Cross domain search timing
By: Chris Evans
URL: http://scarybeastsecurity.blogspot.com/2009/12/cross-domain-search-timing.html

I've been meaning to fiddle around with timing attacks for a while. I've had various discussions in the past about the significance of login determination attacks (including ones I found myself) and my usual response would be "it's all moot -- the attacker could just use a timing attack". Finally, here's some ammo to support that position. And -- actual cross-domain data theft using just a timing attack, as a bonus.

Unfortunately, this is another case of the web being built upon broken specif-ications and protocols. There's nothing to stop domain evil.com referencing resources on some.sensitive.domain.com and timing how long the server takes to respond. For a GET request, a good bet is the < img > tag plus the onerror() / onload() events. For a POST request, you can direct the post to an < iframe > element and monitor the onload() event.

Why should an evil domain be able to read timing information from any other domain? Messy. Actually, it's even worse than that. Even if the core web model didn't fire the relevant event handles for cross-domain loads, there would still be trouble. The attacker is at liberty to monitor the performance of a bunch of busy-loops in Javascript. The attacker then frames or opens a new window for the HTML page they are interested in. When performance drops, the server likely responded. When performance goes up again, the client likely finished rendering. That's two events and actually a leak of more information that the pure-event case.

Moving on to something real. The most usable primitive that this gives the attacker is a 1-bit leak of information. i.e. was the request relatively fast or relatively slow? I have a little demo:

https://cevans-app.appspot.com/static/ymailtimings.html

It takes a few seconds, but if I'm not logged into Yahoo! Mail, I see:

```
DONE! 7 79 76 82
```

From the relatively flat timings of the last three timings (three different inbox searches) and the relative latency between the first number and the latter three, it's pretty clear I'm not logged in to Yahoo! Mail.

If I'm logged in, I see:

```
DONE! 10 366 414 539
```

This is where things get interesting. I am clearly logged in because of the significant server latency inherent in a text search within the inbox. But better still, the last three numbers represent searches for the words nosuchterm1234, sensitive and the. Even with a near-empty inbox, the server has at least a 40ms difference in minimum latency between a query for a word not in the index, and a query for a word in the index. (I mailed myself with sensitive in the subject to make a clear point).

There are many places to go from here. We have a primitive which can be used to ask cross-domain YES/NO questions about a victim's inbox. Depending on the power of the search we are abusing, we can ask all sorts of questions. e.g. "Has the victim ever mailed X?", "If so, within the past day?", "Does the word earnings appear in the last week?", "What about the phrase 'earnings sharply down'?" etc. etc. By asking the right YES/NO questions in the right order, you could reconstruct sentences.

It's important to note this is not a failing in any particular site. A particular site can be following current best practices and still be bitten by this. Fundamentally, many search operations on web sites are non-state-changing GETs or POSTSs and therefore do not need XSRF protection. The solution, of course, is to add it (and do the check before doing any work on the server like walking indexes)!

With thanks to Michal Zalewski for interesting debate and Christoph Kern for pointing out this ACM paper[1], which I haven't read but from the abstract it sounds like it covers some less serious angles of the same base attack.

1. http://portal.acm.org/citation.cfm?id=1242656