Hardware Side Channels in the Cloud

INTRO

At REcon 2015, I demonstrated a new hardware side channel which targeted co-located virtual machines in the cloud. This attack exploited the CPU’s pipeline as opposed to cache tiers which are often used in side channel attacks. When designing or looking for hardware based side channels - specifically in the cloud - I analyzed a few universal properties which define the ‘right’ kind of vulnerable system as well as unique ones tailored to the hardware medium.

Slides and full research paper found here.

The relevance of these types of attacks will only increase. Especially attacks which target the vulnerabilities inherent to systems which share hardware resources - such as in cloud platforms.

1. virtualization of physical resources
BUT WHAT IS A SIDE CHANNEL ATTACK???

Any meaningful information that you can leak from the environment running the target application or, in this case, the victim virtual machine counts as a side channel. However, some information is better than others. In this case a process (the attacker) must be able to repeatedly record this environment ‘artifact’ from inside one virtual machine.

In the cloud, this environment is the shared physical resources on the service used by the virtual machines. The hypervisor dynamically partitions each resource - this is then seen by a single virtual machine as its private resource. The side channel model (2) illustrates this.

Knowing this, the attacker can affect that resource partition in a recordable way, such as by flushing a line in the cache tier, waiting until the victim process uses it for an operation, then requesting that address again - recording what values are now there.

2. side channel model
ATTACK EXAMPLES

Great! So we can record things from our victim's environment - but now what? Depending on what the victim's process is doing we can actually employ several different types of attacks.

➔ crypto key theft

Crypto keys are great, private crypto keys are even better. Using this hardware side channel, it’s possible to leak the bytes of the private key used by a co-located process. In one scenario, two virtual machines are allocated the same space on the L3 cache at different times. The attacker flushes a certain cache address, waits for the victim to use that address, then queries it again - recording the new values that are there [1].

➔ process monitoring (what applications is the victim running?)

This is possible when you record enough of the target's behavior, i.e. CPU or pipeline usage or values stored in memory. A mapping between the recording to a specific running process can be constructed with a varied degree of certainty. Warning, this does rely on at least a rudimentary knowledge of machine learning.

➔ environment keying (great for proving co-location!)

Using the environment recordings taken off of a specific hardware resource, you can also uniquely identify one server from another in the cloud. This is useful to prove that two virtual machines you control are co-resident on the same physical server. Alternatively, if you know the behavior signature of a server your target is on, you can repeatedly create virtual machines, recording the behavior on each system until you find a match [2].

➔ broadcast signal (receive messages without the internet :0)

If a colluding process is purposefully generating behavior on a pre-arranged hardware resource, such as purposefully filling a cache line with 0’s and 1’s, the attacker (your process) can record this behavior in the same way it would record a victim’s behavior. You then can translate the recorded values into pre-agreed messages. Recording from different hardware mediums results in a channel with different bandwidths [3].
The Cache is Easy, the Pipeline is Harder

Now all of the above examples used the cache to record the environment shared by both victim and attacker processes. It is the most widely used in both literature and practice to construct side channels as well as being the easiest to record artifacts from. Basically everyone loves cache.

The cache isn’t the only shared resource, co-located virtual machines also share the CPU execution pipeline. In order to use the CPU pipeline, we must be able to record a value from it. However, there is no easy way for any process to query the state of the pipeline over time - it is like a virtual black-box. The only thing a process can know is the instruction set order it gives to be executed on the pipeline and the result the pipeline returns.

out of order execution
( the pipeline’s artifact )

We can exploit this pipeline optimization as a means to record the state of the pipeline. The known input instruction order will result in two different return values - one is the expected result(s), the other is the result if the pipeline executions them out-of-order.

3. foreign processes can share the same pipeline
strong memory ordering

Our target, cloud processors, can be assumed to be x86/64 architecture - implying a usually strongly-ordered memory model \[^4\]. This is important because the pipeline will optimize the execution of instructions but attempt to maintain the right order of stores to memory and loads from memory.

…. HOWEVER, the stores and loads from different threads may be reordered by out-of-order-execution. Now this reordering is observable if we’re clever.

recording instruction reorder (or how to be clever)

In order for the attacker to record the “reordering” artifact from the pipeline, we must record two things for each of our two threads.

- input instruction order
- return value

Additionally, the instructions in each thread must contain a \texttt{STORE} to memory and a \texttt{LOAD} from memory. The \texttt{LOAD} from memory must reference the location stored to by the opposite thread. This setup ensures the possibility for the four cases illustrated below. The last is the artifact we record - doing so several thousand times gives us averages over time.
4. the attacker can record when its instructions are reordered

➔ sending a message

To make our attacks more interesting, we want to be able force the amount of recorded out-of-order-executions. This ability is useful for other attacks, such as constructing covert communication channels.

In order to do this, we need to alter how the pipeline’s optimization works - either by increasing the probability that it will or will not reorder our two threads. The easiest is to enforce a strong memory order and guarantee that the attacker will receive less out-of-order-executions.
memory barriers

In the x86 instruction set, there are specific barrier instructions that will stop the processor from reordering the four possible combinations of STORE’s and LOAD’s. What we’re interested in is forcing a strong order when the processor encounters an instruction set with a STORE followed by a LOAD

the instruction mfence does exactly this

By have the colluding process inject these memory barriers in the pipeline, the attacker’s instructions will not be reordered, forcing a noticeable decrease in the recorded averages. Doing this in distinct time frames allows us to send a binary message.

5. mfence ensures the strong memory order on pipeline

FIN

The takeaway is that even with virtualization separating your virtual machine from the hundreds of other alien virtual machines, the pipeline can’t distinguish your process’s instructions from all the other ones and we can use that to our advantage. 😊