HTML5 – Building a better botnet

Web Attacks 3.0

What a tangled web we weave. New web technologies address shortcomings in web browsers but create new problems as well. *By Kurt Seifried*

Fresh browsers are here! Opera 11.00 has been released and Firefox 4 is in solid beta, and they both include improved support for HTML5, JavaScript (faster is better), and some newer technologies like WebSockets, Web Workers, and Web Storage. These features are designed to address the simple fact that people now expect the web to provide applications and multimedia (ideally without needing third-party closed source plugins like Flash).

These three web technologies address some shortcomings in web browsers, but like most new technologies, they also create a completely new set of security problems. I'll describe how they can be used to create a massive denial-of-service network for next to nothing and how they can be used to track people.

Web Storage and HTML5

Until recently, the only reliable method for storing (and retrieving) data from a web client was cookies. Cookies are a few kilobytes in size typically, so if you want to store a large amount of data on a client (such as an image or a document), you'd have to split it up across many cookies and hope that none get deleted or replaced. HTML5 introduces Web Storage, which comprises Session Storage, Local Storage, Global Storage, and Database Storage (using SQLite).

Like cookies, these storage objects are generally bound to the domain they came from, which should largely prevent sites from stealing data or using Web Storage to pass data around easily. However, unlike cookies, Web Storage provides new ways to track users in a very persistent manner (e.g., you set a key called "tracking" with a unique string) [1] [2]. This feature can't directly be used to aid in botnet creation, but it does give attackers a way to store data and then retrieve it later.

WebSocket

WebSocket is such a problem that currently (as of December 2010) Opera 11.00 and Firefox 4 beta 8 both ship with it disabled. The reason is WebSocket is designed to allow a web browser to make a request to a third-party site that is generated from within a JavaScript program, for example. For the last few years, Firefox and other browser vendors have been working to prevent such "cross-origin requests" and "cross-site request forgeries" (i.e., *attacker.com* causing your browser to get something from *yourbank.com* and then interacting with it).

Making WebSocket "safe" for use has been attempted by requiring the server to reply to such requests in a way that shows it's okay, but this fails to take into account two problems. The first problem is HTTP header splitting; some web applications and servers can be manipulated (e.g., through the request string) to create a custom HTTP header, which is sent back to the client (which can contain cookies, weird caching values, etc.) and could include the WebSocket reply data needed by the client.

The second problem is simply that a client can make a lot of WebSocket requests (i.e., several thousand per second), turning this into a great denial-of-service platform [3]. With this approach, you simply get people to visit a web page that you control and, as long as that page is open, the browser will hammer away at whatever site you want taken down. Alternatively, this feature can be

KURT SEIFRIED

Kurt Seifried is an Information Security Consultant specializing in Linux and networks since 1996. He often wonders how it is that technology works on a large scale but often fails on a small scale.

used for port scanning. Because the WebSocket interface takes a variable amount of time to return, depending on whether the connection fails, connects, or is refused, you can determine the port status. An example of such a scanning tool is JS-Recon [4], which you can use to port scan your local machine or your local network or to discover the private IP address of the system.

Note that determining the network address of a user is made easier by the fact that most of us are behind NAT boxes using 192.168.*.* (the default for virtually all home routers) or 10.*.*.* (almost no one uses 172.16.0.0 through 172.31.255.255). Simply scanning 192.168.*.1 and 192.168.*.254 will give you a very good chance of finding the router used to provide network access.

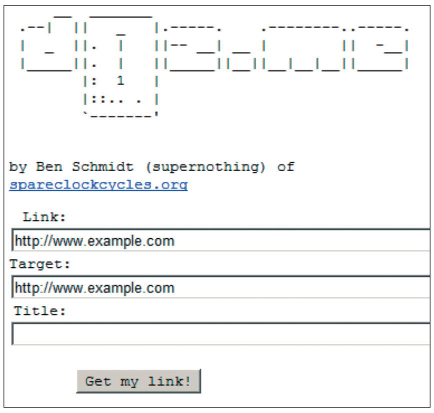
To top it off, one security vulnerability in WebSocket allows attackers to attack web-based proxies or transparent intercepting proxies because many of them don't fully understand how to handle WebSocket connections [5] yet. By poisoning the cache of a web proxy for an item such as *http://www.google-analytics.com/ga.js*, an attacker can send malicious code to any victim accessing a website that uses Google Analytics.

Web Workers

Previously, most JavaScript programs online were not very big and did not run for long periods. Now I have a web browser open with Gmail in a tab for days at a time. The latest figure I can find lists Gmail as 443,000 lines of custom JavaScript.

With the use of Web Workers, you can now create and destroy threads and split up the work among them. Thus, programs can be split into modules and run for a very long time and can also take advantage of multicore processors (which is basically everything except mobile phones, but dual-core mobile phones are coming). Ultimately, this means that web-based JavaScript applications will be able to behave more like traditional applications.

Although JavaScript is slow compared with well-written C or C++ code, it can still generate a huge number of requests on even a low-end machine, turning the system into an effective denial-of-service platform. All the attacker needs to do is keep you on their web page, which is not hard to do with forum discussion sites or online games.


```
     __  ________
 .--|  ||        |.-----.   .--------.-----.
 |  _  ||.  _    ||-- __| __ |        ||  -__|
 |_____||.  |    ||_____||__||__|__|__||_____|
        |:  1    |
        |::.. .  |
        `--------'
```

by Ben Schmidt (supernothing) of spareclockcycles.org

Link: http://www.example.com

Target: http://www.example.com

Title:

Get my link!

Figure 1: The d0z.me website.

Easy XSS – history.pushState()

A lot of these attacks work much better when combined with XSS attacks (injecting hostile content into trusted websites), so is there a new HTML5 component that makes this easier? Of course. The new `history.pushState()` function allows the URL in the history to be modified. Unfortunately, it will also work on the current URL, thereby allowing an attacker to rewrite the address bar – which is again something most browsers have been attempting to prevent for years [6].

Work as Payment

The ability to execute long-running processes and make requests to arbitrary websites leads to an interesting possibility: compute time as a micro-payment system. However, all of the use cases I can think of essentially involve spam or some other unwanted activity, such as denial-of-service attacks against sites.

One example of this is d0z.me [7], a URL-shortening service. You simply type in the URL you want shortened (as usual) and the URL you want to attack. When a user clicks on the d0z.me URL, the program redirects the user to the long URL. However, it will embed the site within an `IFRAME`, while another `IFRAME` constantly reloads the site that's being attacked. This approach, combined with social network sites like Reddit, could easily result in a few hundred or thousand people attacking a site.

One Last Kick at HTML5

Because most sites are getting better at dealing with distributed denial-of-service attacks, attackers will need to invest more time in creating more involved attacks.

Instead of simply sending a request for the front page, for example, they can fill out a contact form and hit submit, thereby flooding the support account or the sales account with junk. HTML5 offers a drag-and-drop interface. This, combined with interactive content, such as a game, could allow the attacker to get clients to fill out the form (in the background, of course) and submit it repeatedly.

Conclusion

As usual, the guys developing new web technologies didn't give much thought to security problems, which is bad if you're a regular user, but great if you're a bad guy [8] [9] or you can do cool things with HTML5 (like building a distributed password cracker).

In parting, I leave you with this interview with Douglas Crockford (the guy who created JSLint and helped develop JSON) [10]. ■■■

INFO

[1] Evercookie: *http://samy.pl/evercookie/*

[2] Nevercookie: *http://www.anonymizer.com/learningcenter/#lc_labs*

[3] Performing DDoS Attacks with HTML5: *http://blog.andlabs.org/2010/12/performing-ddos-attacks-with-html5.html*

[4] JS-Recon port scanner: *http://www.andlabs.org/tools/jsrecon.html*

[5] WebSocket: *https://developer.mozilla.org/en/WebSockets*

[6] history.pushState(): *http://samuli.hakoniemi.net/how-to-conceal-xss-injection-in-html5/*

[7] d0z.me: *http://it.slashdot.org/story/10/12/20/2248219/D0zme-mdash-the-Evil-URL-Shortener*

[8] Attacking with HTML5: *http://www.slideshare.net/clubhack/attacking-with-html5lava-kumar*

[9] HTML5 Security Cheatsheet Project: *http://code.google.com/p/html5security/*

[10] Douglas Crockford on JavaScript and HTML5: *http://www.webmonkey.com/2010/05/douglas-crockford-on-javascript-and-html5/*