17-355 Research Project Proposal: Detecting and Flagging Self-Timing Code

Cam Wong, Joe Doyle

April 10, 2018

1 Motivation

"Timing attacks" are a broad set of attacks that can be used to covertly transmit information or inspect the internal workings of otherwise-isolated programs. Modern microprocessors have many complex execution strategies for their instruction sets, but they are designed so that every execution path is semantically indistinguishable (at least, in theory). However, although the semantics stay the same, performance characteristics – primarily how much time a sequence of instructions takes to complete – can vary depending on the state of the processor and the particular execution strategy used. When performance-critical resources (such as caches) are shared, processes can influence these resources to signal across VM sandboxes, or they can watch cache effects to try to extract security-critical data, or (in the case of Spectre) they can take advantage of processor flaws to inspect kernel memory.

There is existing work doing dynamic analysis of programs to detect attempts to certain kinds of attacks in progress, and there are analysis tools and techniques to make cryptography software less vulnerable to this kind of attack, but to our knowledge there is no existing framework for statically analyzing whether a piece of software attempts to invoke a timing-based attack.

2 Project Description

Our goals for this project are to provide, with reasonable confidence, a list of "critical sections" that are possibly being measured via an "internal timer". We intend to do this by examining calls to (and stores of calls to) "timer functions" (a list of which can be passed into the analysis as an input).

For a proof of concept, we will be performing the analysis on C code, to leverage the more mature parser libraries. A "real world" implementation of this analysis might instead work with Javascript or x86. We will also only be focusing on single procedure analyses, but may consider extending to full-program analysis after the checkpoint should time permit.

For the checkpoint, we aim to have some scaffolding that will demarcate all calls to timer functions and where their results are used. From there, we can extend the analysis to detect whether those results are compared in some way (such as subtracting them), and use that to determine critical sections.

3 Papers

We'd like to approach this as a research variant of the project. As such, we've identified several papers that could be relevant. Most of these papers are on arXiv, rather than coming from journals or conferences – we had a difficult time finding relevant papers in journals, so if any of the instructors have suggestions for relevant conference papers, we'd be happy to have them.

The blog post on Spectre: `https://googleprojectzero.blogspot.com/2018/01/reading-privileged-memory-with-side.html`, and the paper: `https://spectreattack.com/spectre.pdf`.

“MeltdownPrime and SpectrePrime: Automatically-Synthesized Attacks Exploiting Invalidation-Based Coherence Protocols”: `https://arxiv.org/abs/1802.03802`

“The Spy in the Sandbox – Practical Cache Attacks in Javascript”: `TheSpyintheSandbox--PracticalCacheAttacksinJavascript`

“Systematic Classification of Side-Channel Attacks: A Case Study for Mobile Devices”: `https://arxiv.org/abs/1611.03748`

“Robust and Efficient Elimination of Cache and Timing Side Channels”: `https://arxiv.org/abs/1506.00189`

“Rigorous Analysis of Software Countermeasures against Cache Attacks”: `https://arxiv.org/abs/1603.02187`