Investigating Uses of Meltdown For An Attack In A Cloud Computing Scenario

Student: Richard Boyd, Supervisor: Dr. Daniel Page, Project Type: research

University of Bristol, Department of Computer Science

Introduction

The recent Meltdown[a] attack which was widely publicized allows a user of a computer to completely dump the entire physical memory by leveraging out of order execution to load unprivileged memory locations and measure their value with a cache side channel attack. This includes users running code in containerized or paravirtualized environments. This would allow users of certain cloud services where many users are running code within containers or paravirtualized virtual machines to obtain the memory of other users on the same CPU. Say for example that a malicious user running Meltdown dumped all physical memory within a Docker container inside a Kubernetes cluster being used by other customers.

[a] https://meltdownattack.com/meltdown.pdf

1. Project Outline

Given a dump of memory from a machine running Linux known to be running code in a cloud situation such as a container orchestrated behind the scenes by a Kubernetes cluster:

- Can we identify key material of other users or of Kubernetes software?
- Can we use this to use Kubernetes API from a container to schedule resources or obtain logs for example?
- How difficult is it to find useful or interesting data to do with the kernel such as TCP buffers given a dump of memory?
- Can we identify memory belonging to other processes, determine to which process it belongs?

2. Meltdown

The Meltdown attack when running on a machine with Intel TSX can dump the entire physical memory at a rate of 500Kb/s. It achieves this by simply running some code within a process and measuring micro-architectural changes with a Flush+Reload L3 cache side-channel attack. This side channel is created by simply loading memory locations and using them to access a another array. By raising an exception previous to this, the instructions appear to not happen at an architectural level but due to out of order execution they do happen and have an effect on what is cached. Meltdown is quite easily mitigated by implemented Kernel Page Table Isolation.

3. Preliminary Results

Findings so far include

- The Meltdown proof of concept code dumps data in order of 1000 times slower than claimed in the paper when Intel TSX is not available
- Finding key material is rather straightforward - just search for areas of high entropy within data

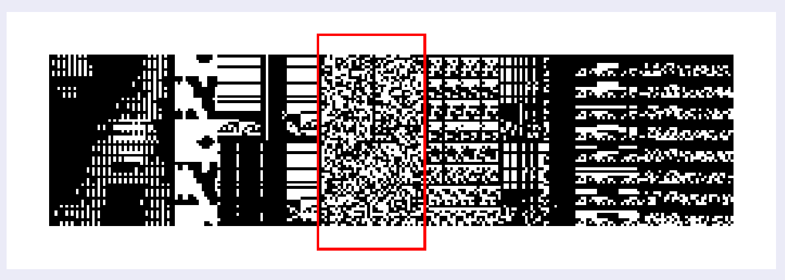

Figure: High entropy keys stand out visually against background low-entropy data. - Playing Hide And Seek With Stored Keys, Adi Shamir and Nicko van Someren, 1998

4. Progress and Status

Complete

- Implemented finding RSA private keys in data
- Implemented using Meltdown PoC to scan memory for a private key relating to a given public key

Next steps

- Implement algorithms for key recovery in erroneous data
- Explore private key recovery further e.g. searching for AES keys, consider EC crypto etc.
- Speed up the Meltdown attacks rate of dumping memory - test out memory loading down a trained branch to avoid exception handling
- Explore what can be done with the private key of the local Kubernetes code running in a node when it is recovered
- Explore