Site whitelisting

Note: this page needs to be updated to reflect changes made in Haketilo 1.0-beta1

Blocking of scripts is tricky. LibreJS rewrites page's HTML as it is downloaded. For this it uses webRequest along with some other, Mozilla-specific API. That only works for HTTP(s) documents and is generally not an option for us, as we wanted to support Chromium too.

The proper way of blocking scripts, even for non-HTTP documents (like those opened from file://) is with a Content Security Policy (CSP). From WebExtension's content script it is possible to add an appropriate <meta> tag under <head> to cause all scripts but those later injected by us to be blocked through CSP.

It is worth noting that for HTTP(s) documents we're additionally using the webRequest API to inject an HTTP response header containing the rule. The problems described below are therefore only relevant for other protocols like file://.

Whitelisting of sites turned out to be problematic in this case. Extension's content script injected to pages needs to decide immediately whether to block their scripts or not. Removing a CSP <meta> tag after it is added does not cause its CSP rule to be lifted. Re-injecting site's own scripts is also likely to fail under some browser/site configurations. The possibility of site relying on synchronous scripts using the document.write() API function makes most hacks we could think of unreliable. Hence, we decided a whitelisted page should not have its scripts blocked in the first place. It seems easy but the content script that injects the CSP rule has to know whether the page it is running on is whitelisted or not. It could get that information from local storage or from background scripts but that would happen asynchronously and it would then be too late to inject a CSP rule if the page is not to be whitelisted.

We came up with 2 solutions. The first was to redirect any whitelisted URL to itself with a "#something" added to it. That "something" (not literal "something", btw) is a value that indicates the page should be whitelisted. Content script then synchronously checks the URL for presence of that whitelist indicator and based on that either injects the CSP rule tag or not. Content script is then also responsible for quickly removing the "#something" from the URL bar so as not to annoy the user.

The approach described above is now used for file:// and ftp:// (in those few browsers that still support it) protocol pages. The redirection is performed by content script once it learns it applied a script-blocking rule where it is not wanted. For HTTP urls such redirection was at some point performed using WebRequest but it lead to injected string not being removed from browser's URL bar in cases where page failed to load and Haketilo's content scripts were not executed.

A second approach, now used for HTTP(s) pages, is smuggling a similar value in a cookie. This is performed by injecting an HTTP response Set-Cookie header using webRequest API. The cookie, itself having a short TTL value set (30 seconds) is immediately deleted by content script and an additional webRequest listener also checks outgoing HTTP requests for cookies that might have slipped through in order to stop them from leaking to sites' servers.

Now, the challenge was to make the smuggled "something" unpredictable for website owners to avoid tricking the extension into whitelisting a page. The solution was to derive "something" from a value that's unique to a given browser instance and that can be retrieved in a content script synchronously. In Mozilla browsers, each extension is assigned a unique per-session id that happens to be part of getURL()'s return value. This was sufficient and we used it. In Chromium browsers there is no per-session id (extension's main id is used everywhere instead), so we resorted to using the "key" value from extension's manifest (we're also considering using a synchronous AJAX call to fetch a file bundled with the extension), assuming it will be different in every browser instance (which would be possible to automate when for example installing the extension from a GNU/Linux distro's repo).

Additionally, for HTTP(s) pages, webRequest API is used to inject CSP headers as a supplemental script-blocking method and a safety measure.

Couldn't we look into other extensions (e.g. NoScript) to see how they solve this problem? We could and we did, but NoScript's solution of "freezing" page's elements on non-HTTP(s) pages and "unfreezing" them afterwards did not present satisfactory reliability.

In the upcoming Manifest v3 port there is another way to solve this - we can dynamically inject content scripts at runtime. We could simply set up a new content script for every whitelisted URL pattern and that script would immediately know it has to allow the scripts there.