Haketilo - Feature #15

make sure page's own csp in <head> doesn't block our scripts

07/01/2021 12:30 PM - koszko

Status:	Closed	**Start date:**	07/01/2021
Priority:	High	**Due date:**	
Assignee:	koszko	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:			

Description

Currently we inject scripts by creating a tag and adding it at the end of . We remove page's own csp HTTP header, so they are no longer a problem. Still, CSP tags in seem like an issue. This needs investigating first.

Related issues:

Related to Haketilo - Bug #53: Interference with existing CSP headers	**Closed**	**07/17/2021**

History

#1 - 07/31/2021 03:21 AM - jahoti

- Assignee set to jahoti

#2 - 08/13/2021 05:13 PM - koszko

I see you tried to remove the offending <meta> csp tags in the csp-PoC branch. Unfortunately, to the extent I tested, removing the tag doesn't revert the csp rules it imposed. This is what I saw on both UC 90 and IceCat 60.

I started looking for a solution and found out a very good thing. In Chromium at document_start we could stop unwanted <meta> tags from being parsed at all by doing something like document.write('<script type="text/plain>"'). After that, everything until the next closing </script> will be written inside our own non-executable <script>. We can then utilize MutationObserver to intervene in the moment our <script> gets closed and open a new one.
While doing this we can collect all data browser writes to the document. In the end (when everything gets written), we have a document with none of page's own tags in it and all the HTML in a variable. We can then sanitize that HTML and finally document.write() it and finish with document.close(). This will cause the entire DOM tree to be re-created (look up the semantics of document.write(); I learned about it when making the benchmark fix for Phoronix).

So, in the end, this will not only allow us to modify the offending csp rules but also impose script-blocking and do other changes **asynchronously**.

For Mozilla, the same approach may not work because content scripts from document_start start executing after a part of the DOM (quite often up until the first <script>) is already there. We can, however, utilize the same method LibreJS does (leveraging that one Mozilla-only API) to rewrite part of the HTML from webRequest and not have to worry about blocking in content scripts at all.

Btw, idk if you noticed, but under some Mozilla browsers (at least the 2 I have), events for MutationObservers registered from within content scripts (usually) do not fire. MutationObserver seems to work, however, when used from page's conext. It works a bit different than in Chrome, though, with possibly more than one mutation happening before mutation event is delivered.

I also noticed that rules from CSP <meta> tags don't apply if we somehow force them to appear after <head>. So for example document.write('<head></head><body>') should - I believe - do the trick. Of course, this is viable only if we're later to rewrite the entire document.

Yet another thing that might be useful when used together with the document.write() trick are custom HTML tags[1].

[1] https://stackoverflow.com/questions/10830682/is-it-ok-to-use-unknown-html-tags

#3 - 08/13/2021 05:17 PM - koszko

- Related to Bug #53: Interference with existing CSP headers added

#4 - 08/14/2021 03:10 AM - jahoti

> I started looking for a solution and found out a very good thing. In Chromium at document_start we could stop unwanted <meta> tags from being parsed at all by doing something like document.write('<script type="text/plain">'). After that, everything until the next closing </script> will be written inside our own non-executable <script>. We can then utilize MutationObserver to intervene in the moment our <script> gets closed and open a new one.
> While doing this we can collect all data browser writes to the document. In the end (when everything gets written), we have a document with none of page's own tags in it and all the HTML in a variable. We can then sanitize that HTML and finally document.write() it and finish with document.close(). This will cause the entire DOM tree to be re-created (look up the semantics of document.write(); I learned about it when making the benchmark fix for Phoronix).
>
> So, in the end, this will not only allow us to modify the offending csp rules but also impose script-blocking and do other changes **asynchronously**.

That's actually pure genius- every cloud has a silver lining, I suppose! I'll start this within the next few days if you haven't already.

> For Mozilla, the same approach may not work because content scripts from document_start start executing after a part of the DOM (quite often up until the first <script>) is already there. We can, however, utilize the same method LibreJS does (leveraging that one Mozilla-only API) to rewrite part of the HTML from webRequest and not have to worry about blocking in content scripts at all.

That's one option. Injecting scripts using the userscript API and ignoring the page CSP might also work, and NoScript seems to do whatever it does in pages using document.write on both Firefox and Chrom*.

> Btw, idk if you noticed, but under some Mozilla browsers (at least the 2 I have), events for MutationObservers registered from within content scripts (usually) do not fire. MutationObserver seems to work, however, when used from page's conext. It works a bit different than in Chrome, though, with possibly more than one mutation happening before mutation event is delivered.

I don't think I've ever encountered that issue; I'll have a look.

> So, in the end, this will not only allow us to modify the offending csp rules but also impose script-blocking and do other changes **asynchronously**.

That's actually pure genius- every cloud has a silver lining, I suppose! I'll start this within the next few days if you haven't already.

I am investigating this. Turns out Chromium is also capable of not delivering Mutation events immediately. A spurious </script> at the beginning of the document could cause serious issues with my method. There are, however, multiple ways to try to work around this. Hopefully, one of them will work :)

> That's one option. Injecting scripts using the userscript API and ignoring the page CSP might also work,

I recall you proposed that before. While there is huge risk of incompatibilities, this might be our last resort for injecting into non-http(s) pages under Mozilla.

> Btw, idk if you noticed, but under some Mozilla browsers (at least the 2 I have), events for MutationObservers registered from within content scripts (usually) do not fire. MutationObserver seems to work, however, when used from page's conext. It works a bit different than in Chrome, though, with possibly more than one mutation happening before mutation event is delivered.

I don't think I've ever encountered that issue; I'll have a look.

Update: I was wrong with regard to Chrome. It's the same there - the thing that forces mutation event to be triggered is a mutation that adds some executable <script>.

#6 - 08/14/2021 01:03 PM - koszko

> A spurious </script> at the beginning of the document could cause serious issues with my method. There are, however, multiple ways to try to work around this. Hopefully, one of them will work :)

Did it!
Consider this*:

```
<script>
  /**
   * This file and `test4.js' are Copyright 2021 Wojtek Kosior
   * Released under Creative Commons Zero 1.0
   */

  /* Assume this <script> has been somehow injected at the beginning. */

  document.scripts[0].remove();

  let pieces = [];

  function finish_hax()
  {
      pieces.push(document.scripts[0].innerHTML);
      document.scripts[0].remove();
      console.log(pieces.join(''));
  }

  document.addEventListener("DOMContentLoaded", finish_hax);

  document.write('<script src="test4.js">');
</script></script>
<!doctype html>
<!--
    Everything after the intial <script> is just garbage used to test
    if all the HTML gets hijacked correctly.
 -->
<html>
  <head>
    <meta charset="utf-8"/>
    <script>
      console.log("kiiiiiiiiiii");
    </script>
    <script>
      console.log("tH3");
    </script>
    <script>
      console.log("World Wild Web");
    </script>
    <meta http-equiv="content-security-policy" content="script-src 'nonce-2345';">
  </head>
  <body>
    <iframe><html><head></head><body><div>
        Kitten fight
    </div></body></html></iframe>
    <script nonce="2345">
      console.log("take that!");
    </script>
    <span>Hello!</span>
  </body>
</html>
```

And this is test4.js:

```
pieces.push(document.scripts[0].innerHTML);
pieces.push('</script>');
```

```
document.scripts[0].remove();

document.write('<script src="test4.js">');
```

Result? Everything after the initial, "somehow injected" <script> gets logged to console. At the end, DOM contains just

```
<html>
  <head>
  </head>
  <body>
  </body>
</html>
```

No <meta> tags get loaded to DOM, no <script>s from the rest of the document get loaded or executed. This hax behaves the same under both Ungoogled Chromium 90 and IceCat 60* ^^

Better yet, we're able to put some kind of preloader there for when page loads xd

* I just realized document.currentScript can be used instead of document.scripts[0] and is more informative
** Mozilla browsers show warnings about data from the network getting reparsed. Honestly, I don't consider it a big issue

EDIT: Under Mozilla, we can use the StreamFilter API to either sanitize the entire document, or inject a script like this one at the beginning of it. I am not entirely sure which option is better...

#7 - 08/15/2021 12:47 PM - jahoti

That is genuine genius- I think it will take me a little while to understand exactly what magic you've pulled :).

#8 - 08/16/2021 11:24 AM - koszko

> I think it will take me a little while to understand exactly what magic you've pulled :).

All that's needed is to realize a few things:

- <script>s execute synchronously with page loading. Even a remote script with scr attribute has to be fetched and executed before subsequent DOM elements can load.
- If a <script> has both some text content (between its tags) and a src attribute, text content gets ignored
- A <script> only starts executing after its closing tag (</script>) gets written to the document. This is fortunately also true for scripts with src attr.

I started implementing this document hijacking for Chromium. Once that's done, the only thing left is to borrow some StreamFilter code from LibreJS to cover Mozilla and then we're good to get rid of policy injection in URL and mark https://hachettebugs.koszko.org/issues/65 as closed :)

#9 - 08/17/2021 07:50 PM - koszko

UPDATE
Bad news (but read on!) - we cannot use document.write() this way from content script nor from any <script> that was injected by content script at document_start :/ This is because execution of a content script at document_start is treated as asynchronous to the flow of document loading...

After a bit of thinking I found another promising way to deal with nasty <meta> CSP tags in pages. Under Chromium, calling document.write() (alternatively, document.open()) from a content script at document_start causes document.documentElement to be replaced with a new HTMLHtmlElement object. The old one, however, still exists, the browser continues to create elements under it and a <meta> with CSP rule, if one appears there, still takes effect. However, we can call .remove() on the old documentElement and subsequent <meta>s under it won't be effective anymore.

Right now I am not yet certain if we're able to re-create a satisfactory good HTML source from the old documentElement to be able to use it with document.write(). However, in the meantime I realized another approach (not yet validated) to deal with bug 65 without the need for asynchronous policy fetching: https://hachettebugs.koszko.org/issues/65#note-1

P.S. Seems the new approach will actually be a bit easier to implement than the failed one. Sad that I already wrote the toughest parts of that one :/

#10 - 08/18/2021 01:19 AM - jahoti

> Sad that I already wrote the toughest parts of that one :/

Sigh :/

At least you've got something to start with if ever we need to try salvaging this approach. It seems like we could end up implementing the same function 100 different ways to try and deal with the mess of incompatibilities and corner cases that are coming up.

Maybe the *extension* should have been named Hydrilla- whenever one path gets cut off, two more grow in its place :).

#11 - 08/20/2021 01:04 PM - koszko

- % Done changed from 0 to 50

- Assignee changed from jahoti to koszko

- Status changed from New to In Progress

> Maybe the *extension* should have been named Hydrilla- whenever one path gets cut off, two more grow in its place :).

True □□

Anyway, I committed the Chromium code for this issue to my branch. I hope to be able to use the modify_on_the_fly() facility under FF as well (I wrote it with this in mind)

#12 - 08/21/2021 08:55 PM - koszko

- % Done changed from 50 to 80

Had some issues again (document created with DOMParser can be written to under Chromium but not under IceCat 60). After those, however, I think I am close to getting the StreamFilter thing working the way I want it to. I just need to fix one more bug in my document on-the-fly-modifier :)

Expect another update on Monday

#13 - 08/23/2021 11:14 AM - koszko

- % Done changed from 80 to 90

The code that uses StreamFilter is now on my branch. The remaining issues are worth mentioning:

1. Under Chromium large pages get sanitized as they load. Under Mozilla, paradoxally, we have to wait for the entire page to load before anything gets displayed. Tested on [1]. This is due to a difference in how those 2 browsers behave when the still-loading document root gets replaced with another one. I'll investigate possible workarounds for Mozilla.
2. It seems that under Mozilla, when extension gets reloaded, content scripts get re-injected into pages and cause document sanitizing to start again and never finish. We can do sth about it

[1] https://www.gnu.org/savannah-checkouts/gnu/autoconf/manual/autoconf-2.70/autoconf.html

#14 - 08/23/2021 06:18 PM - koszko

> I'll investigate possible workarounds for Mozilla.

I did.

- We can make a HTML on-the-fly "parser" by creating an <iframe> with script execution blocked, write()'ing our HTML there and observing DOM nodes that appear there.
- This <iframe> has to be actually added to some live document for this to work.
- In the light of the above we could possibly use background page for this, but it is getting removed in V3, so we'd better temporarily put this frame on the actual page that loads and make it invisible.
- It turns out <meta> tags inside a same-origin <iframe> are able to somehow add CSP rules to the parent frame.
- I tested the above on a frame filled using srcdoc attribute and strangely, even adding a sandbox attribute did not affect this behavior.
- If we make the frame contain some extension-bundled html page, the problem with <meta> tags goes away.
- Attempt to write() to the document of an extension-bundled page causes an error and kills that page.
- It is possible to put another <iframe> inside the frame with extension-bundled page. The inner frame's document is then write()'able and seems to work the way we need.
- Only one minor problem remains in such case: error messages regarding the HTML we parse using this workaround will clobber our extension's console.

I hate web browsers. It all grows waaaay more complex than I expected. Even though I didn't like the idea of using the userScripts API when you proposed it, I am now willing investigate into using it just to save the time it would take to implement the above frame-in-a-frame idea

EDIT: browser.userScripts is undefined under IceWeasel 75. I shall commit to implementing the <iframe> crazyness now

#15 - 08/23/2021 11:56 PM - jahoti

> I hate web browsers. It all grows waaaay more complex than I expected.

Which then wastes half one's energy remembering why "burn it all and start from scratch" isn't a viable alternative :/.

I might try implementing an iframe-based approach too, albeit slightly different from yours (and with far less chance of success). Hopefully after all this we can finally put CSP-based issues to rest!

#16 - 08/25/2021 09:55 PM - koszko

I instead implemented a hacky way that uses multiple invocations of DOMParser to find where page's <head> ends so that we can sanitize just this initial part inside StreamFilter without waitig for the rest of page'a data.

I admit this is hacky and there are holes in it. Well, I see no better way to solve this for now.

I am utilizing the fact <meta> tags are only effective inside <head> and not when they appear inside <body>.

I now suspect it might be possible to improve sanitize_document.js to only sanitize <head> and then restore the original <body> to the document and detach from it, thus solving the performance issue and removing the need for so extensive use of StreamFilter. I feel too tired to do this now, though.

Otherwise, there are ways to optimize the current DOMParser hack:

- don't process <head> once we find it contains no CSP <meta> tags
- don't use StreamFilter on pages into which we're not injecting any payload

#17 - 08/26/2021 12:07 AM - jahoti

> I instead implemented a hacky way that uses multiple invocations of DOMParser to find where page's ends so that we can sanitize just this initial part inside StreamFilter without waitig for the rest of page'a data.
>
> I admit this is hacky and there are holes in it. Well, I see no better way to solve this for now.

To be fair, *anything* is going to be a hack; the sensible way of implementing this would involve modifying the browsers themselves ;).

> I now suspect it might be possible to improve sanitize_document.js to only sanitize and then restore the original to the document and detach from it, thus solving the performance issue and removing the need for so extensive use of StreamFilter. I feel too tired to do this now, though.

I'll try and do this today.

#18 - 08/26/2021 09:54 AM - koszko

> I'll try and do this today.

If it turns out to work, you should be able to use StreamFilter code from 6b53d6c840140fc5df6d7638808b978d96502a35 as it is. As to sanitize_document.js, most of it could probably be thrown away, because:

1. The facility to re-serialize the document is not going to be needed.
2. If we just need to wait for <head> to finish loading, we might get away without using the subtree: true option with MutationObserver. That could help us not slow the page load down too much (mutations with over 1300 records can easily happen otherwise).

#19 - 09/01/2021 01:49 PM - koszko

Did you have any success?

#20 - 09/02/2021 09:37 PM - koszko

I pushed some code for this to new koszko-rethinked-meta-sanitizing branch. I am not yet 100% sure this will work. Will test

#21 - 09/03/2021 09:51 AM - jahoti

Sorry I didn't see your question! I distracted myself with researching around the topic (in the midst of general busyness), and therefore never actually finished actually doing this like I said I would.

I'll test your implementation now.

Also, as to make something of wasted time, notes on some of the more relevant discoveri(es) for this:

- On Chromium, nodes injected by content scripts are CSP-exempt, meaning CSP filtering is unnecessary (albeit harmless). Some work was done in Firefox 58 towards similar behavior, which wasn't followed up and leaves us with a 5-year-old open request https://bugzilla.mozilla.org/show_bug.cgi?id=1267027.
- Both browser families use a simplifying behavior for document structure: (nearly) all nodes go in document.documentElement, under which everything prior to the first "body tag" (i.e. <body>, , etc.) goes in document.head and everything on or after in document.body. This eliminates the many corner cases and- in particular- will make it possible to simplify the new document sanitizer (which I'll do if it works).

#22 - 09/03/2021 10:32 AM - koszko

> - On Chromium, nodes injected by content scripts are CSP-exempt, meaning CSP filtering is unnecessary (albeit harmless).

That would explain some of the behavior I've been experiencing (e.g. with uBO failing to block some of our scripts). Thanks!

> Some work was done in Firefox 58 towards similar behavior, which wasn't followed up and leaves us with a 5-year-old open request https://bugzilla.mozilla.org/show_bug.cgi?id=1267027.

So we still need workarounds under Mozilla :/

Also, we might still need to remove some CSP rules (maybe even more that we do now; thinking about frame-src, img-src, etc.) in case scripts injected by us need to add stuff to the page (this "stuff" could be other scripts or maybe stuff like frames, images, styles, etc.). If I understand correctly, elements added by those injected scripts will not be CSP-exempt...

Perhaps we should somehow pass our nonce to injected scripts so they can use it? Or we could make nonce-less "everything-allowed" execution an optional permission our payload scripts can ask for?

> - Both browser families use a simplifying behavior for document structure: (nearly) all nodes go in document.documentElement, under which everything prior to the first "body tag" (i.e. <body>, , etc.) goes in document.head and everything on or after in document.body. This eliminates the many corner cases and- in particular- will make it possible to simplify the new document sanitizer (which I'll do if it works).

I earlied discovered experimentally that this is the case for Element nodes. Things like comment nodes, though, can appear even outside <html>. Not that it is really necessary to check for this. If not doing this significantly simplifies the code, you can go on with your planned modifications :)

> So we still need workarounds under Mozilla :/

How easy life would be if everything worked reasonably well!

> Also, we might still need to remove some CSP rules (maybe even more that we do now; thinking about frame-src, img-src, etc.) in case scripts injected by us need to add stuff to the page (this "stuff" could be other scripts or maybe stuff like frames, images, styles, etc.). If I understand correctly, elements added by those injected scripts will not be CSP-exempt...

Correct; in fact, this applies on Chromium too, as the CSP-exemption is limited to anything added first-hand by a content script.

> Perhaps we should somehow pass our nonce to injected scripts so they can use it? Or we could make nonce-less "everything-allowed" execution an optional permission our payload scripts can ask for?

Can nonces be added to anything other than scripts? If so, using them sounds best to me provided we take precautions to avoid exposing the nonce to the rest of the page.

For that- and perhaps other permissions- should we try wrapping our scripts in their own scope?

> I earlied discovered experimentally that this is the case for Element nodes. Things like comment nodes, though, can appear even outside . Not that it is really necessary to check for this. If not doing this significantly simplifies the code, you can go on with your planned modifications :)

Ah- that would make sense. I'll look into this first thing tomorrow (the internet chooses the worst moments to go down) and push any changes.

#24 - 09/03/2021 12:23 PM - koszko

If any part of Hachette can be considered infrastructure trap, it's surely this CSP stuff. Having already done so much work, we face the need to additionaly care about possible CSP rules blocking images, styles, etc. Honestly, now, as we are more or less set up to be able to modify sites' CSP before it is applied, the safest thing will be to drop it all for pages on which we want to inject some payload.

And we could implement those permissions later on.

> Can nonces be added to anything other than scripts?

Well, I never wondered about it. Can't they?

> For that- and perhaps other permissions- should we try wrapping our scripts in their own scope?

How? And what for? Since long-term we're not really planning to allow our scripts to run together with page's ones (i.e. "allow site's scripts" is going to exclude "inject our scripts"), this does not seem to be needed. Unless you have some idea why we should facilitate running injected scripts along page's own ones?

Perhaps scoping our scripts would also make sense in cases a few of them are going to be run together (e.g. CloudFlare email decryptor and heuristic-based automatic preloader remover, both of which would make sense to be configured to run by default on all sites)

Actually, it seems the userScripts.register() API function, in scenarios it is available, would give injected scripts their own scope with additional sandbox

#25 - 09/04/2021 12:35 PM - koszko

- % Done changed from 90 to 100

- Status changed from In Progress to Closed

Merged to master

#26 - 09/05/2021 02:20 AM - jahoti

> If any part of Hachette can be considered infrastructure trap, it's surely this CSP stuff. Having already done so much work, we face the need to additionaly care about possible CSP rules blocking images, styles, etc. Honestly, now, as we are more or less set up to be able to modify sites' CSP before it is applied, the safest thing will be to drop it all for pages on which we want to inject some payload.

Very, very true; not to mention that content security policies attempt to address security issues that in part stem from the overreach of the web by granting more power to web developers, which is part of what we're trying to fix!

> Can nonces be added to anything other than scripts?
> Well, I never wondered about it. Can't they?

They can, very pleasingly; nevertheless, it's proving painfully difficult to get a list of exactly where they can be used. Experiments may be needed for whatever we wish to test.

> Since long-term we're not really planning to allow our scripts to run together with page's ones (i.e. "allow site's scripts" is going to exclude "inject our scripts"),

My mistake- I assumed that was enabled intentionally! Should I play with the options page to try and ensure that can't be done, or is it still meant to be possible to enable both on a page manually if they really want to?

> Perhaps scoping our scripts would also make sense in cases a few of them are going to be run together (e.g. CloudFlare email decryptor and heuristic-based automatic preloader remover, both of which would make sense to be configured to run by default on all sites)
>
> Actually, it seems the userScripts.register() API function, in scenarios it is available, would give injected scripts their own scope with additional sandbox

Such scoping could be conducted manually anyway, which makes this purely a policy issue with no need for support in the extension.

#27 - 09/06/2021 10:24 AM - koszko

> Since long-term we're not really planning to allow our scripts to run together with page's ones (i.e. "allow site's scripts" is going to exclude "inject our scripts"),

> My mistake- I assumed that was enabled intentionally! Should I play with the options page to try and ensure that can't be done[...]?

Feel free to do this if you want :)
Keep in mind, however, options_main.js is currntly the most tangled script file in Hachette

#28 - 09/06/2021 11:41 AM - jahoti

> Keep in mind, however, options_main.js is currntly the most tangled script file in Hachette

Perhaps I'll start on that issue first :/.