Haketilo - Support #78

Investigate into how browsers handle files that are not HTML

08/27/2021 10:32 AM - koszko

Status:	Rejected	**Start date:**	08/27/2021
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	60%
Category:		**Estimated time:**	0.00 hour
Target version:			

Description

Our tampering with HTML pages, including rewriting parts of them using the StreamFilter API, might cause problems when those pages are not actually HTML.
Additionally, subtleties of other file types might require us to handle some special cases in order to make script blocking thorough.

History

#1 - 08/28/2021 03:00 AM - jahoti

For the second point at least, I know NoScript operates on XML (and will check uBlock Origin for similar behavior). What exactly the threat being fixed is remains to be discovered.

As for making sure we only filter relevant data, do any browsers try to guess mime types? If not, that should be (reasonably) easy to ensure.

#2 - 08/28/2021 08:56 AM - koszko

> As for making sure we only filter relevant data, do any browsers try to guess mime types?

By guessing you mean analyzing the content in order to find out? I suppose so, not 100% sure. However, just analyzing the headers might be a tough job. I mean, if we don't use the same code browser does, we might get different results for some pathological corner cases. Whether this will cause serious problems in the future is not known.

#3 - 09/03/2021 10:21 AM - jahoti

According to https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types#mime_sniffing:

> In the absence of a MIME type, or in certain cases where browsers believe they are incorrect, browsers may perform *MIME sniffing* — guessing the correct MIME type by looking at the bytes of the resource.
>
> Each browser performs MIME sniffing differently and under different circumstances. (For example, Safari will look at the file extension in the URL if the sent MIME type is unsuitable.) There are security concerns as some MIME types represent executable content. Servers can prevent MIME sniffing by sending the X-Content-Type-Options header.

#4 - 09/03/2021 11:17 AM - koszko

Heuristics. That's bad... For us.

Even mere parsing of response headers is already risky because of some subtleties that might cause Hachette to interpret the headers differently from the browser. Now this...

Perhpas we could instead, in StreamFilter, just try running DOMParser over the first chunk of data and examining the resulting tree? If data is HTML without CSP tags under <head> or not HTML at all, the tree will not have any bad CSP <meta> tags and our code could then decide not to modify it at all!

And, given that we also don't need to modify responses of types other than "main_frame" or "sub_frame", nor responses for pages on which we don't inject anything, chances of breaking something would be low

#5 - 09/03/2021 12:19 PM - jahoti

> Perhpas we could instead, in StreamFilter, just try running DOMParser over the first chunk of data and examining the resulting tree? If data is HTML without CSP tags under or not HTML at all, the tree will not have any bad CSP tags and our code could then decide not to modify it at all!

That should do! Chromium doesn't support StreamFilter, of course; do we even need to address that?

#6 - 09/03/2021 12:36 PM - koszko

No, since under Chromium I've never actually seen our "document_start" content scripts start with DOM partially or fully loaded. This seems to happen under Mozilla only (and only in some cases)

In #83 you mentioned you are going to do work around CSP rules now. If so, how about I focus on this issue?

#7 - 09/03/2021 07:19 PM - koszko

Modified StreamFilter code is now on koszko-rethinked-meta-sanitizing. The policy object now also contains information whether there is some payload to be injected.

Btw, I noticed cookies don't work on non-HTML pages. This doesn't seem to be an issue as long as we assume the concepts of scripting and whitelisting only apply to HTML pages.

Content script in those cases still tries to inject the payload and (under Mozilla) fails because it uses a different nonce than the one that would be smuggled in the cookie.

Now it would make sense to make content script not try to inject payload if document.contentType is not of proper format (use a condition like "/html/.test(document.contentType)" maybe?). Later on we could allow payloads to specify which content types they should be applied to...

HEY! I just realized a very cooooool thing. We are perfectly able to:

1. Use webRequest to modify response headers to spoof and force a content type like "text/plain"
2. Access the raw HTML code from content script
3. Use document.write() to display what we actually want to be displayed

This is going to be useful in some scenarios. E.g. if some custom page resource includes its own HTML file (once we add support for this...) we would be able to avoid having browser even parse the original one and instead provide its raw HTML code to Hachette-injected scripts

#8 - 09/05/2021 01:40 AM - jahoti

> Btw, I noticed cookies don't work on non-HTML pages. This doesn't seem to be an issue as long as we assume the concepts of scripting and whitelisting only apply to HTML pages.

I've been (half-heartedly) looking into this for a while now and found no clear evidence of anything. If anything comes up, I'll tear my hair out and note it here (not in that order :).

> Now it would make sense to make content script not try to inject payload if document.contentType is not of proper format (use a condition like "/html/.test(document.contentType)" maybe?). Later on we could allow payloads to specify which content types they should be applied to...

While I agree, doesn't specifying content types for payloads assume payloads can be applied to anything other than HTML?

> HEY! I just realized a very coooool thing. We are perfectly able to:
>
> - Use webRequest to modify response headers to spoof and force a content type like "text/plain"

That would be a good strategy actually! Is there a good place to make a note of it for when it becomes applicable?

#9 - 09/06/2021 09:56 AM - koszko

> > Now it would make sense to make content script not try to inject payload if document.contentType is not of proper format (use a condition like "/html/.test(document.contentType)" maybe?). Later on we could allow payloads to specify which content types they should be applied to...
>
> While I agree, doesn't specifying content types for payloads assume payloads can be applied to anything other than HTML?

Well, it does. While server might not be able to make user's browser execute scripts in a non-HTML page, we are. Should we restrain from doing this?

#10 - 09/06/2021 11:48 AM - jahoti

> While server might not be able to make user's browser execute scripts in a non-HTML page, we are. Should we restrain from doing this?

Oh, I definitely agree we should- I just wasn't sure if we could. How would it work?

#11 - 09/06/2021 02:57 PM - koszko

> > While server might not be able to make user's browser execute scripts in a non-HTML page, we are. Should we restrain from doing this?
>
> Oh, I definitely agree we should- I just wasn't sure if we could. How would it work?

I'm confused. Do you agree that we should "*allow payloads to specify which content types they should be applied to*" or that we should "*restrain from doing this*"?

#12 - 09/06/2021 06:55 PM - koszko

Now we know why NoScript included special code for SVGs and XMLs:
https://developer.mozilla.org/en-US/docs/Web/SVG/Element/script

Injection of CSP using <meta> tag into document object that is not of type HTMLDocument is impossible. I am working on this now

#13 - 09/06/2021 08:49 PM - koszko

I came up with code that should do with blocking for now. On koszko branch. Could do with more testing

#14 - 09/07/2021 12:02 AM - jahoti

> I came up with code that should do with blocking for now. On koszko branch. Could do with more testing

Doing this ASAP; do you know of a specific issue with XML or is it just the SVG issue also being applicable?

#15 - 09/07/2021 10:52 AM - koszko

I suppose it's the same as with SVG, although I need to make sure it's really the case

#16 - 09/07/2021 10:31 PM - koszko

I now realize what is the problem with all XMLs, including SVGs. Any XML can include elements from other XML namespaces

```
<?xml version="1.0" encoding="UTF-8"?>

<fruits>

  <!-- The following will not execute since it is not recognized as either HTML or SVG script -->
  <script>alert('banana');</script>

  <!-- Will execute -->
  <html:script xmlns:html="http://www.w3.org/1999/xhtml">console.log('grape');</html:script>

  <!-- Will also execute -->
  <vector-graphics:script xmlns:vector-graphics="http://www.w3.org/2000/svg">console.error('raspberry');
</vector-graphics:script>

  <apple>
    <svg viewBox="0 0 10 14" xmlns="http://www.w3.org/2000/svg">
      <!-- Will run when clicked -->
      <circle cx="5" cy="5" r="4" onclick="console.warn('antonowka')" />
      <!-- Will *NOT* run when clicked -->
      <circle cx="5" cy="13" r="4" some-unknown:onclick="console.warn('nowamak')" xmlns:some-unknown=
"https://example.org/blah/blah" />
    </svg>

<!-- In case of wrong namespace URI (or lack thereof), svg subtree will not be recognized as SVG at all -->
    <svg viewBox="0 0 10" xmlns="http://www.w3.org/2000/sv">
      <!-- Will neither run nor be drawn by the browser -->
      <circle cx="5" cy="5" r="4" onclick="console.warn('golden')" />
    </svg>
  </apple>
</fruits>
```

So far I've always seen the document object as an instance of either HTMLDocument or XMLDocument. Contrary to what NoScript's code suggests, it IS possible to apply CSP rule to an XMLDocument, too. I managed to do it with:

```
html = document.createElementNS("http://www.w3.org/1999/xhtml", "html");
head = document.createElementNS("http://www.w3.org/1999/xhtml", "head");
meta = document.createElementNS("http://www.w3.org/1999/xhtml", "meta");
meta.content = "script-src 'none';";
meta.httpEquiv = "Content-Security-Policy";
head.append(meta);
html.append(head);
backed_up_root = document.documentElement;
document.documentElement.replaceWith(html);
document.documentElement.replaceWith(backed_up_root);
```

I hope this snippet would work on all relevant browsers. On some it would be perhaps sufficient to add just the <meta> or maybe to add the <html> beneath the root element. It seems IceCat is more picky that Chromium as to where and how the <meta> may appear if it is to take effect.
When I realized this I got worried this might allow pages to add bad <meta> tags after <head> in HTML documents as well but no, this turns out to be handled more strictly there than in XMLDocument.

One problem is that at least some Mozilla browsers assume XML document is non-modifiable and altering it in any way disrupts the preview... I guess

we have no option but to swallow that inconvenience for the sake of blocking :/

I am going to continue with this tomorrow. Btw, I realized some mistakes (including being unaware of what I just described) in the code I committed previously. Now, I am going to utilize this CSP discovery in blocking intrinsics

#17 - 09/09/2021 12:15 PM - koszko

> I am going to continue with this tomorrow. Btw, I realized some mistakes (including being unaware of what I just described) in the code I committed previously. Now, I am going to utilize this CSP discovery in blocking intrinsics

I pushed something to koszko branch. There are new things worth noting:

1. Even when CSP is injected, under Mozilla it fails to affect the scripts and intrinsics that were already there. Hence, I had to resort to blocking of intrinsics with wrappedJSObject and using "beforescriptexecute" to block <script>s
2. On Mozilla I noticed that, at least for XML documents from file://, when at document_start I added a <meta> with CSP rule that allowed 'unsafe-eval', other <meta> present in the page was unable to further tighten this CSP and using window.eval from withing content script (which under Mozilla executes in page's context according to one of their Wiki pages I cannot find right now) was still possible. We could investigate that at some later point - perhaps it is something that would allow us to drop StreamFilter usage?

#18 - 09/11/2021 04:38 AM - jahoti

> I pushed something to koszko branch.

Rather than reply to all the commits you've made independently, I'll just note here I'll look through them all today. It sounds like we must nearly be ready for 0.1!

> There are new things worth noting:
>
> - Even when CSP is injected, under Mozilla it fails to affect the scripts and intrinsics that were already there. Hence, I had to resort to blocking of intrinsics with wrappedJSObject and using "beforescriptexecute" to block s
> - On Mozilla I noticed that, at least for XML documents from file://, when at document_start I added a with CSP rule that allowed 'unsafe-eval', other present in the page was unable to further tighten this CSP and using window.eval from withing content script (which under Mozilla executes in page's context according to one of their Wiki pages I cannot find right now) was still possible. We could investigate that at some later point - perhaps it is something that would allow us to drop StreamFilter usage?

Perhaps, albeit the effect would probably be more on how we inject scripts than StreamFilter (didn't the CSP-filtering part of StreamFilter get removed anyway?).

#19 - 09/11/2021 04:52 AM - koszko

> didn't the CSP-filtering part of StreamFilter get removed anyway?

It did, although the part that remains is still there because of CSP (to inject a <script> and therefore force content script to run before the first <meta> appears)

#20 - 09/11/2021 05:08 AM - jahoti

Good point!

#21 - 09/11/2021 12:17 PM - jahoti

Your most recent push seems to be working well!

#22 - 09/14/2021 11:48 PM - jahoti

- % Done changed from 0 to 60

Rough estimate of progress (it's hard to tell without knowing in advance what the solution will involve)

#23 - 10/13/2022 12:09 PM - koszko

- Status changed from New to Rejected

This is not as relevant to new Haketilo proxy because we only inject scripts to HTTP responses with html mime type. And script blocking is probably thorough because we're only concerned with HTTP(s) where CSP does all the job