Meltdown/Spectre

By Surma

Surma is a contributor to Web**Fundamentals**

Overview

On January 3rd Project Zero revealed vulnerabilities in modern CPUs that a process can use to read (at worst) arbitrary memory — including memory that doesn't belong to that process. These vulnerabilities have been named Spectre and Meltdown. What is Chrome doing to help keep the web secure, and what should web developers do for their own sites?

TL; DR

As a **user browsing the web**, you should make sure you keep your operating system and your browser updated. In addition, Chrome users can consider enabling Site Isolation.

If you are a **web developer**, the Chrome team advises:

- Where possible, prevent cookies from entering the renderer process' memory by using the `SameSite` and `HTTPOnly` cookie attributes, and by avoiding reading from `document.cookie`.
- Make sure your MIME types are correct and specify an `X-Content-Type-Options: nosniff` header for any URLs with user-specific or sensitive content, to get the most out of Cross-Origin Read Blocking for users who have Site Isolation enabled.
- Enable Site Isolation and let the Chrome team know if it causes problems for your site.

If you are wondering *why* these steps help, read on!

The risk

There have been a wide variety of explanations of these vulnerabilities, so I am not going to add yet another one. If you are interested in how these vulnerabilities can be exploited, I recommend taking a look at the blog post by my colleagues from the Google Cloud team.

Both Meltdown and Spectre potentially allow a process to read memory that it is not supposed to be able to. Sometimes, multiple documents from different sites can end up sharing a process in Chrome. This can happen when one has opened the other using `window.open`, or `<a href="..." target="_blank">`, or iframes. If a website contains user-specific data, there is a chance that another site could use these new vulnerabilities to read that user data.

Mitigations

There are multiple efforts the Chrome and V8 engineering team is deploying to mitigate this threat.

Site Isolation

The impact of successfully exploiting Spectre can be greatly reduced by preventing sensitive data from ever sharing a process with attacker-controlled code. The Chrome team has been working on a feature to achieve this called "Site Isolation":

> "Websites typically cannot access each other's data inside the browser[...]. Occasionally, security bugs are found in this code and malicious websites may try to bypass these rules to attack other websites. [...] Site Isolation offers a second line of defense to make such attacks less likely to succeed. **It ensures that pages from different websites are always put into different processes, each running in a sandbox that limits what the process is allowed to do.**"

Site Isolation has not been enabled by default yet as there are a couple of known issues and the Chrome team would like as much field testing as possible. If you are a web developer, you should enable Site Isolation and check whether your site remains functional. If you'd like to opt-in now, enable `chrome://flags#enable-site-per-process`. If you find a site that doesn't work, please help us by filing a bug and mention that you have Site Isolation enabled.

Cross-Site Document Blocking

Note: After the publication of this document, Cross-Site Document Blocking was renamed to Cross-Origin Read Blocking. These two terms refer to the same concept.

Even when all cross-site pages are put into separate processes, pages can still legitimately request some cross-site subresources, such as images and JavaScript. To help prevent sensitive information from leaking this information, Site Isolation includes a "cross-site document blocking" feature that limits which network responses are delivered to the renderer process.

A website can request two types of data from a server: "documents" and "resources". Here, documents are HTML, XML, JSON, and text files. A website is able to receive documents from its own domain or from other domains with permissive CORS headers. Resources include things like images, JavaScript, CSS and fonts. Resources can be included from any site.

The cross-site document blocking policy prevents a process from receiving "documents" from other origins if:

1. They have an HTML, XML, JSON, or text/plain MIME type, and
2. They have either a `X-Content-Type-Options: nosniff` HTTP response header, or a quick content analysis ("sniffing") confirms that the type is correct
3. CORS doesn't explicitly allow access to the document

Documents that are blocked by this policy are presented to the process as empty, although the request still happens in the background.

For example: Imagine an attacker creating an `<img>` tag that includes a JSON file with sensitive data, like `<img src="https://yourbank.com/balance.json">`. Without Site Isolation, the contents of the JSON file would make it to the renderer process's memory, at which point the renderer notices that it is not a valid image format and doesn't render an image. With Spectre, however, there is now a way to potentially read that chunk of memory. Cross-site document blocking would prevent the contents of this file from ever entering the memory of the process the renderer is running in because the MIME type is blocked by cross-site document blocking.

According to user metrics, there are a lot of JavaScript and CSS files that are delivered with `text/html` or `text/plain` MIME types. To avoid blocking resources that are accidentally marked as documents, Chrome attempts to sniff the response to ensure the MIME type is correct. This sniffing is imperfect, so if you are sure that you are setting the correct `Content-Type` headers on your website, the Chrome team recommends adding the `X-Content-Type-Options: nosniff` header to all your responses.

If you want to try cross-site document blocking, opt-in to Site Isolation as described above.

`SameSite` cookies

Let’s go back to the example above: `<img src="https://yourbank.com/balance.json">`. This only works if yourbank.com has stored a cookie that automatically logs the user in. Cookies typically get sent for all requests to the website that sets the cookie — even if the request is made by a third party using an `<img>` tag. SameSite cookies are a new attribute that specify that a cookie should only be attached to a request that originates from the same site, hence the name. Sadly, at the time of writing, only Chrome and Firefox 58+ support this attribute.

`HTTPOnly` and `document.cookie`

If your site's cookies are only used server-side, not by client JavaScript, there are ways you can stop the cookie's data from entering the renderer process. You can set the `HTTPOnly` cookie attribute, which explicitly prevents the cookie from being accessed through client side script on supported browsers, such as Chrome. If setting `HTTPOnly` isn't possible, you can help limit the exposure of loading cookie data to the rendered process by not reading `document.cookie` unless absolutely necessary.

Open External Links Using `rel="noopener"`

When you link to another page using `target="_blank"`, the opened page has access to your `window` object, can navigate your page to a different URL, and without Site Isolation will be in the same process as your page. To better protect your page, links to external pages that open in a new window should always specify `rel="noopener"`.

High-resolution timers

To exploit Meltdown or Spectre, an attacker needs to measure how long it takes to read a certain value from memory. For this, a reliable and accurate timer is needed.

One API the web platform offers is `performance.now()` which is accurate to 5 microseconds. As a mitigation, all major browsers have decreased the resolution of `performance.now()` to make it harder to mount the attacks.

Another way to get a high-resolution timer is to use a SharedArrayBuffer. The buffer is used by a dedicated worker to increment a counter. The main thread reads this counter and uses that as a timer. For the time being browsers have decided to disable SharedArrayBuffer until other mitigations are in place.

V8

To exploit Spectre, a specifically crafted sequence of CPU instructions is needed. The V8 team has implemented mitigations for known attack proofs of concept, and is working on changes in TurboFan, their optimizing compiler, that make its generated code safe even when these attacks are triggered. However, these code generation changes may come at a performance penalty.

Keeping the web safe

There has been a lot of uncertainty around the discovery of Spectre and Meltdown and their implications. I hope this article shed some light on what the Chrome and V8 teams are doing to keep the web platform safe, and how web developers can help by using existing security features. If you have any questions, feel free to reach out to me on Twitter.

Last updated July 12, 2018.