Side-Channel Notes

note: this document only contains the main points and eferences to good/usable sections for later use.

1 Contr-Channel Attacks (Paging15)

- virtually no noise, only 1 run required
- bypass 2 shielding systems (like Haven (uses SGX) and InkTag)
- Attacking Hunspell and libjpeg decompression

Challenges

- granularity of 4KB pages
- interrupts/handlers/switching in/out of enclave → large performance overhead
- does not work for off the shelf -O3 etc. optimized code

Requirements

- Memory management by OS
- Application code must be known, and not hardened

Idea

- offline analysis, outside the secure environment
- OS inserts page fault traps to observe input dependent or data dependent accesses
- we record page-fault sequences
- Input dependent accesses:

two addresses are associated with a page fault: The address which triggered the page fault and the instruction that was executed when it happened. When they are equal it's a code page fault, o/w it's a data page fault.

- Only track a small set of relevant pages
- To bypass ASLR we record the very early page faults of an executable, this allows finding the base address of the loader.

Control transfers

- get a couple of page fault traces P_i, and also convert it to page base address trace Q_i.
- For code page faults aka control transfers: identify the target address f. For each occurrence of f in any P_i, search the shortest sequence of preceding pages for which the corresponding sequence in Q_i leads only to f.

Data accesses

- Record all accesses outside the shielding system
- Identify all data page faults
- ∀ data page faults we search for a minimum sequence of code page faults right before, that can be used to uniquely infer the data access.

Implementation

- The custom page-fault handler is installed by overwriting the IDT interrupt descriptor table.
- For InkTag page-fault handler was inserted in the Linux page-handler

2 SGX Cache Attacks r practical (17)

- Nice short background chapters on cache, SGX
- ∃ extended version of this paper! https://arxiv.org/pdf/1702.07521.pdf
- SGX enclaves can disable PMCs by activating a feature called ASCI (Anti Side Channel Interference)

Goal

- Untrusted OS
- Prime & probe attacks on RSA decryption and genomic processing
- OS ensures victim enclave and attack program run on same core → uninerrupted → hard to detect by enclave, do interrupts and PMCs on other process.
- Generate cache access trace by precise L1 monitoring using CPU performance counters (PMC), learn access pattern step by step +1 data access at a time

Limitations

- Does not work against hardened code
- "We compiled enclave RSA decryption code with default optimization flags and compiler settings" =?

Thread model

- Attacker knows mapping of memory addresses to cache lines and can re-init resp replay the enclave and its inputs.
- Attacker controls allocation of resources & core of enclave
- Attacker can modify interrupt handler, frequency of timers etc.
- He can **not** see enclave memory content or registers

Noise Reduction Techniques For not having to run the attacks thousands of times

- Isolated core by modifying the linux scheduler
- Self-pollution exploit the fact that L1D and L1I are separate → less noise
- Uninterrupted execution: avoid AEX (async enclave exit) and the OS's ISR (interrupt service routine). That means configuring the interrupt controller to not deliver interrupts to the core the attack core. The only per-core interrupt is the timer interrupt. So they lowered its frequency.
- Measurement: apparently *reading the TSC* is itself in the order of L1-L2 access time difference → too noisy, use PMC ∃ RDTSC and RDPMC !!

RSA Attack

- Non-hardened version of RSA decryption found in Intel IIP crypto library
- fixed-size sliding window exponentiation using a precomputed multiplier table
- The algorithm iterates over all exponent windows and, depending on the window value, it may perform a multiplication with a value from the table
- monitor a single multiplier access at a time
- each multiplier in the table is 1024 bits, accessing the multiplier causes 16 repeated memory accesses
- repeat this process for a subset of all possible multipliers (10 out of 16 in our case), because extracting 70% of a key is enough. Additional advantage because some cache sets had more noise than others!