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

- 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)

Start answering questions

- 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 ;)

[0] http://www.insinuator.net/2014/03/how-to-use-intel-amt-and-have-some-fun-with-mainboards/

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

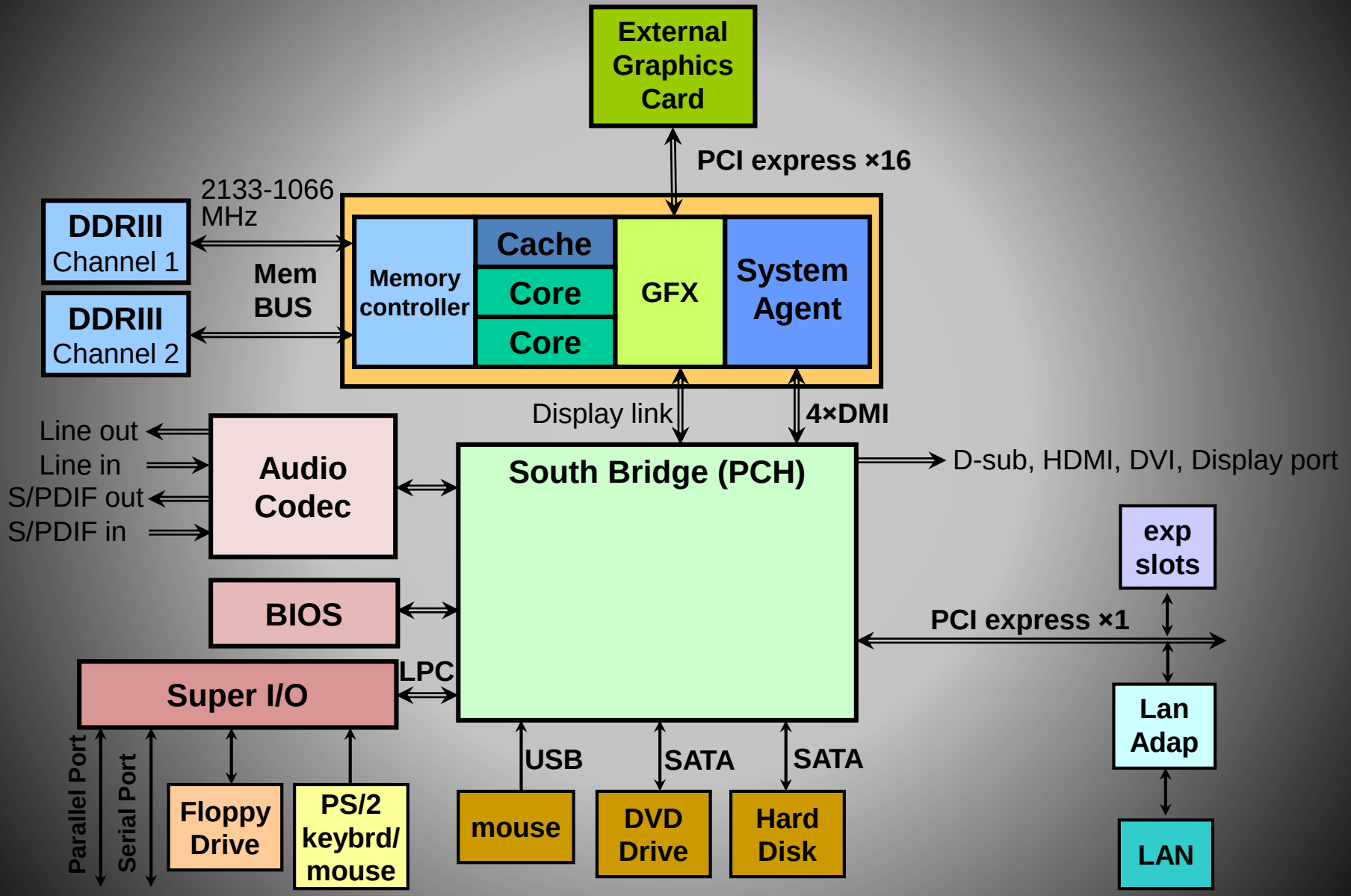
- **µArchitecture**
  The collection of features or way of implementation of a processor (or a system) that do not affect the user.

- 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!)*

Your computer



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

[1] http://developers.txe.iil.intel.com/discover

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]

[2] http://www.alchemistowl.org/arrigo/Papers/Arrigo-Triulzi-PACSEC08-Project-Maux-II.pdf

- VCU (Validation Control Unit)

- BMC (servers)… [3]

[3] https://community.rapid7.com/community/metasploit/blog/2013/07/02/a-penetration-testers-guide-to-ipmi

Tricky? Lets make it easier then…

- How many execution modes your processor has?
  - Re
  - Vir
  - Pr
  - SM

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

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

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…

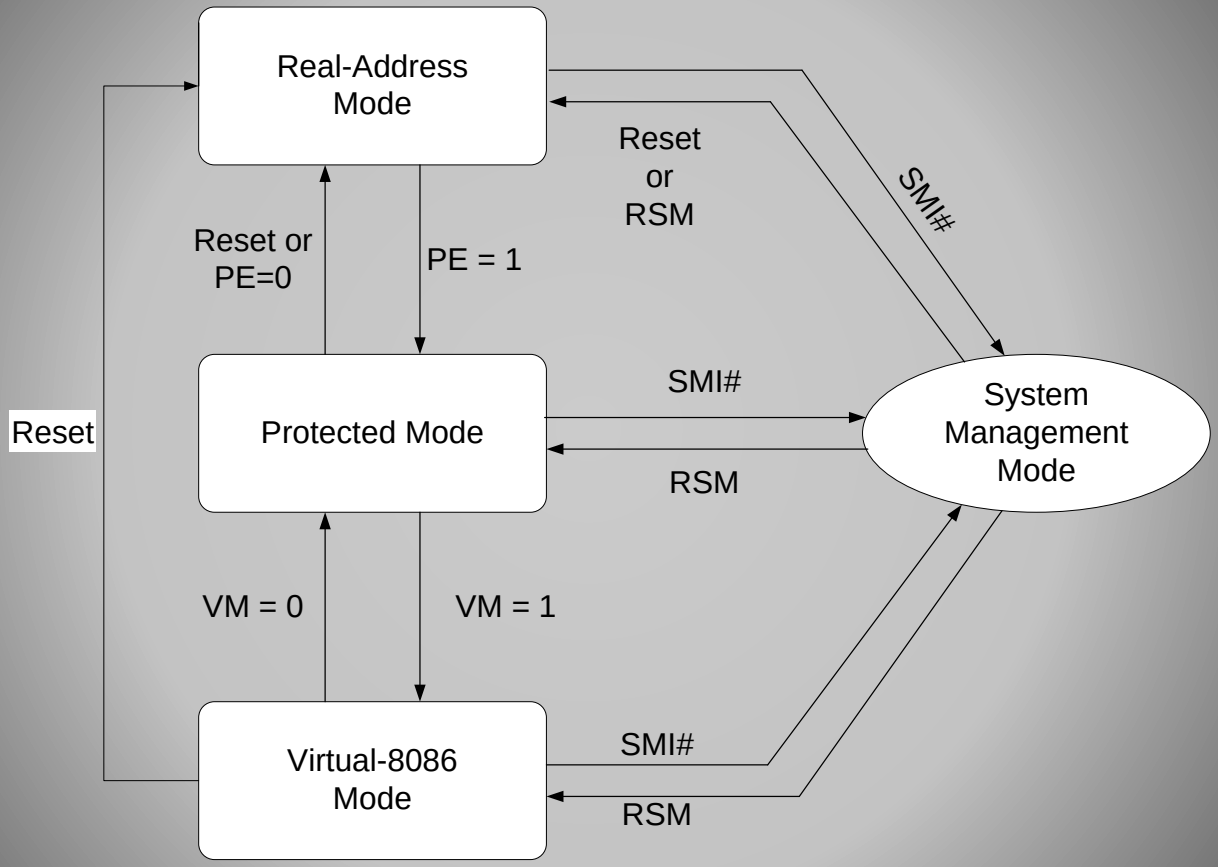
- So, I'm bul*********?
  - One is new (SGX is not even available yet) and the other is not exactly a mode…
- What about cram mode then??
  - Anyone? Note: CRAM != CAR (Yeap, I love Intel acronyms)

Modes…

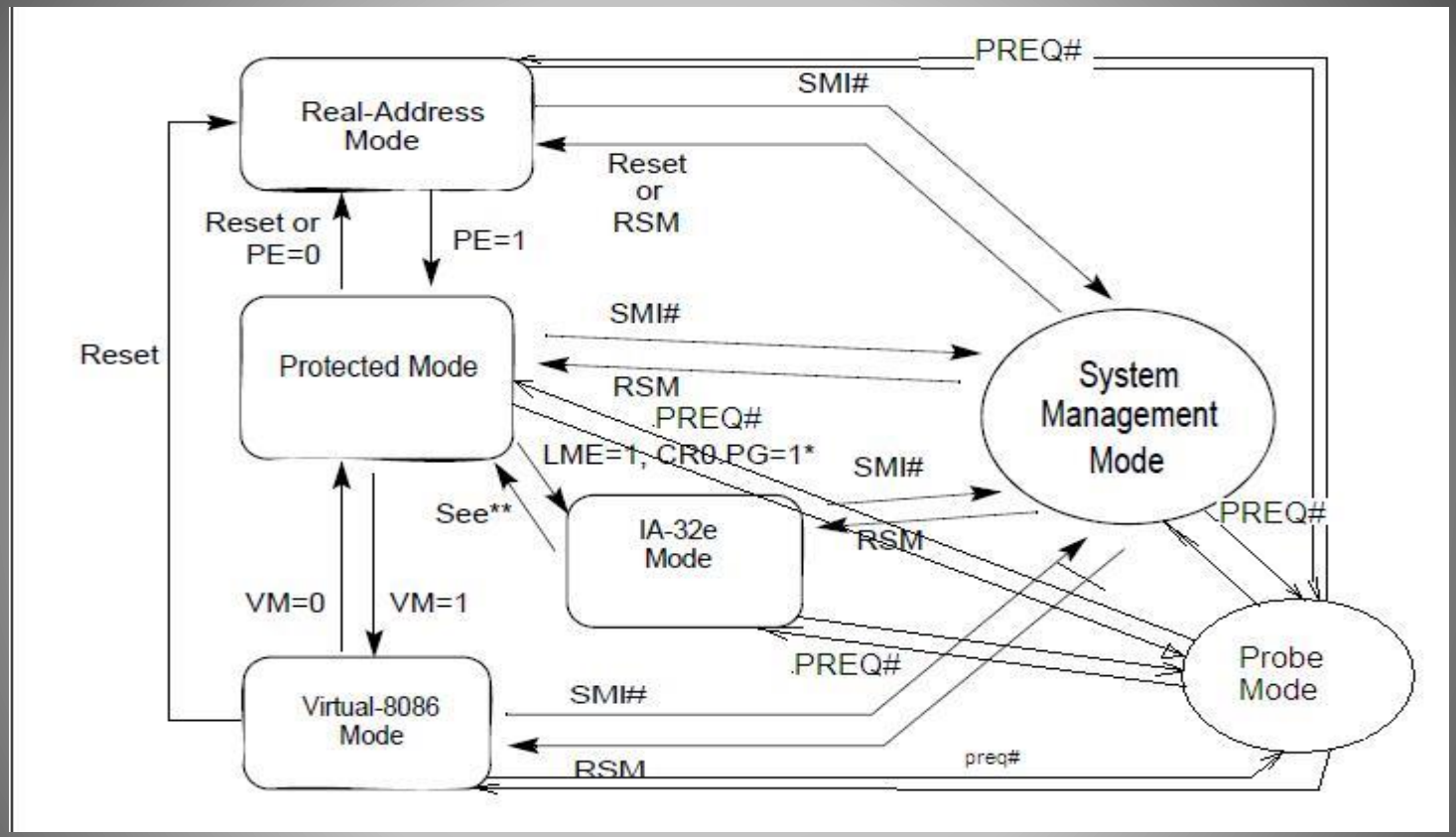
- What about TXT? (GETSEC[SENTER]) to execute SINIT ACM -> AMD PRESIDIO?
  - I'll discuss this later as well ;)
  - ACM (Authenticated Code Modules) run in CRAM mode
  - Memory Areas protected by VT-d PMR or DPR (DMA Protected Region)
  - Any code can be called and execute a MLE (Measured Launch Environment)
  - 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)
      - » READPDRT (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… ;)



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?

Software, 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]

[5] https://github.com/OpenAttestation/

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

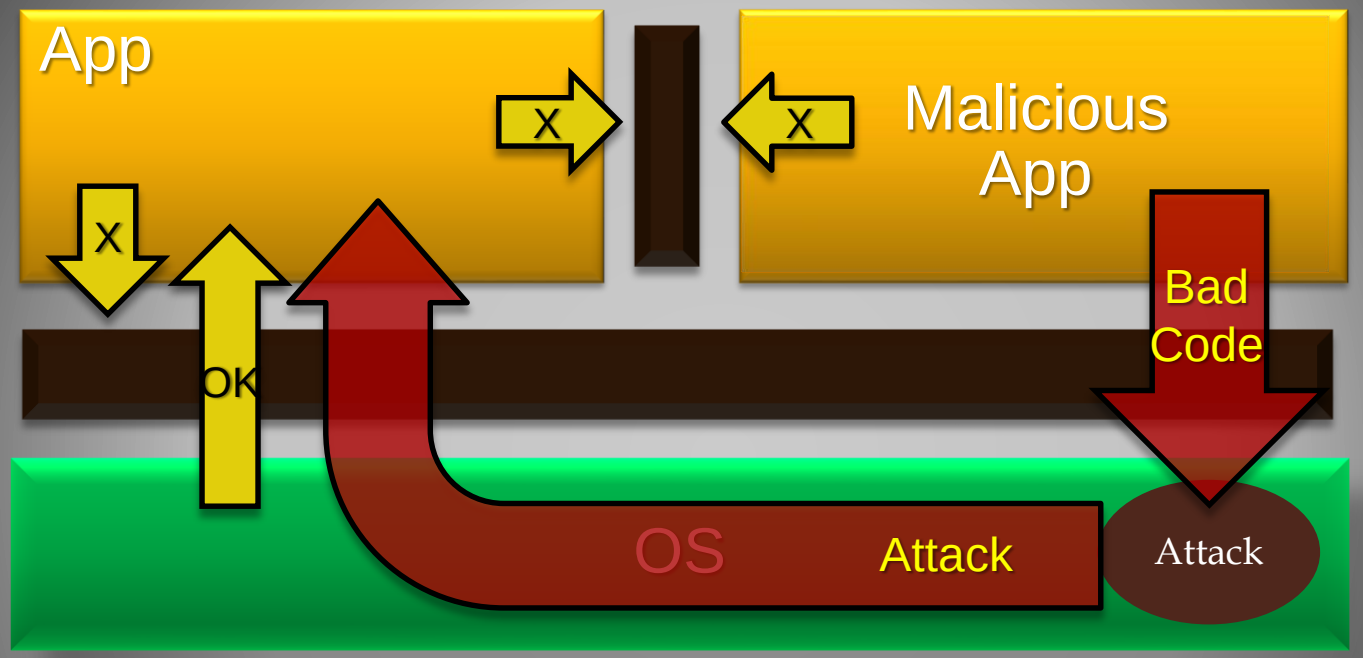
[6] http://software.intel.com/en-us/articles/intel-software-development-emulator

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

[7] http://software.intel.com/en-us/blogs/2013/09/26/protecting-application-secrets-with-intel-sgx

[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

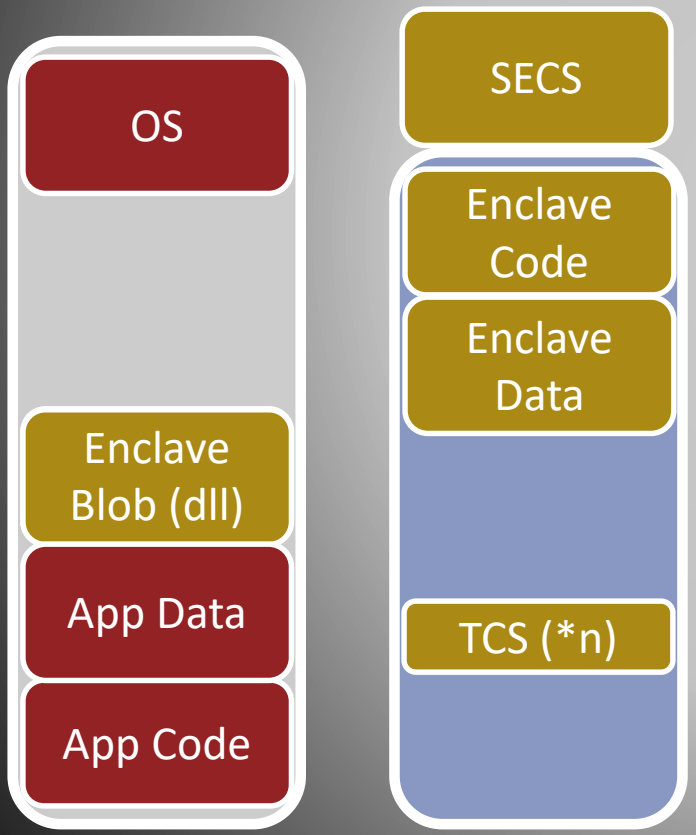
Apps not protected from privileged code attacks

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

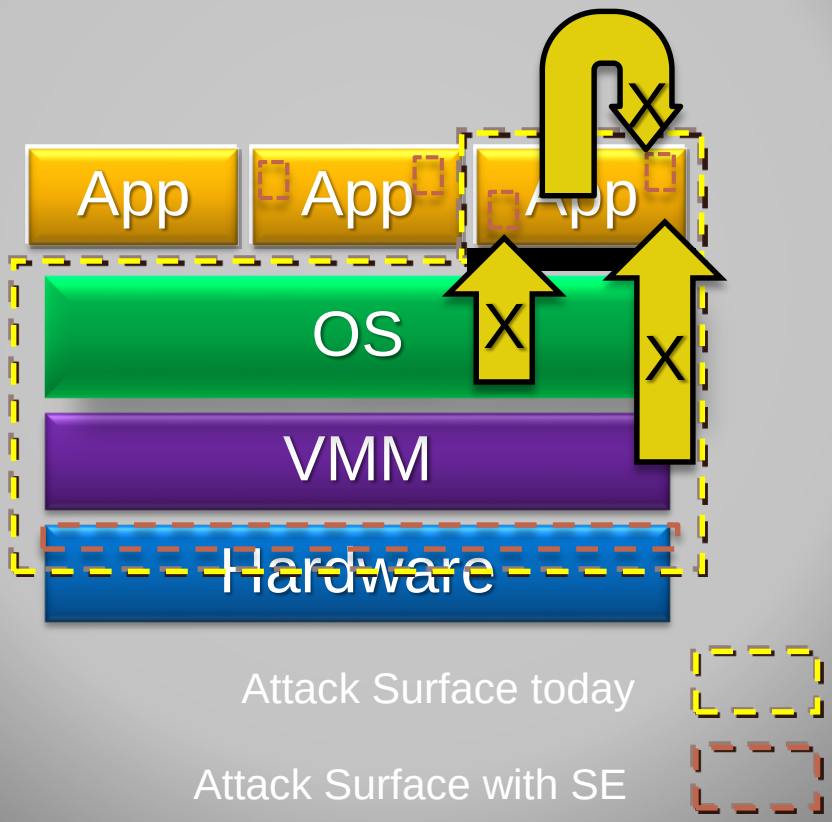
Provide Integrity

Controlled Entrypoints

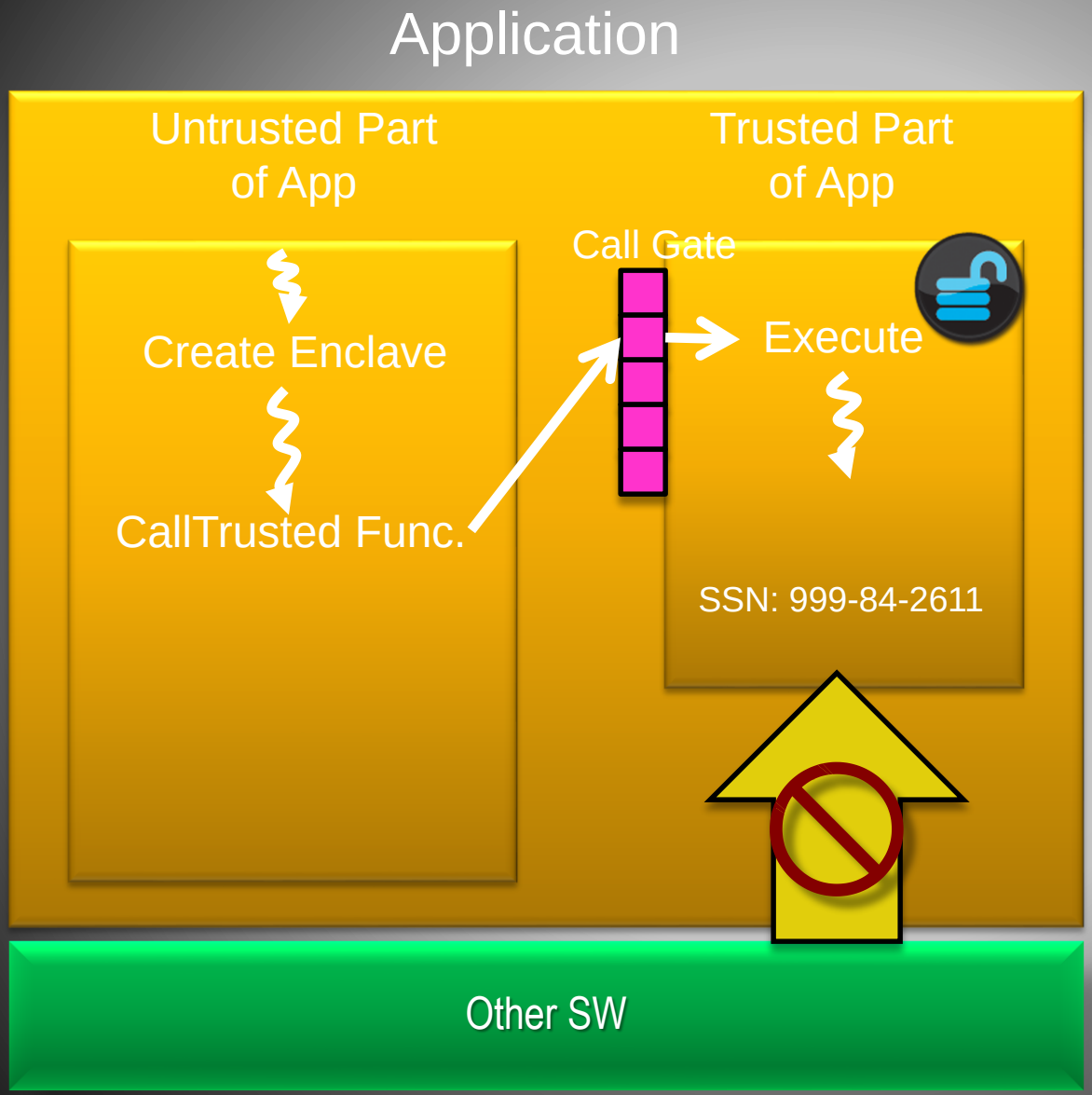
Supports multi-threading

With full access to app memory

SGX – Attack Surface

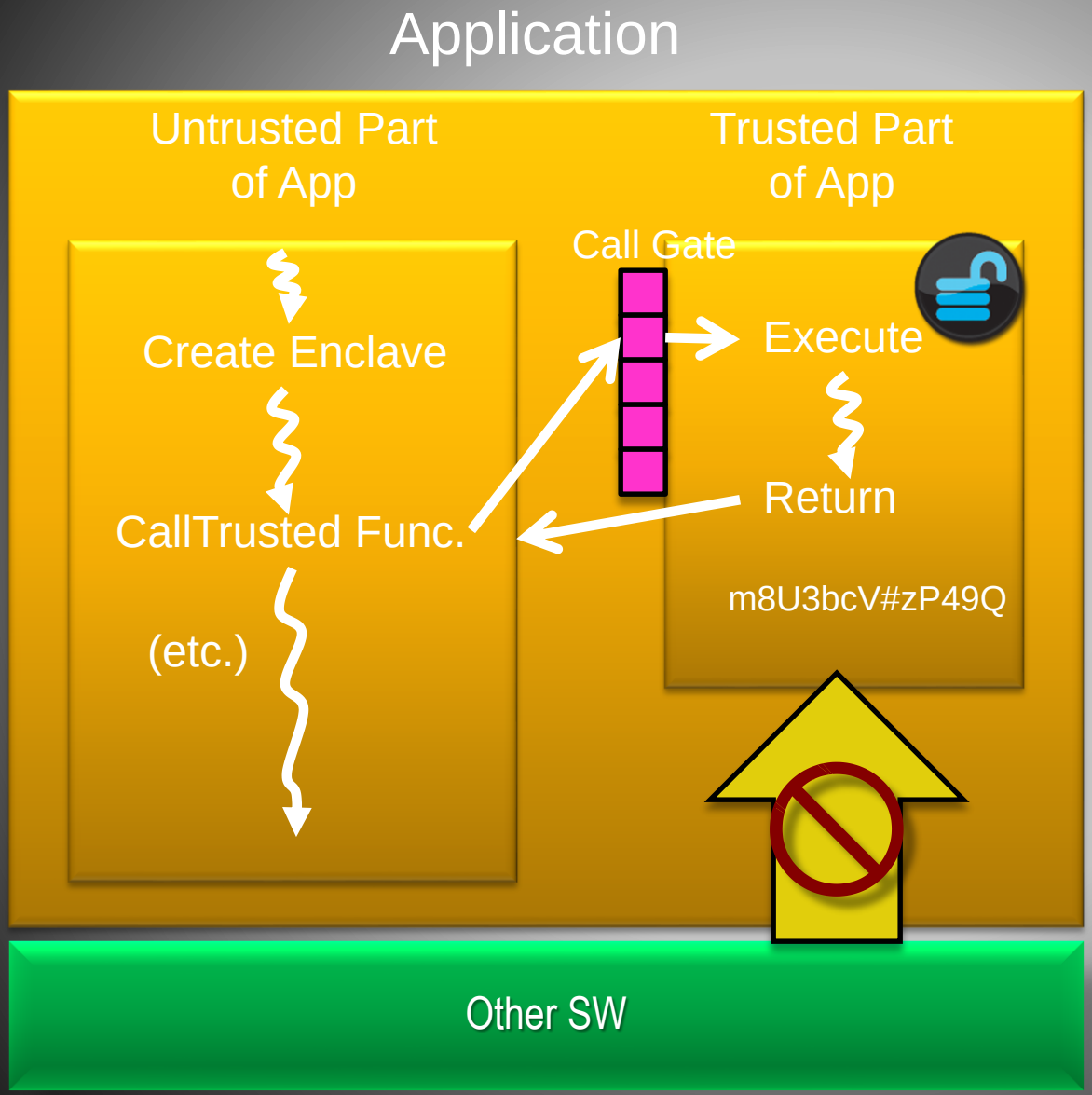


How SE Works: Protection vs. Software Attack



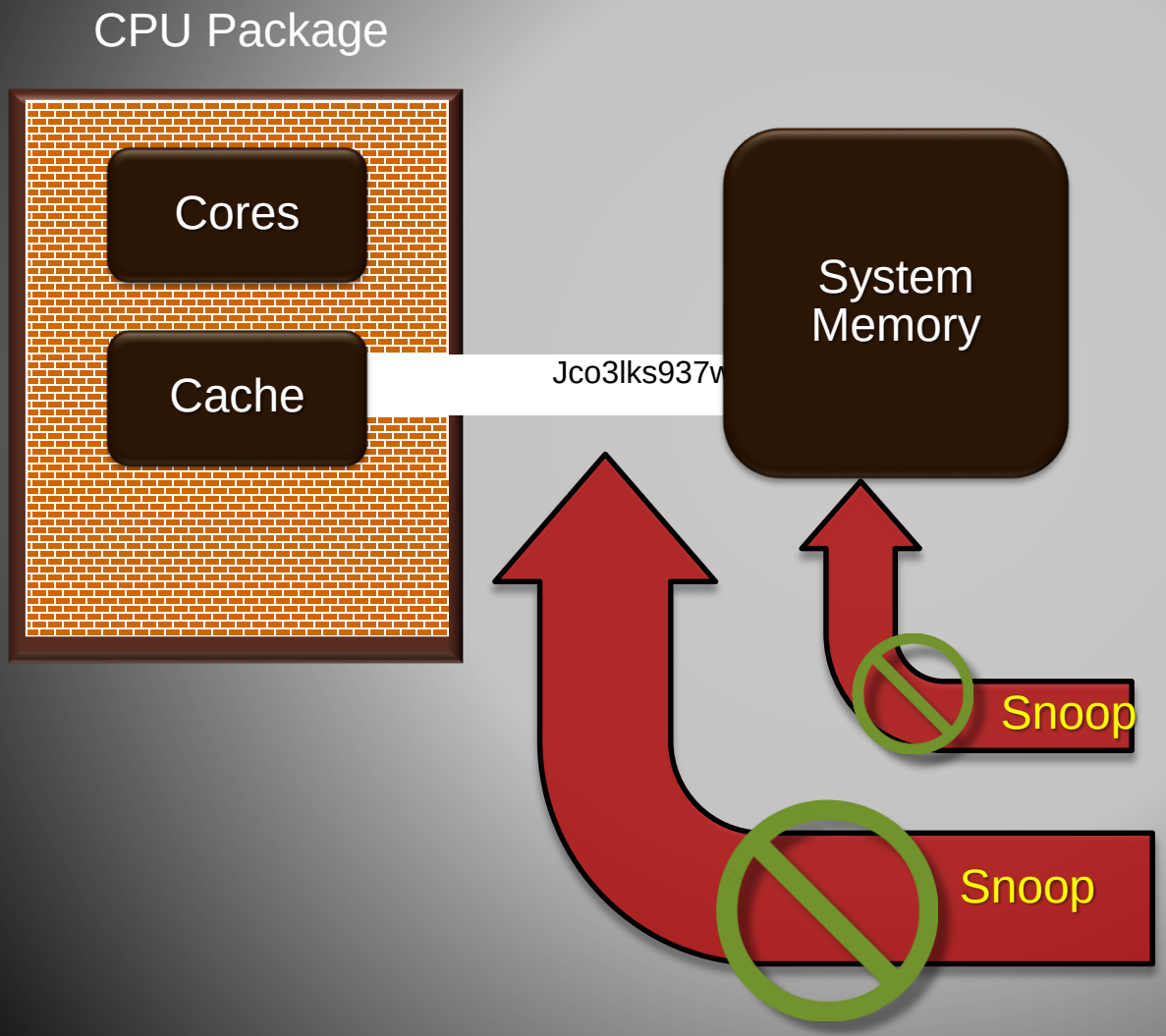
1. App is built with trusted and untrusted parts
2. App runs & creates enclave which is placed in trusted memory
3. Trusted function is called; code running inside enclave sees data in clear; external access to data is denied

How SE Works: Protection vs. Software Attack



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

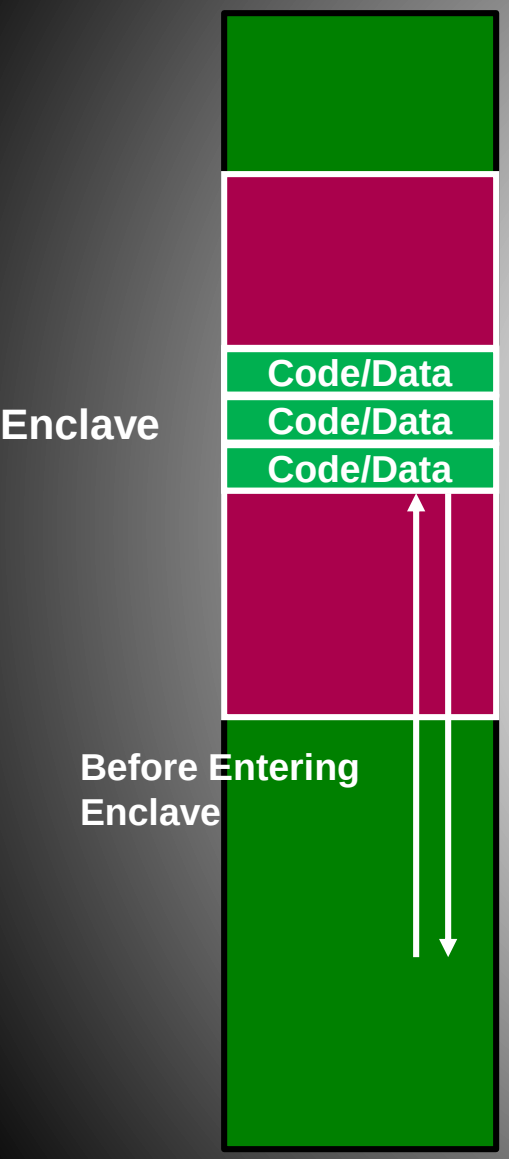
SGX - MEE



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

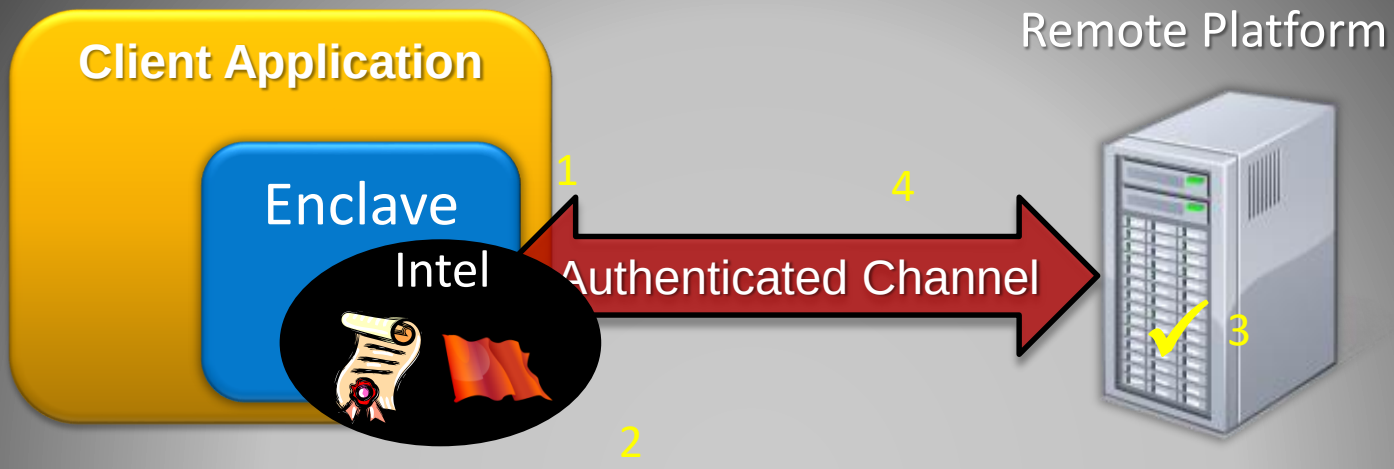
Build



ECREATE (Range)
EADD (Copy Page)
EEXTEND
EINIT
EENTER
EEXIT
EREMOVE

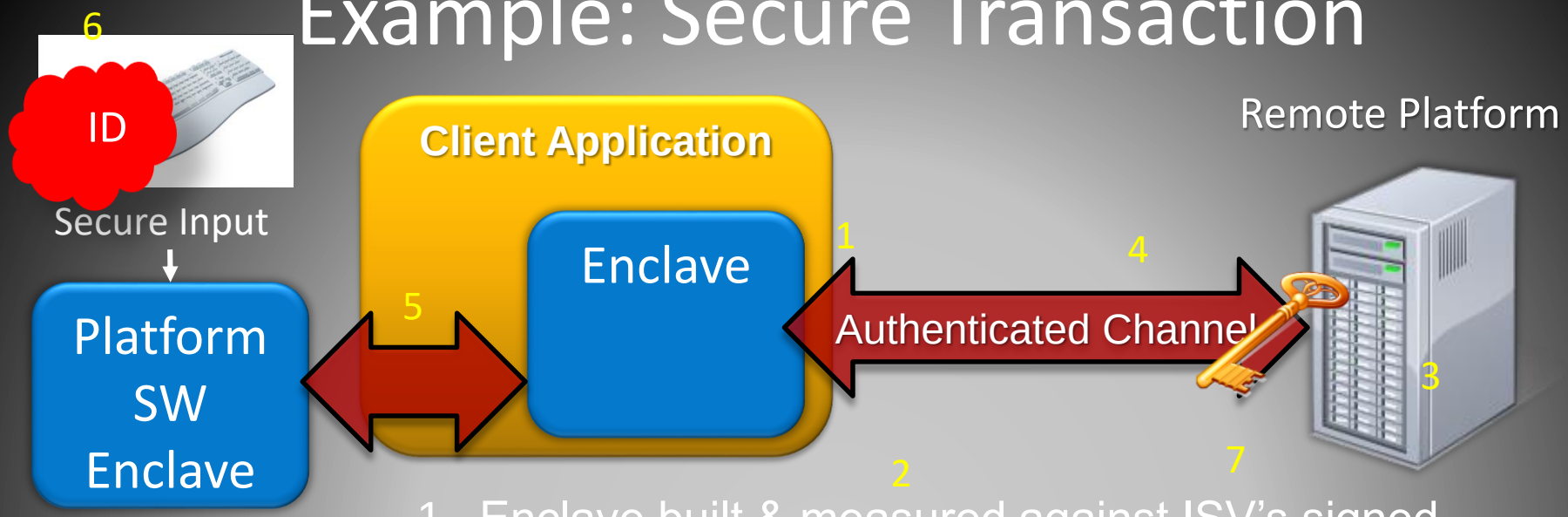
1. Application is launched by OS
2. Application calls SE driver to allocate enclave
3. Driver calls **ECREATE** to allocate SECS
4. Application calls SE driver to add enclave pages to EPC
5. Driver calls **EADD** to add pages to EPC
6. Driver calls **EEXTEND** to extend measurement with initial contents 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

Example: Secure Transaction

Secure Input

Platform SW Enclave

Client Application

Enclave

8

5

7

9

1

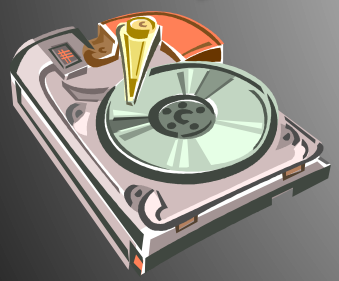
4

Authenticated Channel

Remote Platform

3

2



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, EDBGRD, EDBGWR, EREPORT, EGETKEY
- 2 Opcodes: 1 privileged; 1 unprivileged

SGX
Ring 0 Instructions

EAX	Leaf Function	Description
0x0	ECREATE	Sets up the initial state environment, sets up the measurement, adds SECS page to EPC, measures attributes and xsave mask
0x1	EADD	Adds a page to the enclave. Page can be used as enclave control, application data, code, stack, or heap
0x2	EINIT	Verifies permit and marks the enclave ready to run
0x3	EREMOVE	Removes pages from an enclave
0x4	EDBGRD	Reads 8 bytes from a debug enclave
0x5	EDBGWR	Writes 8 bytes to a debug enclave
0x6	EEXTEND	Extends the measurement of the enclave with a measurement of an additional chunk of memory

SGX
Ring 3 Instructions

EAX	Leaf Function	Description
0x0	EREPORT	Demonstrates cryptographically what was placed inside the enclave
0x1	EGETKEY	Provides access to system secrets & user level keys
0x2	EENTER	Transfers control to a predefined entry point within the enclave
0x3	EIRET	Resume execution from its interrupt / exception point
0x4	EEXIT	Returns from the enclave to the application

BIOS, ACPI, others

- Yes, huge code base!
- Lots of OEM capabilities
- Already been exploited by three letter agencies
  - 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

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