Proc Freezer
A Mitigation Technique Against Flush+Flush Style Cache Side-Channel Attacks

Brendan Dolan-Gavitt — Rafael De Los Santos
NYU Tandon School of Engineering

May 2017

1 Abstract

Cache side-channel attacks have been shown to exploit vulnerabilities in CPU cache hierarchies by spying on victim processes through shared memory pages. The Flush+Flush attack in particular creates no cache misses, and is therefore not detectable by standard methods that rely on CPU performance monitoring alone. In this paper, we present [name of thing], a novel detection mechanism for Flush+Flush attacks that works without knowing which process is causing cache misses. By utilizing the Linux cgroup freezer, [Name of thing] can isolate a malicious Flush+Flush attack through a simple binary search of userland processes.

In this paper, we show that [Name of thing] is effective at isolating a Flush+Flush attack on OpenSSL T-Table AES encryption running in the userspace of a Linux machine. We also demonstrate that [Name of thing] need not be triggered via user interaction, and is a viable oracle for detection of Flush+Flush attacks.

2 Introduction

Flush+Flush and other cache side-channel attacks operate by spying on shared memory pages that are stored in CPU cache memory after usage. By flushing a specific cache line with the clflush instruction and measuring its execution time, the attacker will gain information on whether or not that line has been used by the victim. As many attacks of this nature invoke memory accesses to achieve this, most state of the art detection methods utilize hardware performance counters to detect malicious behavior, such as unusually high cache misses, which can be pinpointed to a single process.

Unlike other cache attacks however, Flush+Flush makes no memory accesses, therefore rendering it undetectable through hardware performance monitoring alone. Even so, by using clflush to continuously flush data used by the victim, system cache misses will still increase. For this reason, we propose [Name of thing], which utilizes this system wide abnormality to begin its inquiry. When an abnormal cache miss rate is detected across all cores on the system, [Name of Thing] will binary search through the list of userland processes, freezing one half using Linux's cgroup freezer, and checking whether or not the cache miss rate is being triggered somewhere in the second half.

[Name of thing] is meant to act as a system oracle that can run as a background process and be triggered automatically if the system cache miss rate reaches a critical level. In this paper, we demonstrate that [Name of thing] is effective at isolating a Flush+Flush attack on OpenSSL T-Table AES encryption running in the userspace of a Linux machine [more information about system needed]. Most importantly, we show that the load of [Name of thing], both in the idle cache monitoring phase and binary search phase, is low enough to impact the system no greater than any other userland process at runtime [need to rephrase this], and therefore is viable as an always-running oracle program against Flush+Flush cache attacks.

3 background

Cache memory
Page sharing
Flush+Reload and Flush+Flush (or just Flush+Flush)
Binary search?

4 Name of Thing

About construction and operation of [Name of thing]

5 Detection of Flush+Flush

Specifics on the detection of Flush+Flush
perhaps this can be placed in one of the earlier sections

6 Runtime metrics

Did we successfully isolate the attacker?
Can this run in the background? (idle)
Is human input needed? (this is self explanatory, might remove)
Does detection slow down the host? (binary search)

7 Ways around [Name of Oracle]

8 Related Work

9 Works Cited