Intervening against document.write()

By Paul Kinlan

Paul is a Developer Advocate

Have you recently seen a warning like the following in your Developer Console in Chrome and wondered what it was?

```
(index):34 A Parser-blocking, cross-origin script,
https://paul.kinlan.me/ad-inject.js, is invoked via document.write().
This may be blocked by the browser if the device has poor network connectivity.
```

Composability is one of the great powers of the web, allowing us to easily integrate with services built by third parties to build great new products! One of the downsides of composability is that it implies a shared responsibility over the user experience. If the integration is sub-optimal, the user experience will be negatively impacted.

One known cause of poor performance is the use of `document.write()` inside pages, specifically those uses that inject scripts. As innocuous as the following looks, it can cause real issues for users.

```
document.write('<script src="https://paul.kinlan.me/ad-inject.js"></script>');
```

Before the browser can render a page, it has to build the DOM tree by parsing the HTML markup. Whenever the parser encounters a script it has to stop and execute it before it can continue parsing the HTML. If the script dynamically injects another script, the parser is forced to wait even longer for the resource to download, which can incur one or more network roundtrips and delay the time to first render of the page

For users on slow connections, such as 2G, external scripts dynamically injected via `document.write()` can delay the display of main page content for tens of seconds, or cause pages to either fail to load or take so long that the user just gives up. Based on instrumentation in Chrome, we've learned that pages featuring third-party scripts inserted via `document.write()` are typically twice as slow to load than other pages on 2G.

We collected data from a 28 day field trial on 1% of Chrome stable users, restricted to users on 2G connections. We saw that 7.6% of all page loads on 2G included at least one cross-site, parser-blocking script that was inserted via `document.write()` in the top level

document. As a result of blocking the load of these scripts, we saw the following improvements on those loads:

- **10%** more page loads reaching first contentful paint (a visual confirmation for the user that the page is effectively loading), **25%** more page loads reaching the fully parsed state, and **10%** fewer reloads suggesting a decrease in user frustration.
- **21%** decrease of the mean time (over one second faster) until the first contentful paint
- **38%** reduction to the mean time it takes to parse a page, representing an improvement of nearly six seconds, dramatically reducing the time it takes to display what matters to the user.

With this data in mind, Chrome, starting with version 55, intervenes on behalf of all users when we detect this known-bad pattern by changing how `document.write()` is handled in Chrome (See Chrome Status). Specifically Chrome will not execute the `<script>` elements injected via `document.write()` when **all** of the following conditions are met:

1. The user is on a slow connection, specifically when the user is on 2G. (In the future, the change might be extended to other users on slow connections, such as slow 3G or slow WiFi.)
2. The `document.write()` is in a top level document. The intervention does not apply to document.written scripts within iframes as they don't block the rendering of the main page.
3. The script in the `document.write()` is parser-blocking. Scripts with the '`async`' or '`defer`' attributes will still execute.
4. The script is not hosted on the same site. In other words, Chrome will not intervene for scripts with a matching eTLD+1 (e.g. a script hosted on js.example.org inserted on www.example.org).
5. The script is not already in the browser HTTP cache. Scripts in the cache will not incur a network delay and will still execute.
6. The request for the page is not a reload. Chrome will not intervene if the user triggered a reload and will execute the page as normal.

Third party snippets sometimes use `document.write()` to load scripts. Fortunately, most third parties provide asynchronous loading alternatives, which allow third party scripts to load without blocking the display of the rest of the content on the page.

How do I fix this?

This simple answer is don't inject scripts using `document.write()`. We are maintaining a set of known services that provide asynchronous loader support that we encourage you to keep checking.

If your provider is not on the list and does support asynchronous script loading then please let us know and we can update the page to help all users.

If your provider does not support the ability to asynchronously load scripts into your page then we encourage you to contact them and let us and them know how they will be affected.

If your provider gives you a snippet that includes the `document.write()`, it might be possible for you to add an `async` attribute to the script element, or for you to add the script elements with DOM API's like `document.appendChild()` or `parentNode.insertBefore()` much like Google Analytics does.

How to detect when your site is affected

There are a large number of criteria that determine whether the restriction is enforced, so how do you know if you are affected?

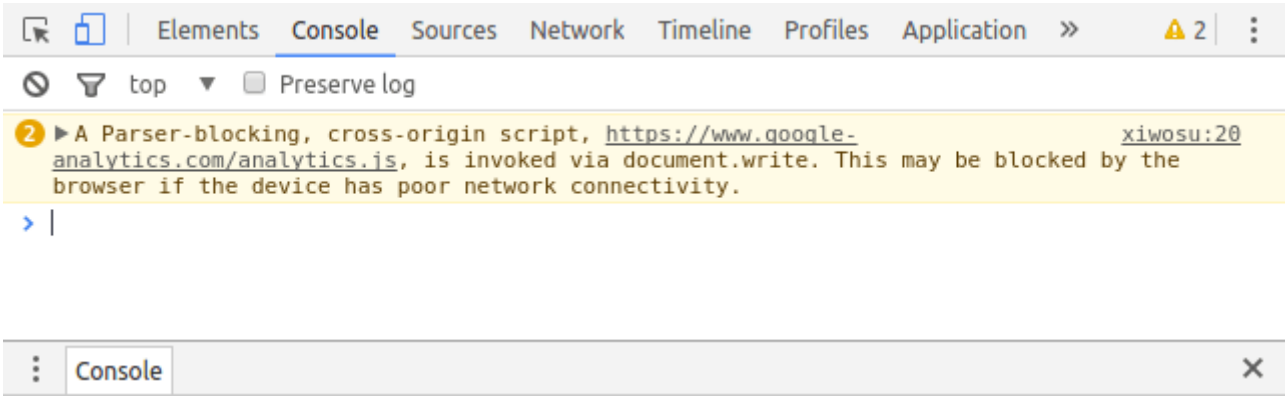
Detecting when a user is on 2G

To understand the potential impact of this change you first need to understand how many of your users will be on 2G. You can detect the user's current network type and speed by using theNetwork Information API that is available in Chrome and then send a heads-up to your analytic or Real User Metrics (RUM) systems.

```
if(navigator.connection &&
   navigator.connection.type === 'cellular' &&
   navigator.connection.downlinkMax <= 0.115) {
  // Notify your service to indicate that you might be affected by this restricti
}
```

Catch warnings in Chrome DevTools

Since Chrome 53, DevTools issues warnings for problematic `document.write()` statements. Specifically, if a `document.write()` request meets criteria 2 to 5 (Chrome ignores the connection criteria when sending this warning), the warning will look something like:



Seeing warnings in Chrome DevTools is great, but how do you detect this at scale? You can check for HTTP headers that are sent to your server when the intervention happens.

Note: You can enable "2G" mode in Chrome DevTools, however in Chrome 55 it doesn't properly trigger the intervention. The best option is to enable this intervention using `chrome://flags/#disallow-doc-written-script-loads`.

Note: One of the heuristics for this intervention to detect if the file is not already in the user's Cache. DevTools allows you to disable the Cache, however you will not see the `onerror` event on the script element being triggered, but you will see it when not using DevTools. This is already fixed in Chrome 56.

Check your HTTP headers on the script resource

When a script inserted via **`document.write`** has been blocked, Chrome will send the following header to the requested resource:

```
Intervention: <https://shorturl/relevant/spec>;
```

When a script inserted via **`document.write`** is found and could be blocked in different circumstances, Chrome might send:

```
Intervention: <https://shorturl/relevant/spec>; level="warning"
```

The intervention header will be sent as part of the GET request for the script (asynchronously in case of an actual intervention).

What does the future hold?

The initial plan is to execute this intervention when we detect the criteria being met. We started with showing just a warning in the Developer Console in Chrome 53. (Beta was in July 2016. We expect Stable to be available for all users in September 2016.)

We will intervene to block injected scripts for 2G users tentatively starting in Chrome 54, which is estimated to be in a stable release for all users in mid-October 2016. Check out the Chrome Status entry for more updates.

Over time, we're looking to intervene when any user has a slow connection (i.e, slow 3G or WiFi). Follow this Chrome Status entry.

Want to learn more?

To learn more, see these additional resources:

- Specification of the document.write() intervention and its feedback loops
- Chrome Status (intervention for users on 2G)
- Chrome Status (intervention for users on effectively slow connections)
- Design document
- Additional rationale for this effort
- blink-dev Intent to implement thread

Last updated July 2, 2018.