Servo

Spectre and the interfaces between browser, operating system and hardware

2.28.2019

Alan Jeffrey
Research Engineer
1. Servo browser architecture
   What is Servo, and what is its architecture?

2. Spectre and multiprocess
   What is Spectre, and how does it impact a browser?

3. Hardware/OS support for browsers
   What could a browser hope from its environment?
Servo browser architecture
What is Servo?

Mozilla’s next-generation browser engine

- Designed for modern CPUs (multicore) and GPUs (shaders)
- Implemented in Rust (memory safety)
- Targeting Mixed Reality (VR, AR, XR,...)
Browser architecture

Highly simplified (i.e. contains lies)
Multithreaded browser architecture

Independent computations can be run concurrently

Spec requirement: main JS thread
Style doesn't depend on siblings
Layout often doesn't depend on siblings
OS requirement*: main gfx thread

User input

Script

Style

Style

Style

Style

Style

Layout

Layout

Paint

Graphics

JS

HTML

CSS

Spec requirement: main JS thread
Style doesn't depend on siblings
Layout often doesn't depend on siblings
OS requirement*: main gfx thread

*on some OSs

Different colors represent different threads
Multithreaded browser architecture

Different security domains are independent

Spec requirement: main JS thread but limited x-domain communication

Style doesn't depend on siblings

Layout often doesn't depend on siblings

OS requirement*: main gfx thread

User input

Script for alice.com

bob.edu

charlie.org

Style

Style

Style

Layout

Layout

Layout

Paint

JS

HTML

CSS

Graphics

Content the user wants to see

Adverts, trackers, bitcoin miners, etc.

*on some OSs
Multiprocess browser architecture

Different security domains need to be in different address spaces*

*reasons coming up
Spectre and multiprocess
What is Spectre?
In which I tell you things you already know

• Exploits μarchitecture (caching and branch prediction)
• Abstraction has leaked
• Allows bypassing of dynamic security checks

### Simplified Spectre v1 attack:

```plaintext
if (canRead(SECRET)) { a[SECRET] = 1; } else if (touched(a[0])) { secretMustHaveBeen = 0; } else if (touched(a[1])) { secretMustHaveBeen = 1; }
```

Attacker checks if a memory location has been used
High-security SECRET flowed to low-security variable

Attacker fails this!
This code doesn't run (or so the dev thinks)

### 1. Introduction

Computations performed by physical devices often leave observable side effects beyond the computation's nominal outputs. Side-channel attacks focus on exploiting these side effects to extract otherwise-invaluable secret information. Since their introduction in the late ’90s [15], many physical effects such as power consumption [41, 42], electromagnetic radiation [59], or acoustic noise [20] have been leveraged to extract cryptographic keys as well as other secrets.

Physical side-channel attacks can also be used to extract secret information from complex devices such as PC and mobile phones [21, 22]. However, because these devices often execute code from a potentially unknown origin, they face additional threats in the form of software-based attacks, which do not require external measurement equipment. While some attacks exploit software vulnerabilities (such as buffer overflows) [5] or double-fault errors [12], other software attacks leverage hardware vulnerabilities to leak sensitive information.

Attacks of the new suite include microarchitectural attacks exploiting cache timing [8, 30, 48, 52, 55, 66, 74], branch prediction history [1, 2], branch target buffers [14, 44] or open DRAM rows [16]. Software-based techniques have also been used to mount fault attacks that alter physical memory [39] or internal CPU values [16].

Several microarchitectural design techniques have facilitated the increase in processor speed over the past decades. One such advancement is speculative execution, which is widely used to increase performance and involves having the CPU pass likely future execution decisions and prematurely execute instructions on these paths. More specifically, consider an example where the program’s control flow depends on an untrusted value located in external physical memory. As this memory is much slower than the CPU, it often takes several hundred clock cycles before the value becomes known. Rather than waiting these cycles by idling, the CPU attempts to guess the direction of control flow, saves a checkpoint of its register state, and proceeds to speculatively execute the program on the guessed path. When the value eventually arrives from memory, the CPU checks the correctness of its internal guess. If the guess was wrong, the CPU discards the incorrect speculative execution by restoring the register state back to the point of checkpoint, resulting in performance comparable to idling. However, if the guess was correct, the speculative execution results are committed, yielding a significant performance gain as useful work was accomplished during the delay.

From a security perspective, speculative execution involves executing a program in possibly incorrect ways. However, because CPUs are designed to maintain functional correctness by reverting the results of incorrect speculative executions to their prior states, these errors were previously assumed to be safe.

### 4. Our Results

In this paper, we analyze the security implications of such incorrect speculative execution. We present a class of microarchitectural attacks which we call Spectre attacks. At a high level, Spectre attacks trick the processor into speculatively executing instruction sequences that should not have been executed under correct program execution. As the effects of these instructions on the nominal CPU state are eventually unknown.
Why do browser vendors care?

We’re in the “sandboxed execution of untrusted code” business

Spectre is implementable in JavaScript.
Dynamic security checks are on array bounds.

\[
a[b[indexWayOutOfBounds]] = 1;
\]

Code executed by browser

\[
\text{if (indexWayOutOfBounds < b.length) }
\]
\[
\text{tmp = *(b + indexWayOutOfBounds);} \\
\text{if (tmp < a.length) } \{ \text{*(a + tmp) = 1; } \}
\]

JS from evil.com

Allows scanning of entire address space!
Dealing with a leaky abstraction

Spectre is not the first time this has happened

- Instruction reordering in μarchitecture
- Used to be behind an abstraction barrier (devs thought in terms of sequential consistency)
- This abstraction leaked: weak memory models
- Will speculative evaluation go the same way?
What are browser vendors doing about it?

Spectre mitigation

Short term fixes:
- Disable high-precision timers
- Disable shared-memory concurrency

Medium term fixes:
- Run security from different security domains in different address spaces

Long term fixes:
- Whatever hardware/OSs come up with
Hardware / OS support for browsers
Hardware/OS support for browsers

My wishlist by me

CPUs

- Spectre mitigation
- Security boundaries inside a process
- Call stack protection
- Async all the things  
  (blocking → polling + callbacks)

GPUs

- GPUs/drivers as stable as CPUs/compilers
- Cross-platform open gfx stack
- Fast shared memory between CPU and GPU
- Video encoding support  
  (e.g. Servo in the cloud)
Thank You