Framing Attacks on Smart Phones and Dumb Routers: Tap-jacking and Geo-localization Attacks

Gustav Rydstedt Baptiste Gourdin Elie Bursztein Dan Boneh*
Stanford University
{*rydstedt,bgourdin,elie,dabo*}*@cs.stanford.edu*

Abstract

While many popular web sites on the Internet use frame busting to defend against clickjacking, very few mobile sites use frame busting. Similarly, few embedded web sites such as those used on home routers use frame busting. In this paper we show that framing attacks on mobile sites and home routers can have devastating effects. We develop a new attack called tap-jacking that uses features of mobile browsers to implement a strong clickjacking attack on phones. Tap-jacking on a phone is more powerful than traditional clickjacking attacks on desktop browsers. For home routers we show that framing attacks can result in theft of the wifi WPA secret key and a precise geo-localization of the wifi network. Finally, we show that overlay-based frame busting, such as used by Facebook, can leak private user information.

1 Introduction

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 iframed the framing page can mount a variety of attacks. The most common example is clickjacking [5] 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 [4] (where the framing page places frames on top of the victim page), drag-and-drop attacks [13], and scrolling attacks [13]. We discuss these attacks in more detail 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 [11]. 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 [11] 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 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 so 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 leak private user information.

Framing attacks on mobile web sites. 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

*Supported by NSF.

deliver 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 two sites that frame bust both their main and mobile site). As a result almost all mobile sites are vulnerable to framing attacks on the phone. In Section 2 we introduce *tap-jacking*, a framing attack on mobile web sites that is far more effective than its clickjacking cousin on desktops. These attacks show that unless frame busting is used mobile sites can be 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 2.2.

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. Moreover, 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 a web mail site on the desktop, an attacker on the desktop browser can frame the mobile version of the web mail site and use the user's credential from the main site to send mail 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.

Framing Facebook. Facebook deploys an interesting frame busting defense — when framed the site places a dark transparent `div` over the page (see Figure 3). 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 3 that although this frame busting defense may protect against traditional clickjacking attacks, an attacker can still learn private information about the user's Facebook profile. Our approach makes use of Stone's scrolling technique [13]. We use *specific* hashtags on the framed page to expose private information on the page (despite the same origin policy). We give the details in Section 3.

Framing attacks on routers. 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 using a default password: *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.

2 Phone TapJacking

In this section we introduce TapJacking — a clickjacking attack that leverages the accessibility features implemented in mobile browsers. TapJacking illustrates the im-

portance of frame busting on mobile sites. We hope this section will convince more sites to do so.

2.1 TapJacking Safari on the 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. Many popular sites, such as Facebook and Twitter, have a very constrained user interface for scaling, but this can be mitigated by framing them and scaling the top frame.

Figure 1 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

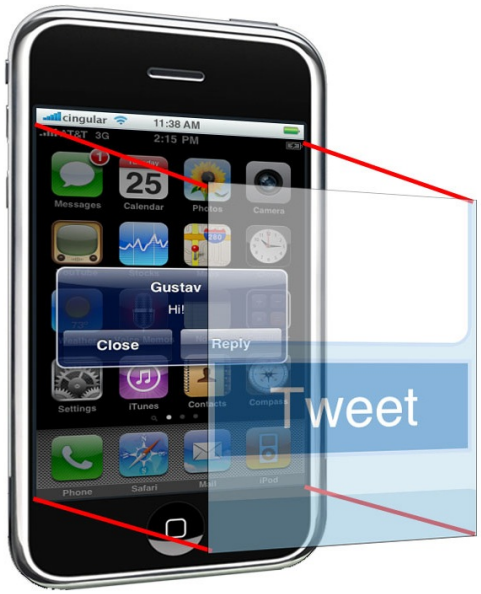

Figure 1: Tapjacking Twitter with a zoomed button

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. The 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>
```

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 2 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 1 shows what appears to be an incoming SMS text message notification. Under the hood it is not the SMS application but a webpage rendered to look like a native app. Because users know they need to

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 2: Faking the URL bar

click either "Close" or "Reply" upon receiving a text message notification, they click without a second thought. In Figure 1 clicking will not 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.

2.2 Other mobile browsers

The Android Browser. We also tested the Android browser on a Motorola Droid. All the tapjacking techniques we outlined for the iPhone are applicable to 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.

3 Framing Facebook

Figure 3 shows Facebook's clever clickjacking defense which places a semi-transparent DIV overlay on top of the page. Users can see their session content but not interact with it. When the DIV is clicked, Facebook attempts to frame bust using standard techniques. While this defense works reasonably well against standard click-jacking attacks, we show that it leaks private user information. We use a scrolling attack due to Paul Stone.

Stone [13] presented a technique for bypassing the same origin policy on pages that do not frame bust. Roughly speaking, the idea is that the attacking page loads the victim page `victim.com` in a small iframe. The attacking page then navigates the victim frame to `victim.com#test`. If the hashtag '`test`' exists on `victim.com` then the browser will scroll the victim frame to the position of that hashtag. If the hashtag does not exist the victim frame is unchanged. By reading the scroll position, the attacking page can learn if '`test`' exists on the victim page (in violation of the DOM same origin policy).

In this section we show that Stone's Frame Leak Attack (FLA) has broad applicability to defeating overlay-based frame busting, as in Figure 3. We use Facebook as an example.

The attack works as follow: When the user visits the attacker's 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 reveals the user's login status to the attacker. Similarly, the attacker can test if the victim is a specific Facebook user by navigating the invisible iframe to a vanity-name url and note if hashtag `#box_app_2305272732` is present. The attacker can test if a specific user is a friend of the victim by navigating the sub-frame to an appropriate hashtag. Many features of the user's profile can be extracted this way. A demo of this attack is available at `http://ly.tl/v2`.

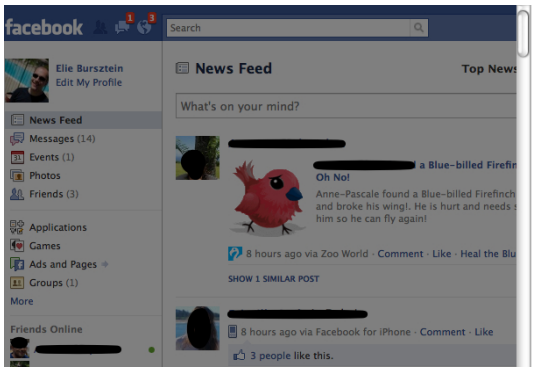

Figure 3: Facebook Black `div` defense

4 Router attacks

Many routers provide a web interface for configuration and monitoring. In this section we show that a multi-stage attack, including framing or XSS injection, enables an attacker to steal a router's WPA key and determine its physical location. The attack involves seven steps outlined in Figure 4. Since the specifics are highly dependent on browser behavior, the first step fingerprints the browser. The second step scans the local network to find the router. The third step fingerprints 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 logs 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 attacker extracts the WPA key and the wifi mac address and sends both to the attacker. Finally, in the last step, the

attacker uses Mozilla's "Location-Aware Browsing" protocol [8] to geo-localize the router.

A video of our attack that implements all the steps above in an automated fashion is available at `http://ly.tl/v1`. We tested feasibility of these attacks on routers from *Belkin, Netgear, D-link, Linksys, Buffalo Zyxtel, SMC, TrendNet*. All are vulnerable to drag and drop framing attacks and at least 4 can be exploited directly by XSS.

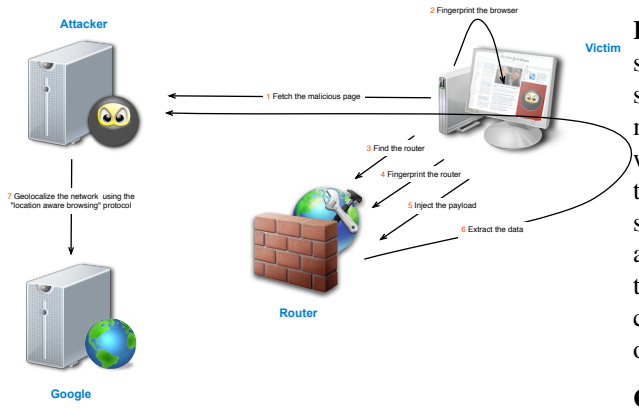

Figure 4: Router secret stealing attack flow

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 4; often with little success. Security patches and fixes in modern browsers and Flash forced us to be more creative. For instance, the current version of Flash (10.1) makes it difficult 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 in every router we tested the UPNP component was disabled by default.

4.1 Dealing with Browser Behavior

Knowing the specific victim browser is crucial for this attack.

Firefox. (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 without using the classic onError image approach (e.g. as in [2]). Second, we exploit two bugs we found[1] which make steps 2, 3, and 4 much easier. The first bug lets us detect if the router is using HTTP authentication or a web form authentication. The second bug lets us brute force HTTP authentication and run an image-based fingerprint on the router without triggering an HTTP authentication popup.

Internet Explorer 8. IE is the hardest browser to exploit since it strictly disallows cross-domain XHR unless the script is from the *file://* origin. This forces us to implement port scanning using the onError event on image tags, which is slow. The bigger problem with IE is with routers that use HTTP basic authentication. All versions of IE since November 2007 reject URLs containing a username and password [7]. As a result there is no way to exploit IE to automatically log in to a router using HTTP authentication. We are only able to use IE for an automated attack only if the router uses web authentication.

Chrome. The Firefox bug that let us fingerprint a router using images without triggering a popup works on Chrome too. Generally speaking most methods we used on Firefox also work on Chrome.

4.2 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 proof of concept 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 the 512 requests at once.

On IE, the port scanning is performed by spanning 512 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 answer whereas the image timeout is 12 seconds which makes this technique

[1] Since these bugs are still under review by Mozilla, we do not include all the details here.

possible. This is a rudimentary and slow, but works in practice.

4.3 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 in Chrome.

4.4 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 web 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 [2]. There are other ways to do this more reliably. The first one, *Cross site Url Hijacking* (XSUH) [3], 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 is a technique called "framing leak attack" (FLA) discussed in the previous section. This technique leverages Stone's [13] 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 is present the user is logged in, otherwise not. Note that this attack works on every router since none of them deploy frame busting defenses.

4.5 Stealing WiFi information

Once we are logged into the router, there are two options of acquiring needed WIFI keys: 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 keys 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 Stone's drag and drop attack [13]. The attack lets an attacker extract data from a framed page by abusing the HTML 5 drag-and-drop capabilities[2]. This works on all routers and all browsers we tested due to the complete lack of framing defenses. The downside is that it requires some social engineering to get the user activate the drag.

4.6 Geolocalization

Once the attacker has the MAC address, geolocalization can be done using Mozilla's "Location-Aware Browsing" protocol [8]. The ability to locate a network from its MAC address is an important advance in offensive tools since it allows the attacker to know the exact location of the victim. Geo-localizing a router using XSS was first demonstrated by Samy [6].

5 Related Work

Niu et al. [9] 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

[2] A demo of the drag and drop attack is available at: http://ly.tl/rt1

bar is presented. Here we use a similar mechanism as one step in framing attacks.

Clickjacking attacks on the iPhone were mentioned in [10]. These attacks used a specific bug in the iPhone browser that 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. [12] 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. We had to resort to the long sequence of steps in Figure 4.

Bojinov et al. [1] show that many web sites embedded in consumer electronics are vulnerable to web attacks. They focus mostly on specific application logic errors where as we focus on generic framing attacks that work against a large set of routers.

6 Summary and recommendations

This paper discusses a significant vulnerability in mobile web sites that is easily corrected by including frame busting in these sites. Mobile web sites that do not use frame busting are vulnerable to tap-jacking and expose their users to unnecessary risk. We hope that our discussion of tap-jacking will encourage more sites to embed frame busting in their web pages.

Beyond mobile sites, we studied the effectiveness of overlay-based frame busting as used by Facebook. We showed that while this defense may prevent traditional click-jacking, it can result in exposure of private user information. When possible it is much safer to use traditional frame busting [11] that prevents user content from rendering in a sub-frame of an unknown domain.

We also showed that web vulnerabilities, including framing and XSS, can result in theft of a wifi's WPA key in routers that use a default password. As an added twist we noted that Mozilla's location-aware browsing protocol can help the attacker determine the exact location of the victim's wifi network. While conceptually simple, getting these attacks to work in practice took considerable effort. The complexity of the attack may suggest that browsers are getting better at protecting users from basic web exploits, however several holes such as drag-and-drop attacks still remain.

References

[1] H. Bojinov, E. Bursztein, and D. 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.

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

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

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

[5] R. Hansen. Clickjacking. ha.ckers.org/blog/20080915/clickjacking.

[6] S. Kamkar. mapxss: Accurate geolocation via router exploitation. http://samy.pl/mapxss/, January 2010.

[7] 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.

[8] Mozilla. Location-aware browsing. www.mozilla.com/en-US/firefox/geolocation.

[9] Y. Niu, F. Hsu, and H. Chen. iphish: Phishing vulnerabilities on consumer electronics. In *Proc. of UPSEC*, 2008.

[10] J. Resig. Clickjacking iphone attack, 2008. ejohn.org/blog/clickjacking-iphone-attack.

[11] G. Rydstedt, E. Bursztein, D. Boneh, and C. 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.

[12] S. Stamm, Z. Ramzan, and M. Jakobsson. Drive-by pharming. In *Proc. of ICICS*, pages 495–506, 2007.

[13] P. Stone. Next generation clickjacking. media.blackhat.com/bh-eu-10/presentations/Stone/BlackHat-EU-2010-Stone-Next-\Generation-Clickjacking-slides.pdf, 2010.