Security Analysis of the Computer Architecture *OR*
What a software researcher can teach you about *YOUR* computer

Rodrigo Rubira Branco (BSDaemon)
rodrigo *noSPAM* kernelhacking.com
https://twitter.com/bsdaemon
Disclaimers

• I don’t speak for my employer. All the opinions and information here are of my responsibility (actually no one ever saw this talk before);

• The talk focus mostly in Intel Skylake Architecture (Wikipedia says it will be released 2015-2016). I’ll try to generalize as much as I can though – in that, probably many mistakes might happen, so…

• Interrupt me if you have questions or important comments at any point.
  • IMPORTANT: No, I’m not part of the Intel Security Group (McAfee)
I already anticipate a few questions:

- RNG -> I have no idea how strong it is, I believe it is not as strong as a specialized hardware but probably stronger than usual random generation mechanisms existent in the platform
- AMT -> I’ll talk a bit about the ME, but I’ll let the AMT research by Insinuator [0] to continue ;)

Arch x uArch

• **Architecture**
The collection of features of a processor (or a system) as they are seen by the “user”
  – User: a binary executable running on that processor, or
  – assembly level programmer

• **μArchitecture**
The collection of features or way of implementation of a processor (or a system) that do not affect the user.

• Features which change Timing/performance are considered microarchitecture.
Arch and uArch elements

- **Architecture**
  - Registers data width (8/16/32/64)
  - Instruction set
  - Addressing modes
  - Addressing methods (Segmentation, Paging, etc...)
  - Protection
  - etc...

- **µArchitecture**
  - Bus width
  - Physical memory size
  - Caches & Cache size
  - Number of execution units
  - Execution Pipelines, Number & type execution units
  - Branch prediction
  - TLB
  - etc...

- Timing is considered µArchitecture (*though it is user visible!*
Computer Complexity

• IOMMU (Intel VT-d)
  – What is it?
  – How does it work -> Is it a chip or a software?
  – Where is it located?
Easy, another one...

• How many processors your machine runs? (4? 8? 2 x 4 threads?)

• Well...
  – Your cpu’s (and threads)
  – Your ME (Manageability Engine)
    • INTERRUPT RECEIVED!!
Interrupt Handler

• DAL [1] (Dynamic Application Loader)
  – Runs Java (permits 3rd part applications)
  – ARC Processor
  – Complete real-time OS
    – IRET

Processors...

- **PCU (Power Control Unit)**
  - Pure assembly OS (similar, but not fully compatible x86 ISA)
  - Mainly comprises watchdogs for voltage failures (trying to recover or reset)
  - Harvard Architecture (separate RAM/ROM)

- **PMC (Power Management Controller)**
  - Your PCH PCU

- **GPCU (Graphics PCU)**
Processors...

- Your gigabit card
  - Uops, research already done in that [2]


- VCU (Validation Control Unit)

- BMC (servers)... [3]

Tricky? Let's make it easier then…

• How many execution modes your processor has?
  – Real-­‐mode?
  – Virtual-­‐8086 mode?
  – Protected mode?
  – SMM Mode?

[4] My troopers talk and Phrack article on the subject

POC | | GTFO 0x03
Net Watch: System Management Mode is not just for Governments
Modes...

• Where are the x64 folks?
  – 32e mode?
  – 64-bit mode?
Those are more like ISA-related modes, similar to ARM Thumb I and/or II modes

• More??
  – VT-x anyone?
  – SGX (Software Guard Extensions) anyone?
    • More on this later
Modes...

• So, I’m bul*********?
  – One is new (SGX is not even available yet) and the other is not exactly a mode...

• What about cram mode then??
  – Anyone? Note: CRAM != CAR (Yeap, I love Intel acronyms)
Modes...

- What about TXT? (GETSEC[ENTER]) to execute SINIT ACM -> AMD PRESIDIO?
  - I’ll discuss this later as well ;)
  - ACM (Authenticated Code Modules) run in CRAM mode
  - Memory Areas protected by VT-d PMR or DPR (DMA Protected Region)
  - Any code can be called and execute a MLE (Measured Launch Environment)
  - BTW: Interesting ACM to have a look at: SCLEAN (clean system memory)

- Probe mode anyone?
  - For debugging with hardware debugger (ITP – In-Target Probe)
  - Supports JTAG standard/protocol
  - No regular instruction fetch (IP register not in use)
    - PIR (Probe Mode Instruction Register)
    - PDR (Probe Mode Data Register)
    - Probe-mode special instructions
      » PROBEMODE (to specify each core in PM or NOT)
      » WRSUBPIR (to send instruction to CPU)
      » READPDR (read PDR)
    - MSRs exclusive to probe mode (not real hardware register, only ucode functions)
      » prob_LT_sp_cyc – Access LT private space
      » prob_change_pg_bit – Disable/enable paging in 64 bit mode
  - SMM special relation (Enter Probe mode thru Redirection)
    » ICECTLPMR (redirection configuration bits) – MSR available in probe mode
    » SMM_ENTER, SMM_EXIT, MC_ENTER, INIT, IR (interrupt)
Yeap, life was easy... ;)

- Real-Address Mode
  - Reset or PE=0
  - PE = 1
- Protected Mode
  - VM = 0
  - VM = 1
- Virtual-8086 Mode
- System Management Mode
  - SMI#
  - RSM

Reset
Still missing a few we discussed
Now it is very clear

Six Stages of Debugging

1. That can’t happen.

2. That doesn’t happen on my machine.

3. That shouldn’t happen.

4. Why does that happen?

5. Oh, I see.

6. How did that ever work?
So9ware, software, software

• That gets even better…:
  – How much of your hardware is actually software?
    • The ‘exported’ ISA is not exactly the same behind the wheels
    • What about uCode patches? (who here don’t love cpu erratas?)
    • Your ISA is translated to uCode, which has uInstructions that depend on the uArch (and parts of the process are bare metal)
So, there are any backdoors?

- Fair response (which I once heard but don’t remember the author to give the proper credits):
  
  - None that I’m aware of (in my case I actually read the entire mov implementation)... but if you ask me again next Troopers (I’m sure you will not want to miss the next one after coming to this one, do you?) and I say I prefer to not comment, you might take your own conclusions ;)


The feeling about security?
TCB (Trusted Computing Base)

• I don’t think considering vendor backdoors in Hardware by the big players should be in any TCB since:
  – There are other (easier) ways to get access without involving a complex and difficult to hide and change hardware component
    • Logistics interception
    • Exploiting vulnerabilities (at this point I believe everyone agree that giving the complexity, the number of issues published is *SUPER* small – yeap, the teams at the companies are doing a good job, but still)
What you can do to improve?

• Work on validating your systems using FREE and WIDELY available technologies:
  – TXT to get an image of what you have and compare with other companies that use different supply chains and are in different countries (this protects you against supply chain hijacks)
  – Intel has a software called MtWilson that helps you do that (it also integrates with VM-based environments to collect data on kernel components of it) -> Honestly, I don’t even know what (if any) is the cost of it, but there are an open-source version (also made by Intel) [5]

Suggestion for a Project

• Maybe a nice project to have is to share integrity values around the world, like nowadays people starting to share IOCs?
  – Different BIOSes integrity values
  – Different VMMs

• This helps everyone finding already compromised systems, force the manufactures to publish such information and avoids supply chains hijacks ;)

And what about the future?

- Pointer Lookout
  - SDE (Software Development Emulator) [6]
  - Patch already available on GCC


- SGX (Software Guard Extensions) [7] [8]
  - Secure Enclaves

[8] https://sites.google.com/site/haspworkshop2013/workshop-program
SGX and what it tries to solve

... and apps from each other ...

... UNTIL a malicious attacker gains full privileges and then tampers with the OS or other apps

Apps not protected from privileged code attacks
SGX details

- Very different architecture (compared to ARM’s TrustZone):
  - A Memory Encryption Engine (MEE) exists to encrypt memory data
  - This protects a given memory region from hardware-based access
  - Even with Intel privileged access, the keys are not exposed (they are erased if an unlock happens)
  - OS is part of the TCB and thus seem as a malicious element
SGX

Protected execution environment embedded in a process.

With its own code and data
Provide Confidentiality
Provide Integrity
Controlled Entrypoints
Supports multi-threading
With full access to app memory
SGX – Attack Surface

- Attack Surface today
- Attack Surface with SE
How SE Works: Protection vs. Software Attack

Application

1. App is built with trusted and untrusted parts
2. App runs & creates enclave which is placed in trusted memory
3. Trusted function is called; code running inside enclave sees data in clear; external access to data is denied
How SE Works: Protection vs. Software Attack

Application

1. App is built with trusted and untrusted parts
2. App runs & creates enclave which is placed in trusted memory
3. Trusted function is called; code running inside enclave sees data in clear; external access to data is denied
4. Function returns; enclave data remains in trusted memory
1. Security perimeter is the CPU package boundary
2. Data and code unencrypted inside CPU package
3. Data and code outside CPU package is encrypted and integrity checked with replay protection
4. External memory reads and bus snoops see only encrypted data
5. Attempts to modify memory will be detected
Life Cycle of An Enclave

1. Application is launched by OS
2. Application calls SE driver to allocate enclave
3. Driver calls **ECREATE** to allocate SECS
4. Application calls SE driver to add enclave pages to EPC
5. Driver calls **EADD** to add pages to EPC
6. Driver calls **EEXTEND** to extend measurement with initial contents of pages
7. Application calls SE driver to initialize enclave, providing SIGSTRUCT and LICTOKEN
8. Driver calls **EINIT** with SIGSTRUCT and LICTOKEN
9. Application enters enclave with **EENTER**
10. Enclave returns control to the application with **EEXIT**
11. Upon application exit, driver reclaims EPC pages with **EREMOVE**
Example: Secure Transaction

1. Enclave built & measured against ISV’s signed manifest
2. Enclave requests REPORT from HW
3. REPORT & ephemeral key sent to server & verified
4. A trusted channel is established with remote server
Example: Secure Transaction

1. Enclave built & measured against ISV’s signed manifest
2. Enclave requests REPORT from HW
3. REPORT & ephemeral key sent to server & verified
4. A trusted channel is established with remote server
5. A trusted channel is similarly established with secure input via the PSW enclave
6. User info sent securely to server
7. Application Key provisioned to enclave
1. Enclave built & measured against ISV’s signed manifest
2. Enclave requests REPORT from HW
3. REPORT & ephemeral key sent to server & verified
4. A trusted channel is established with remote server
5. A trusted channel is similarly established with secure input via the PSW enclave
6. User info sent securely to server
7. Application Key provisioned to enclave
8. Enclave requests the HW Enclave-platform Sealing Key
9. Application Key encrypted using Sealing Key & stored for successive sessions
SGX

- EPC: Decrypted code and data space with
  - SE access control
  - Located in system memory space
  - Implementation as stolen main memory (protected by MEE)
  - Protected from SW by range registers
  - Protected from HW by encryption and integrity protection
- EPCM: Provide meta data for each EPC page
- SE1 Instructions: 12 new operations
  - ECREATE, EADD, EEXTEND, EINIT, EENTER, EIRET, EEXIT, EREMOVE, EDBGDRD, EDBGWR, EREPORT, EGETKEY
- 2 Opcodes: 1 privileged; 1 unprivileged
<table>
<thead>
<tr>
<th>EAX</th>
<th>Leaf Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>ECREATE</td>
<td>Sets up the initial state environment, sets up the measurement, adds SECS page to EPC, measures attributes and xsave mask</td>
</tr>
<tr>
<td>0x1</td>
<td>EADD</td>
<td>Adds a page to the enclave. Page can be used as enclave control, application data, code, stack, or heap</td>
</tr>
<tr>
<td>0x2</td>
<td>EINIT</td>
<td>Verifies permit and marks the enclave ready to run</td>
</tr>
<tr>
<td>0x3</td>
<td>EREMOVE</td>
<td>Removes pages from an enclave</td>
</tr>
<tr>
<td>0x4</td>
<td>EDBGRD</td>
<td>Reads 8 bytes from a debug enclave</td>
</tr>
<tr>
<td>0x5</td>
<td>EDBGWR</td>
<td>Writes 8 bytes to a debug enclave</td>
</tr>
<tr>
<td>0x6</td>
<td>EEXTEND</td>
<td>Extends the measurement of the enclave with a measurement of an additional chunk of memory</td>
</tr>
</tbody>
</table>
# SGX

## Ring 3 Instructions

<table>
<thead>
<tr>
<th>EAX</th>
<th>Leaf Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>EREPORT</td>
<td>Demonstrates cryptographically what was placed inside the enclave</td>
</tr>
<tr>
<td>0x1</td>
<td>EGETKEY</td>
<td>Provides access to system secrets &amp; user level keys</td>
</tr>
<tr>
<td>0x2</td>
<td>EENTER</td>
<td>Transfers control to a predefined entry point within the enclave</td>
</tr>
<tr>
<td>0x3</td>
<td>EIRET</td>
<td>Resume execution from its interrupt / exception point</td>
</tr>
<tr>
<td>0x4</td>
<td>EEXIT</td>
<td>Returns from the enclave to the application</td>
</tr>
</tbody>
</table>
BIOS, ACPI, others

- Yes, huge code base!
- Lots of OEM capabilities
- Already been exploited by three letter agencies
  - Chipsec: [https://github.com/chipsec/chipsec](https://github.com/chipsec/chipsec)
  - Recently announced in CanSecWest (past week?)
ISA Extensions?

- AES-NI
- AVX (SIMD)

- Nice to discover huge NOPs in older systems:
  `66 66 0F 1F 84 00 00 00 00 00 00`
Conclusions

• Folks, I could just keep going but I believe I made my points clear, but just to reinforce them:
  – Your computer is actually a network of computers, running lots of software (Sergey Bratus and Langsec here at Troopers!)
  – Modern architecture provides a way for you to generate a hardware attestation (TXT) and this is your way to protect against supply chain based attacks (sorry NSA, but I’m Brazilian)
  – We need efforts on the community to use features that are already present and that solve many platform-related problems:
    • Central base for sharing known good attestations?
    • Software to manage and gather attestation information in networks, comparing to known good states and previously saved ones (OpenAttestation?)
Thank You

Rodrigo Rubira Branco (BSDaemon)
rodrigo *noSPAM* kernelhacking.com
https://twitter.com/bsdaemon

• BSDaemon