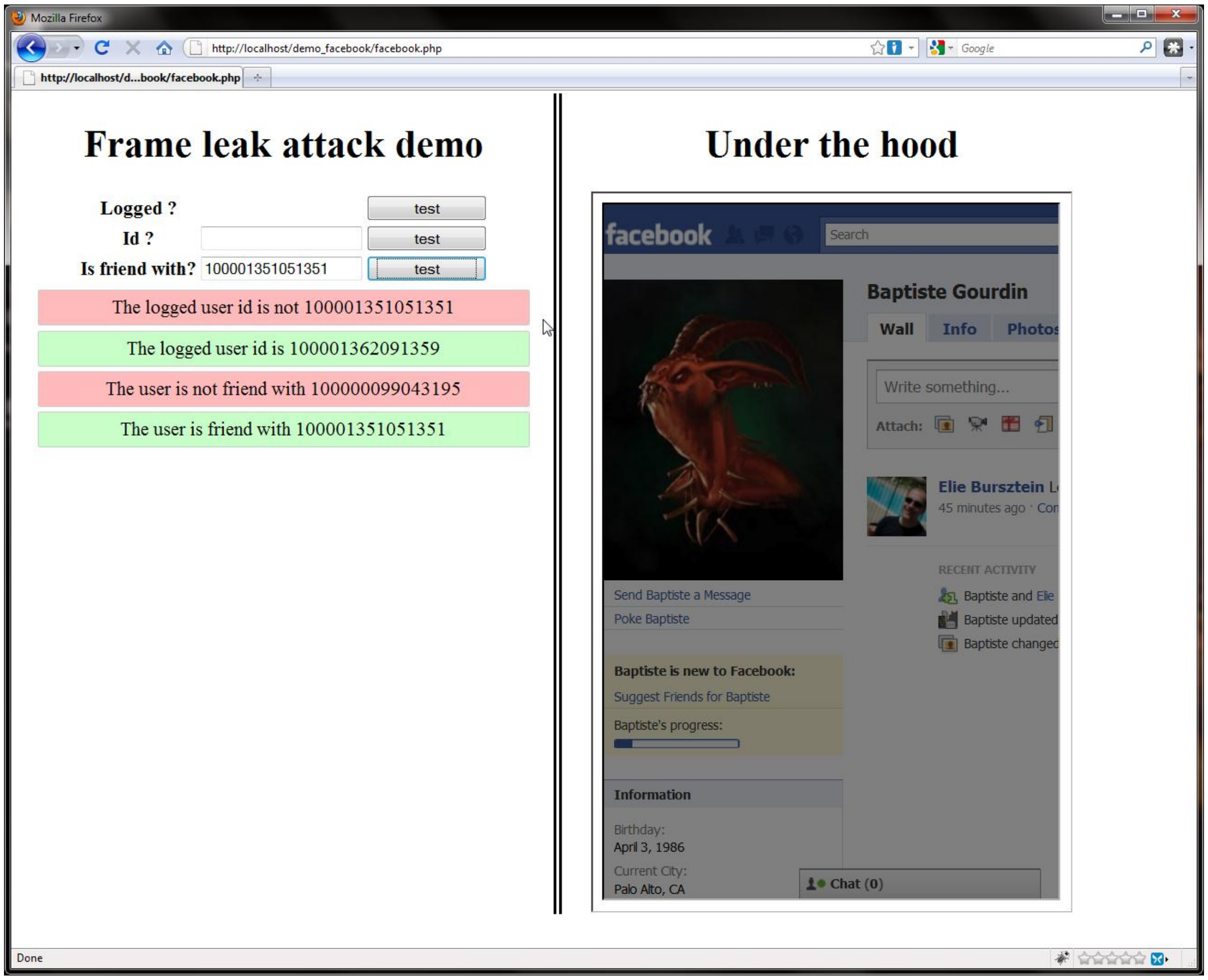
8 hours ago via Facebook for iPhone · Comment · Like

3 people like this.

[redacted]

Figure 9: Facebook clickjacking Black Div defense

As shown in figure 9 Facebook's clickjacking defense is based on the idea of overlaying a semi-transparent DIV on top of the page. Users can see their session-based content but not interact with it. When the DIV is clicked, Facebook attempts to frame bust using standard technique. While this defense works reasonably well against standard click-jacking attacks, it is unable to defend against privacy attack using FLA. Despite the overlay, browsers will automatically scroll to the specified hashtag. The attack works as follow: When the victim visits the attacker page, an invisible double iframe with a very small width and height is displayed. The inner iframe is redirected to Facebook with a specified hashtag: `#pagelet_intentional_stream`. If the user is logged-in the scrollbar will move. This movement can be dynamically read and serve as an indicator to the attacker of the user's session status. We extended this attack to many scenarios; for instance the attacker can test if the victim is a given the Facebook user by navigating the invisible iframe to a vanity-name or profile id url and note if hashtag `profile_action_poke` is present. A screenshot of our attack tool is visible in figure 10 and a video of the attack is available at `http://ly.tl/v2`.



Frame leak attack demo

Logged ?		test
Id ?		test
Is friend with?	100001351051351	test

The logged user id is not 100001351051351

The logged user id is 100001362091359

The user is not friend with 100000099043195

The user is friend with 100001351051351

Under the hood

Baptiste Gourdin

Write something...

Elie Bursztein

45 minutes ago

Baptiste and Elie

Baptiste updated

Baptiste changed

Send Baptiste a Message

Poke Baptiste

Baptiste is new to Facebook:

Suggest Friends for Baptiste

Baptiste's progress:

Information

Birthday:
April 3, 1986

Current City:
Palo Alto, CA

Figure 10: Facebook frame leak attack tool

In this section we introduce TapJacking attacks, a clickjacking attack that leverages the accessibility features implemented in mobile browsers to mount powerful attacks on smart phones.

5.1 TapJacking Safari on Iphone

The iPhone Safari browser supports all the basic functionality to pull off a classic clickjacking attack: transparency and IFRAMEs. Transparency is supported through the CSS opacity attribute in Safari Mobile. However, extra features of the iPhone make the attack far more dangerous.

Zooming

On desktop browsers an attacker can ensure that the user clicks at the right place in the victim IFRAME. One approach is to consistently move the IFRAME into place after a MouseMove event is detected so that the mouse always points to the button that the attacker wants clicked. Since this method is more difficult to pull off on the iPhone we instead use the iPhone's zooming functionality. Recall that scaling on smart-phones is often done via the viewport meta tag:

```
<meta name = "viewport"  content = "width = 320,
initial-scale = 10, user-scalable = no">
```

In this example the initial scaling of the entire viewport is set to 10 (maximum). At this level of zoom, any regular button will cover the entire width of the screen. By putting this enlarged button in an IFRAME, the clickjacking attack becomes much more efficient (considering the ”tappable” area is very large). Interestingly, scaling properties of the the top frame takes precedence over those of framed sites. Most of the popular sites, such as Facebook and Twitter, have a very constrained interface when it comes scaling, but this can be mitigated by framing them. Figure 11 shows an example where the Twitter publishing button has been enlarged in a transparent IFRAME. To further ensure the user is constrained to click the targeted area, we can disable any further scaling by setting the user-scalable attribute to 0.

Hiding or faking the URL Bar

An important difficulty with clickjacking on the desktop is making the browser's address bar point to a legitimate-looking URL. This problem is not an issue with TapJacking since on the phone an attacker can cause the address bar to disappear. This following code hides the URL bar out of sight as soon as the site is loaded by scrolling the URL navigation bar out of the visible window:

```
<body onload="setTimeout(function()
   { window.scrollTo(0, 1) }, 100);">
</body>
```

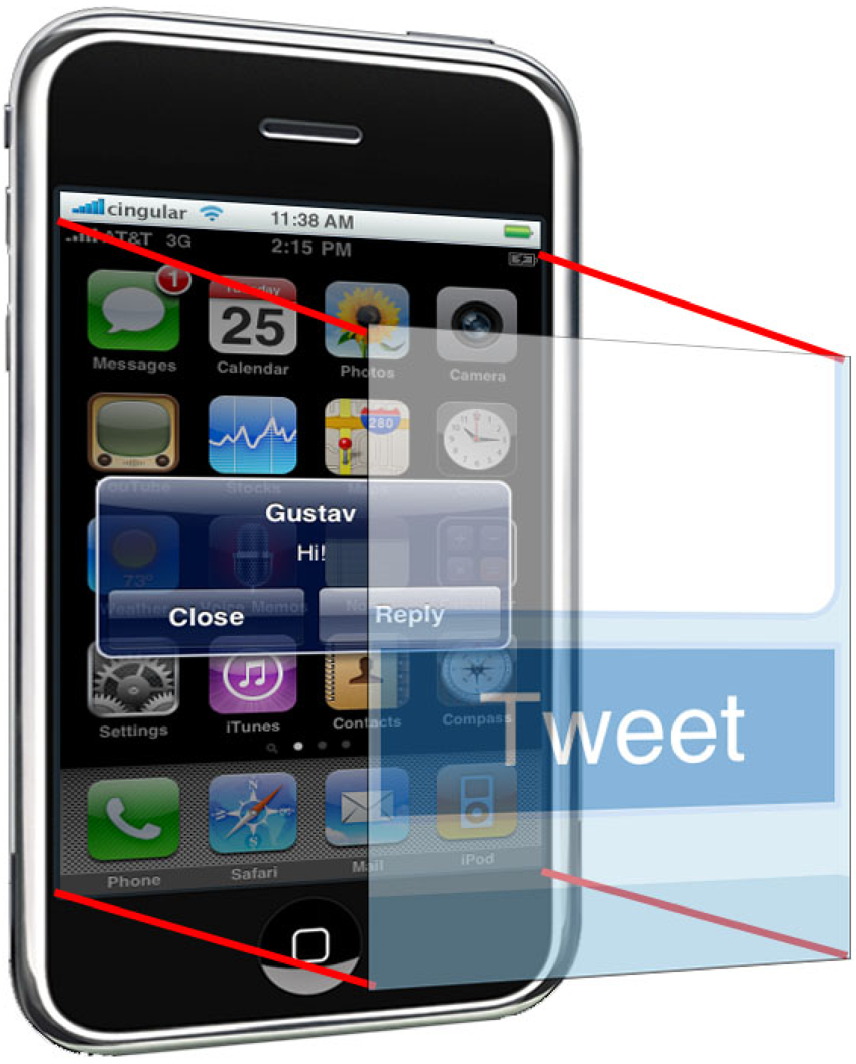
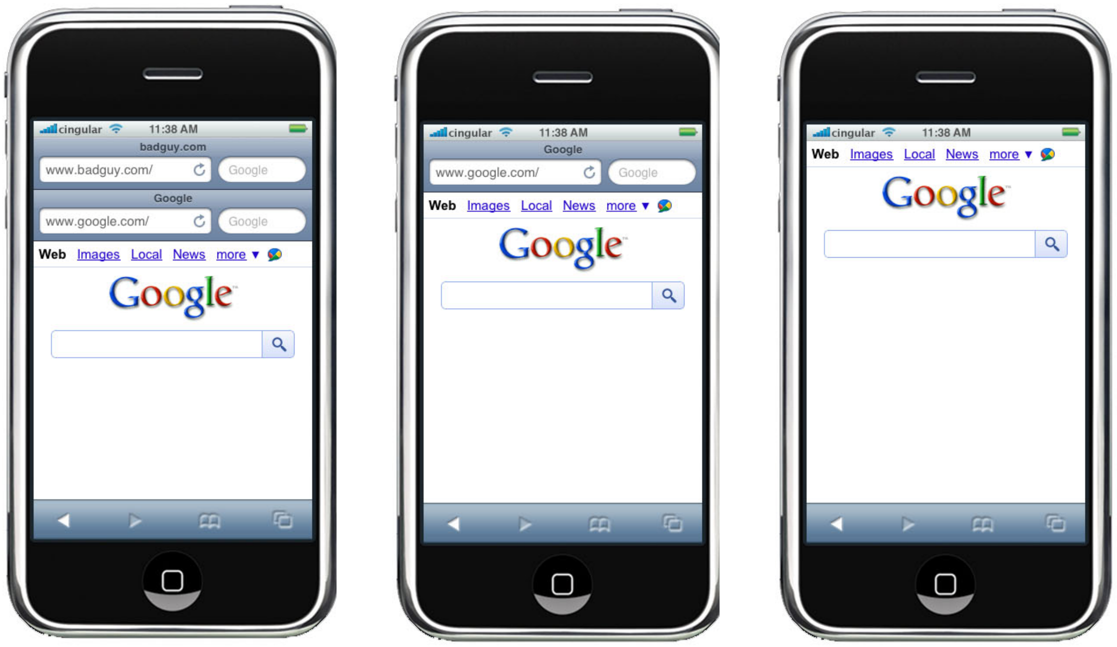


Figure 11: Tapjacking Twitter with a zoomed button



The left figure shows the fake address bar under the real one. The middle figure shows the fake URL replacing the real one. The right figure shows no URL bar.

Figure 12: Faking the URL bar

An attacker can embed a picture of a fake URL address bar in the framing page thereby making the page appear to come from a legitimate site. Figure 12 gives an example.

Abusing the shared screen real-estate

The tight integration and sharing of screen real-estate between the browser and IPhone UI supports another way to strengthen tapjacking. The idea is to create a page that masquerades as a well known phone behavior, unrelated to the web. For example, Figure 11 shows what appears to be an incoming SMS text message notification. However under the hood it is not the SMS application but a webpage rendered as a native app look-a-like. Because the user knows he needs to click either ”Close” or ”Reply” upon receiving a text message notification, he will click without second thought. However in this case clicking won't acknowledge the text message but instead publish a tweet.

Strengthening Tapjacking by turning off navigation and using dynamic scrolling

It is possible to prevent any touch gesture on a tapjacking page using the touchMove event to disable the default behavior. This is done by calling the function preventDefault as shown in the code below:

```
function touchMove(event) {
    event.preventDefault();
}
```

Furthermore it is possible to dynamically position the viewport by using the standard JavaScript function `window.scrollTo(x,y)`. This helps the attacker dynamically position the viewport window just above the targeted button.

Session handling

Without a session to hijack clickjacking attacks are not very interesting. Sessions identifiers are often stored in “session cookies.” On desktop browsers, these session cookies expire when the user closes the browser. This is not true on the iPhone as the session persists when Safari Mobile is closed. This helps the attacker since sessions lay dormant for possible clickjacking attacks. A malicious link can be sent to the user in an e-mail causing the browser to load a live session.

While analyzing the Alexa Top 100 top sites, we noted that some “mobile cookies” expire further in the future than their desktop counterparts. Presumably this is designed to minimize the number of times that the user needs to login on a cell phone. Again, these longer lived sessions help the attacker.

Defenses: the X-Frame-Options HTTP header

This header instructs the latest version of all main browsers (other than Firefox) not to render the page in a sub-frame. Both the iPhone 3.0's Safari Mobile and the Android 2.1 browser support this header. The header should be added whenever the user agent is one of these browsers. When used, this header provides adequate protection from framing attacks.

5.2 Other mobile browsers

The Android Browser

We also tested the Android browser on a Motorola Droid. All the tapjacking techniques we outlined in previous iPhone section are possible on the Android browser. Support for IFRAMEs, opacity changes, scaling, viewport meta tags, makes the Android browser a prime target for tapjacking.

Opera Mini

Opera Mini uses a proxy-rendering system to display webpages faster. Although Opera Mini has growing JavaScript and CSS support we conclude that a traditional clickjacking attack is not possible on the Opera Mini (we tested on version 5.0.5 on the iPhone). Although IFRAMEs are supported, changing their opacity and size reliably is not. This makes the classic approach to clickjacking difficult since we cannot effectively redress clickable UI of the target page.

6 Related Work

Niu et al. [10] previously used the iPhone's browser scrolling mechanism to design a phishing attack where the address bar scrolls off the screen and a fake address bar is presented. Here we use a similar mechanism as one step in framing attacks. Clickjacking attacks on the iPhone were mentioned in [11], although these attacks used a specific bug in the iPhone browser. The bug was fixed long ago (iPhone OS 2.2) and is no longer an issue. Our tap-jacking attack uses main stream features of the browser that are unlikely to be changed. In 2006 Stamm et al. [13] showed that routers are vulnerable to cross site request forgeries that can result in a take-over of a home or corporate network. These attacks are quite difficult to mount on modern routers as explained earlier. Bojinov et al. [2] show that many web sites embedded in consumer electronics are vulnerable to web attacks. However, they focus mostly on specific application logic errors where as we focus on generic framing attacks that work against a large set of routers.

7 Conclusion

In this paper we demonstrated that attackers can make secure communication protocols irrelevant by targeting their data storage mechanism. We illustrated the weakness of current storage mechanisms by showing the following four kind of attacks: first, we showed how an attacker can remotely locate and break into a Wifi network by crafting a malicious web page that targets its access point. Secondly, we demonstrated how an attacker can inject a malicious library that is capable of compromising subsequent SSL sessions by leveraging the fact that websites trust external javascript libraries, such as Google Analytics. We then described how to easily fool the user into accepting this malicious javascript library by exploit- ing browser UI corner cases. Next, we introduced frame leak attacks that are capable of extracting private information from the website (and not from the user) by leveraging the recent scrolling technique of Stone. Our frame leak attacks defeat click-jacking defenses that have previously been considered secure. In addition, we illustrated how a frame leak attack works by demonstrating how to use it to extract Facebook profile information, bypassing Facebooks framebusting defenses in the process. Finally, we developed a new attack called tap-jacking that uses features of mobile browsers to implement a strong click-jacking attack on phones. We show that tap-jacking on a phone is more powerful than traditional clickjacking attacks on desktop browsers, and thus imply smartphones should not be considered a secure form of data storage.

[1] Yahoo! User Interface blog. Mobile browser cache limits: Android, ios, and webos. http://www.yuiblog.com/blog/2010/06/28/mobile-browser-cache-limits/, June 2010. 12

[2] Hristo Bojinov, Elie Bursztein, and Dan Boneh. XCS: cross channel scripting and its impact on web applications. In *CCS '09: Proceedings of the 16th ACM conference on Computer and communications security*, pages 420–431. ACM, 2009. 25

[3] A. Bortz, D. Boneh, and P. Nandy. Exposing private information by timing web applications. In *Proc. of WWW'07*, pages 621–628, 2007. 10

[4] Comodo. Ssl market share. http://www.whichssl.com/ssl-market-share.html, June 2010. 12

[5] Soroush Dalili. ross site url hijacking by using error object in mozilla firefox. http://packetstormsecurity.org/papers/general/xsuh-firefox.pdf, May 2010. 10

[6] Gnucitzien. More advanced clickjacking – ui redress attacks. http://www.gnucitizen.org/blog/more-advanced-clickjacking-ui-redress-attacks/, 2008. 5

[7] Robert Hansen. Clickjacking. ha.ckers.org/blog/20080915/clickjacking. 4

[8] Samy Kamkar. mapxss: Accurate geolocation via router exploitation. http://samy.pl/mapxss/, January 2010. 11

[9] Microsoft. Internet explorer does not support user names and passwords in web site addresses (http or https urls). support.microsoft.com/kb/834489, Nov 2007. 9

[10] Yuan Niu, Francis Hsu, and Hao Chen. iphish: Phishing vulnerabilities on consumer electronics. In *Proc. of UPSEC*, 2008. 25

[11] John Resig. Clickjacking iphone attack, 2008. ejohn.org/blog/clickjacking-iphone-attack. 25

[12] Gustav Rydstedt, Elie Bursztein, Dan Boneh, and Collin Jackson. Busting frame busting: a study of clickjacking vulnerabilities at popular sites. In *IEEE Oakland Web 2.0 Security and Privacy (W2SP'10)*, 2010. seclab.stanford.edu/websec/framebusting. 5

[13] Sid Stamm, Zulfikar Ramzan, and Markus Jakobsson. Drive-by pharming. In *Proc. of ICICS*, pages 495–506, 2007. 25

[14] Paul Stone. Next generation clickjacking. media.blackhat.com/bh-eu-10/presentations/Stone/BlackHat-EU-2010-Stone-Next-\Generation-Clickjacking-slides.pdf, 2010. 4, 5, 10, 11, 18