Firmware is the new Black – Analyzing Past 3 years of BIOS/UEFI Security Vulnerabilities

Bruce Monroe & Rodrigo Rubira Branco (@bsdaemon) & Vincent Zimmer (@vincentzimmer)

{ bruce.monroe || rodrigo.branco || vincent.zimmer } @ intel.com

DISCLAIMER

- Intel has official documentation that is highly comprehensive and should be used to make technical decisions related to Intel's technologies. To have a high-level of accuracy for such documentation, lots of reviews are performed

- The accuracy of this talk can't be compared and should not be used to compare with official documentation. We are going to discuss directions, strategies and initiatives being proposed, giving recommendations to OEMs, researchers and customers but everything should be treated as *OUR* opinion instead of official statements

- There possibly are other initiatives and focus areas that we are not at liberty to talk about or that we are even unaware of, so this should not be considered the full scope of the problem, but instead, *OUR* vision of it

- This talk is about BIOS/UEFI security. We are not pedantic on terminology (since we accept that the majority outside of Intel have a harder time to know/differentiate what is the responsibility of different components of the platform in order to guarantee the security of it). As so, we did include things that are *NOT* directly related to errors in BIOS/UEFI per-se, but that somehow affected the expectations for it

Agenda

- BIOS/UEFI background
- BIOS/UEFI ecosystem
- BIOS/UEFI security technologies
- Dataset & Methodology
- Bug classes (a proposal for UEFI)
- Vulnerability distribution (by bug classes)
- Platform Firmware Threat Modelling
- Future

UEFI Background

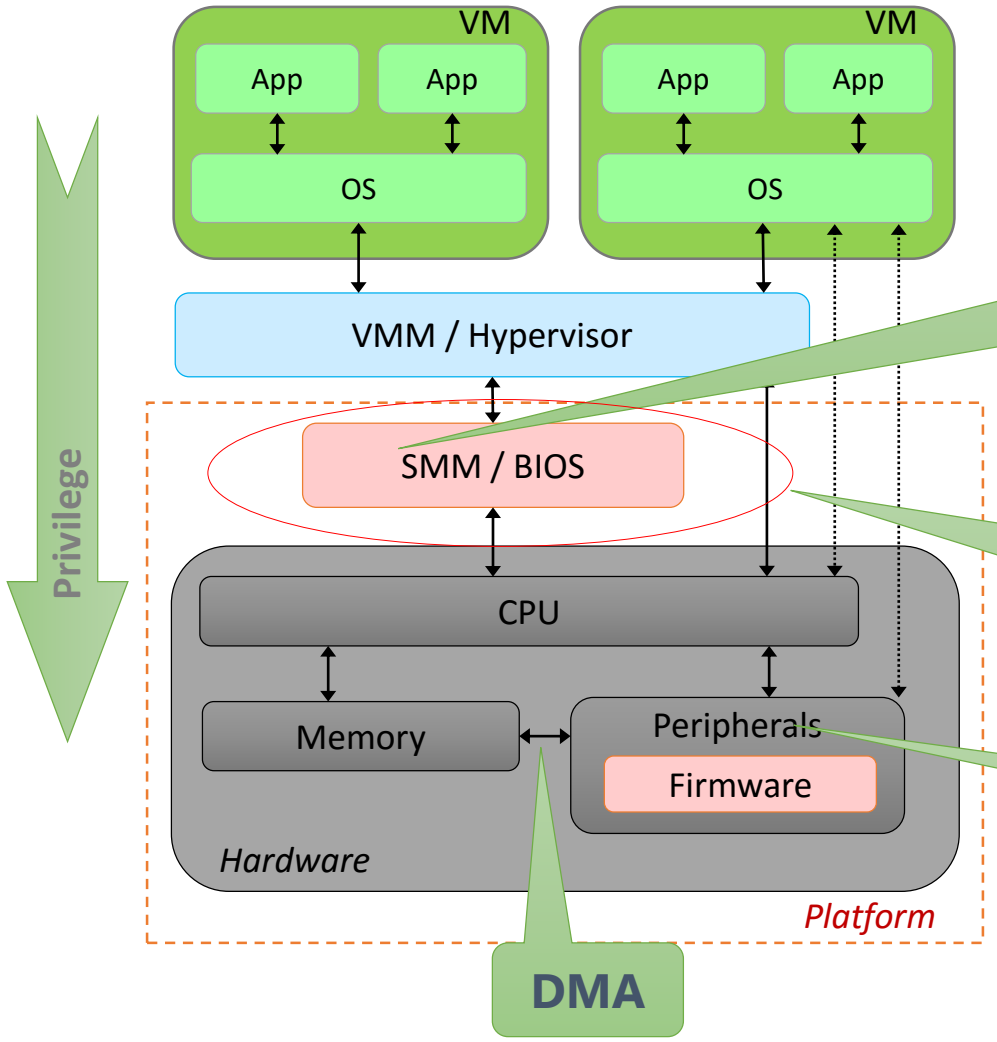
UEFI – Unified Extensible Firmware Interface old school terms…(BIOS)

- First up root of trust on the system
- It hands over control to the operating system
  - Rest of the magic then occurs ;)
- UEFI Membership spans the compute spectrum

http://uefi.org/members

UEFI – sets up the platform to run…

Where is UEFI firmware

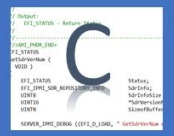


BIOS & OS/VMM share access, but not trust UEFI + PI SMM

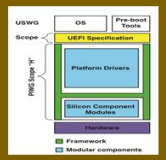
Hypervisor can grant VM direct hardware access

A specific Peripheral may have its own processor, and its own firmware, which is undetectable by host CPU/OS.

What's in UEFI



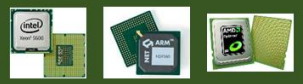
Mostly written in C.
High code re-use.



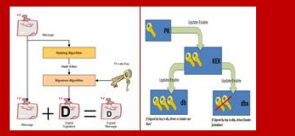
Emphasis on Specifications.
Standards compliance.

Better platform scaling. For e.g. removes shadow ROM limits.

Storage.
GPT removes 2.2 TB MBR restriction.

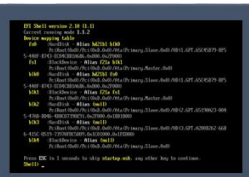


CPU Architecture independent. Platform design flexibility.



Secure boot solves "trust" related system integration challenges.

Pre-boot Networking.
Ipv4, Ipv6, PXE,
VLAN, iSCSI etc.

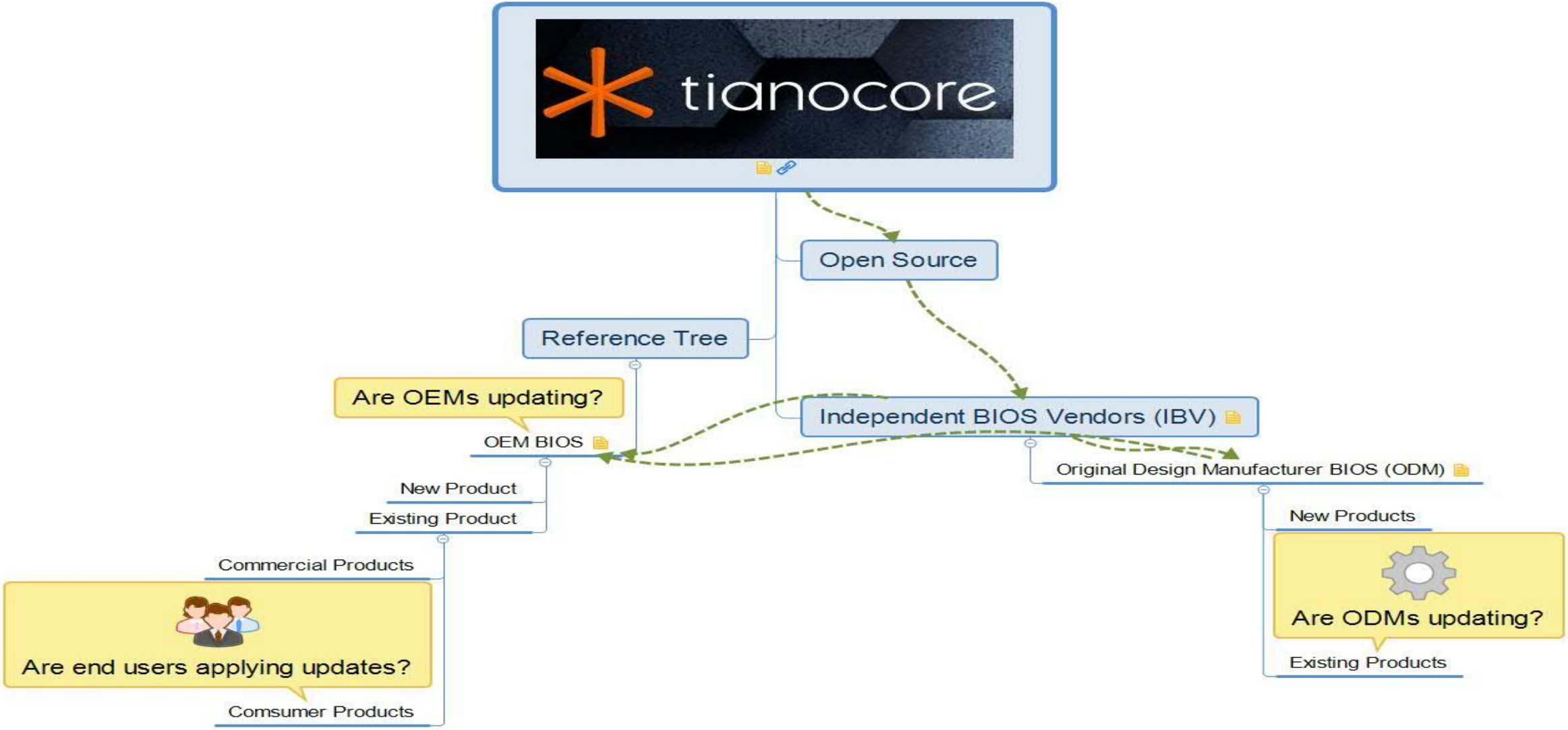


UEFI shell improves pre-boot testing & diagnostics experience.

UEFI Ecosystem Overview

- UEFI Ecosystem is an “Onion”. Layers upon layers…
- Peel the onion and you have:
  - Tianocore (Open Source EDK II) Intel is the sole maintainer
  - IBV – Independent BIOS Vendors
  - OEM – Other Equipment Manufacturers (The folks building your systems)
  - ODM – Original Design Manufacturer
  - Consumers (deliberate action to download/install updates)
- If a vuln mitigation goes out it has to navigate the onion
  - **Additional update lag time is introduced because end users have to take deliberate action to download/install updates**

UEFI Ecosystem



History

UEFI.org is a standards body

- UEFI is a specification
- About 2 years ago UEFI.org stood up USRT
  - USRT – is Unified Security Response Team

Sea change. UEFI had never taken a stance on implementations.

They Are Now…

UEFI USRT

- USRT is comprised by Firmware Engineers from member companies
- To report a security issue in UEFI Firmware implementation from a vendor:
  - Send email to security@uefi.org
  - Encrypt sensitive info with their PGP public key - security@uefi.org
- Please provide as much information as possible, including:
  - The products and versions affected
  - Detailed description of the vulnerability
  - Steps to demonstrate the vulnerability or reproduce the exploit, including specific configurations or peripherals, if relevant
  - Potential impact of the vulnerability, when exploited
  - Information on known exploits

Reporting to Intel Bug Bounty

Tianocore EDK II that is solely maintained by Intel is in bug bounty scope

- Send vulnerability report to secure@intel.com
  - Encrypt using PGP Public Key of secure@intel.com
- Intel Bug Bounty Program is **Private Invite Only**
- If you find a good quality bug in Tianocore we will extend an invite providing you agree to abide by Intel Bug Bounty program terms
  - **Note: you need to be invited to be eligible**
- You can find more information on the Intel Bug bounty program online
- Full participation guidelines https://security-center.intel.com/docs/BugBountyParticipationGuidelines.pdf

Intel Bug Bounty Recap

The Intel Bug Bounty Program is a Private Invite Only program

- If the security vulnerability is in open source implementation of Tianocore solely maintained by Intel
  - It's Eligible for the Intel Bug Bounty Program administered by HackerOne
- You must be invited to join via HackerOne platform

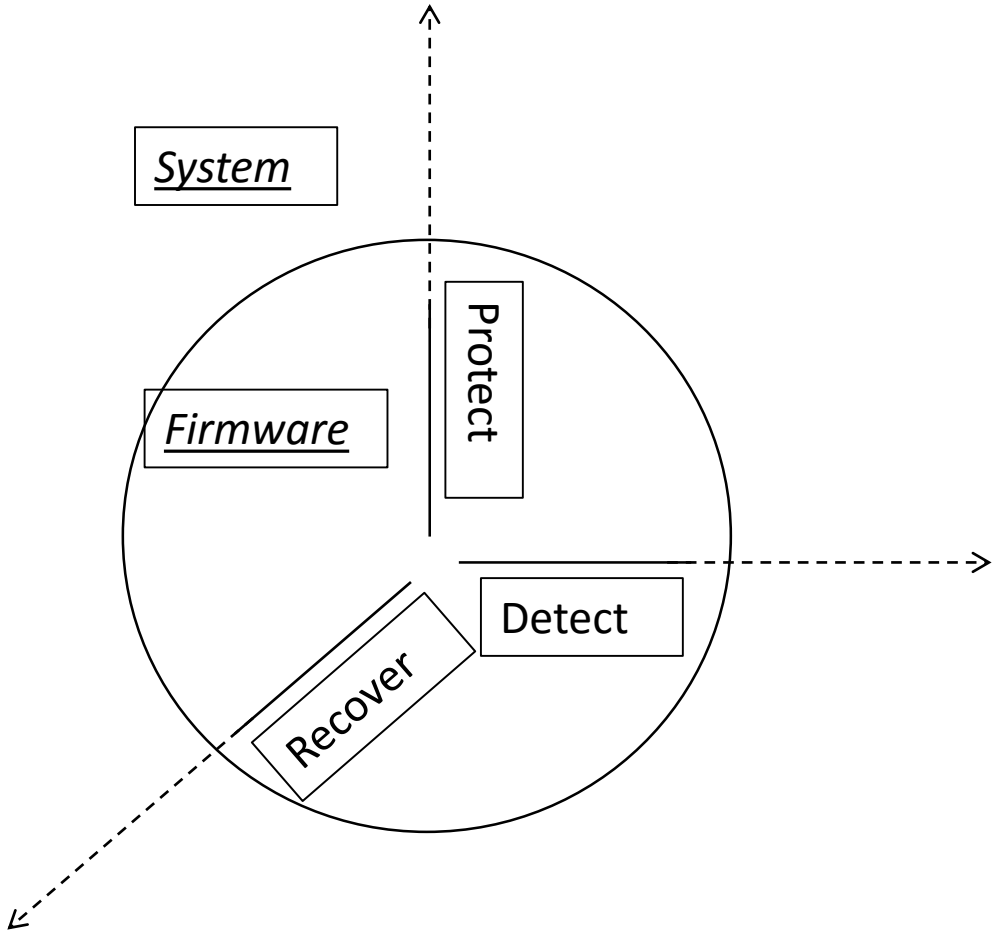
Vulnerability Severity	Intel Software	Intel Firmware	Intel Hardware
Critical	Up to $7,500	Up to $10,000	Up to $30,000
High	Up to $2,500	Up to $5,000	Up to $10,000
Medium	Up to $1,000	Up to $1,500	Up to $2,000
Low	Up to $500	Up to $500	Up to $1,000

EDKII Bugzilla

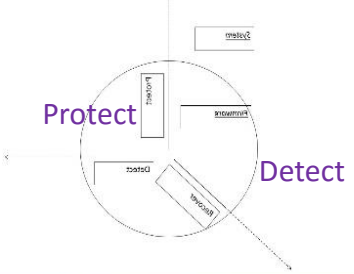
- The EFI Developer Kit II (EDKII) provides an open source implementation http://www.tianocore.org/
  - Core features https://github.com/tianocore/edk2
  - Platform examples https://github.com/tianocore/edk2-platforms
- Reporting security issues on open source https://github.com/tianocore/tianocore.github.io/wiki/Reporting-Security-Issues
  - Advisories https://www.gitbook.com/book/edk2-docs/security-advisory/details

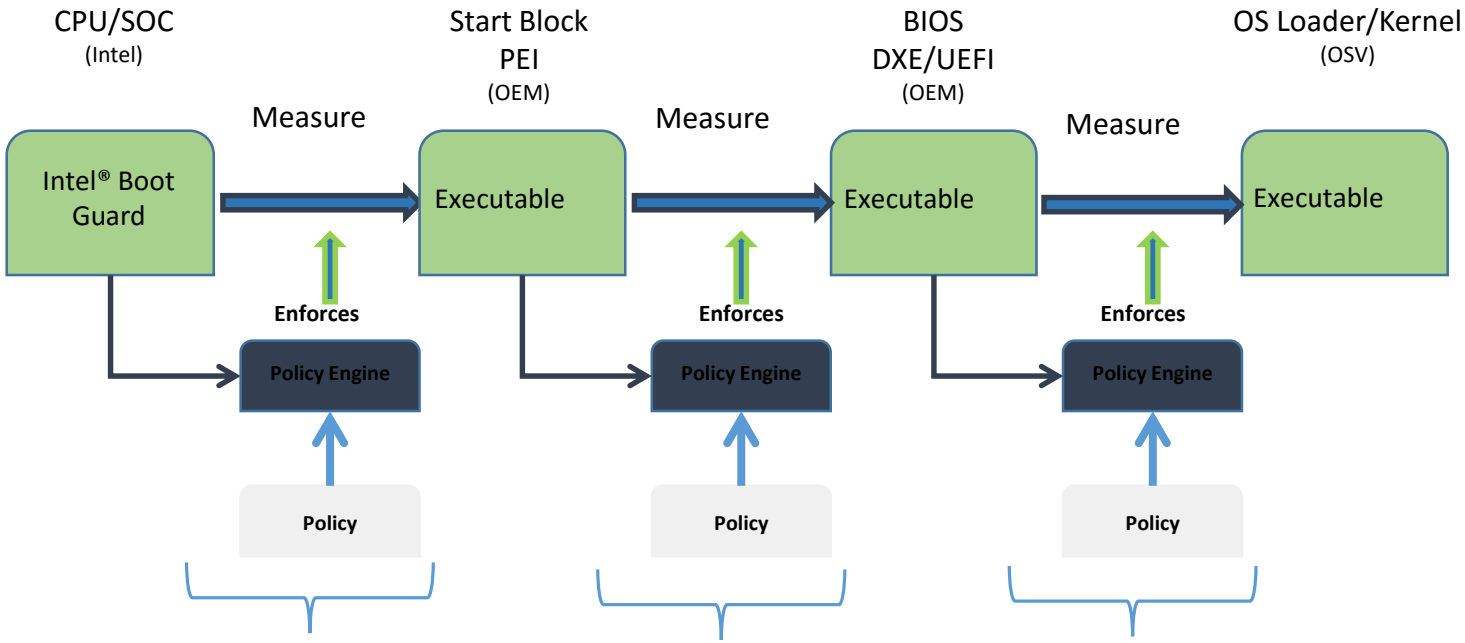
Use of EDKII Defenses

- UEFI is the specification
- EDK II is the code



Fully Verified Boot Sequence





Intel® Device Protection Technology with Boot Guard

http://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/4th-gen-core-family-mobile-brief.pdf

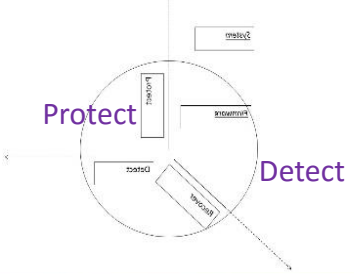
OEM PI Verification Using PI Signed Firmware Volumes

Vol 3, section 3.2.1.1 of PI 1.3 Specification

OEM UEFI 2.7 Secure Boot

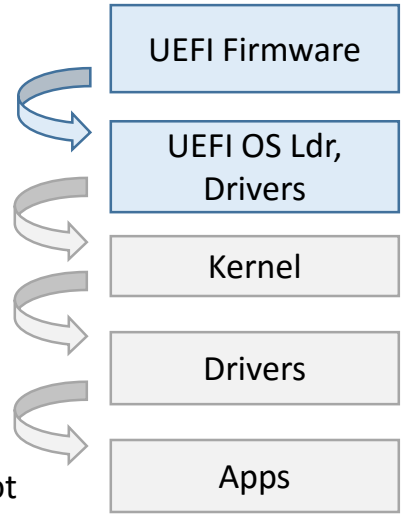
Chapter 27.2 of The UEFI 2.4 Specification

Trusted Versus Secure Boot

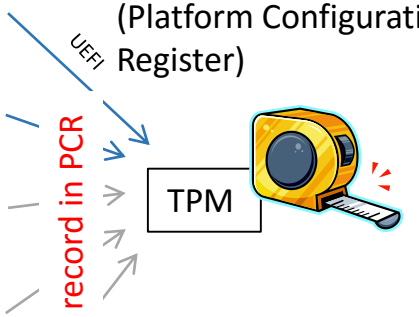


UEFI authenticate OS loader
(pub key and policy)

Check signature of before loading



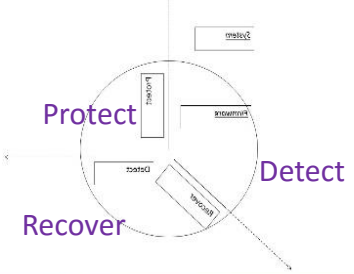
UEFI PI will measure OS loader & UEFI drivers into TPM (1.2 or 2.0) PCR (Platform Configuration Register)



- UEFI Secure boot will stop platform boot if signature not valid (OEM to provide remediation capability)
- UEFI will require remediation mechanisms if boot fails

- TCG Trusted boot will never fail
- Incumbent upon other software to make security decision using attestation

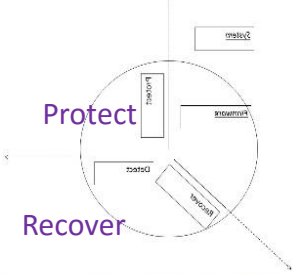
EDKII Defenses (1/3)



- UEFI Secure and TCG Measured boot - Upper layer API's defined in UEFI and TCG spec
  - Implementation source + document for UEFI Secure & TPM measured boot
    - https://github.com/tianocore/edk2/tree/master/SecurityPkg
    - https://firmware.intel.com/sites/default/files/resources/A_Tour_Beyond_BIOS_Implementing_TPM2_Support_in_EDKII.pdf
  - http://bluestop.org/edk2/docs/specs/A_Tour_Beyond_BIOS_into_UEFI_Secure_Boot_White_Paper.pdf

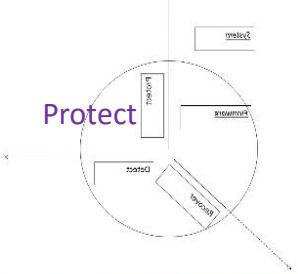
- Signed updates - Capsules defined in UEFI Spec
  - Signed updates required by NIST 800-147 http://csrc.nist.gov/publications/nistpubs/800-147/NIST-SP800-147-April2011.pdf for core and NIST-193 http://csrc.nist.gov/publications/drafts/800-193/sp800-193-draft.pdf for platform components
  - https://github.com/tianocore/edk2/tree/master/SignedCapsulePkg
  - https://github.com/tianocore-docs/Docs/raw/master/White_Papers/A_Tour_Beyond_BIOS_Capsule_Update_and_Recovery_in_EDK_II.pdf

EDKII Defenses (2/3)



- Defense in depth mechanisms of UEFI Core
  - NX, ASLR, stack canaries during pre-OS https://www.gitbook.com/book/edk2-docs/a-tour-beyond-bios-memory-protection-in-uefi-bios/details
- Hardened SMM
  - Implementing the Windows SMM Mitigation Table (WSMT) https://github.com/tianocore-docs/Docs/raw/master/White_Papers/A_Tour_Beyond_BIOS_Secure_SMM_Communication.pdf
  - A_Tour_Beyond_BIOS_Securiy_Enhancement_to_Mitigate_Buffer_Overflow_in_UEFI https://github.com/tianocore-docs/Docs/raw/master/White_Papers/A_Tour_Beyond_BIOS_Securiy_Enhancement_to_Mitigate_Buffer_Overflow_in_UEFI.pdf
- And lock-box protocol, user identification, password, and random number generation
- Overall dev BKM (Best Known Methods)
  - Security BKM https://github.com/tianocore-docs/Docs/raw/master/White_Papers/A_Tour_Beyond_BIOS_Security_Design_Guide_in_EDK_II.pdf

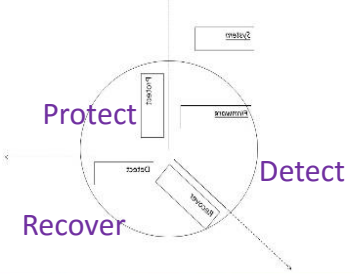
Continued SMM Defenses



- Intel STM (SMI Transfer Monitor): https://firmware.intel.com/content/smi-transfer-monitor-stm
- BIOS Guard (PFAT): SMM compromises do not directly lead to flash access (persistence)
- Other SMM protections:

SMM Attack	Mitigation
SMRAM is unlocked	1) Lock SMRAM at PI SmmReadyToLock
Cache Poisoning	1) Enable SMM Range Register MSR
SMRAM remap	1) Lock Remap register
Branch Outside of SMRAM	1) Enable Smm_Code_Access Register MSR 2) Setup Non-Executable (NX) paging outside of SMM
SMM Communication Buffer Attack	1) Check SMM Communication Buffer 2) Check MMIO bar access

EDKII Defenses (3/3)



- VT-d boot protection
  - https://lists.01.org/pipermail/edk2-devel/2017-April/009454.html
    https://firmware.intel.com/sites/default/files/resources/A_Tour_Beyond_BIOS_Using_Intel_VT-d_for_DMA_Protection.pdf
- And to pull things together - Full open source EDKII-based platforms
  - Quark https://github.com/tianocore/edk2/tree/master/QuarkPlatformPkg
  - Atom BYT https://github.com/tianocore/edk2-platforms/tree/minnowboard-max-udk2015
  - Atom APL https://github.com/tianocore/edk2-platforms/tree/devel-MinnowBoard3-UDK2017
  - Intel KBL https://github.com/tianocore/edk2-platforms/tree/devel-MinPlatform core platform tree
- Atom and core platforms based upon Intel Firmware Support Package (FSP)
  - https://github.com/IntelFsp/FSP
  - https://firmware.intel.com/sites/default/files/A_Tour_Beyond_BIOS_Using_the_Intel_Firmware_Support_Package_with_the_EFI_Developer_Kit_II_%28FSP2.0%29.pdf
  - http://www.apress.com/us/book/9781484200711
- Very through training package made available by Intel's ATR* – covers chipsec
  - https://github.com/advanced-threat-research/firmware-security-training
  - **Intel's ATR moved from SeCoE to Intel Security and is now part of McAfee**

Dataset & Methodology (1/2)

- It is quite hard to provide really good analysis of datasets related to security issues:
  - A must see if somehow you wonder why: https://media.blackhat.com/us-13/US-13-Martin-Buying-Into-The-Bias-Why-Vulnerability-Statistics-Suck-Slides.pdf

- We used Intel's PSIRT Data in different ways:
  - For the past 3 years (***124*** issues related to BIOS/Firmware), to give an 'idea' of the amount of issues (which adds attrition bias as well)
  - For as long as we could find (Circa 2007), to generate taxonomy

- Intel's PSIRT data include issues found internally (vendor selection bias) that somehow were identified to affect released products
  - It does not mean every internally found issue is there
  - The way the data is capture changed along the years (hopefully for better), but that is one of the mistakes clearly stated in the aforementioned presentation ☺
  - We constantly receive reports for issues that affect a specific OEM-only (and as so, have nothing to do with Intel – **please, keep involving us, it is helpful!**): We counted those too, but obviously, lots of issues are not reported to us as well (that also adds a researcher selection bias, since some researchers always involve us and as so, whatever issues they are finding, are accounted for, but not all researchers do that)
  - PSIRT does sometimes group multiple issues under one case (specially internal ones), and as so, the count itself also added an abstraction bias
  - For some other cases, issues are divided in multiple cases (given it potentially spawning multiple different actions): We tried to clean the obvious cases
  - We only considered the confirmed issues that affect at least one product, therefore there is also publication bias

- Total number of issues considered/used: ***77 issues***

Dataset & Methodology (2/2)

- **With so many problems in the dataset, does it have any value?**
  - **Yes**, but not to compare platforms (so we do not include such information)
  - **Yes**, for defining a taxonomy (so we did generate/propose one)
  - **Yes**, to help defining next steps/future strategies (so we did do that)
  - **Maybe**, to define trends (as in: are those kind of problems getting more external attraction?)
  - **No,** if you want to say that there are more bugs (not bugs being reported, which is the previous point) now than few years back -> **DO NOT USE THE DATA FOR THAT PURPOSE** ☺

HW Bugs?

- Those bugs are not BIOS-bugs per-se, but they do affect platform security and what is expected from BIOS (**we did not consider them in the total count presented**, and we did group those in the taxonomy as a single class: 'HW bugs'). Some examples:
  - LAPIC bug (memory range overlapping and decoding priorities leading to SMRAM access)
  - SMM memory written-back cached data from Ring0 leading to SMRAM access
  - Fuse mis-configuration for BootGuard leading to platform compromise (See for example "Safeguarding rootkits: Intel BootGuard" presented at ZeroNights 2016 (https://dsec.ru/upload/medialibrary/82b/82b222a0ab6470a724108b42208f0630.pdf)

Proposal of Bug Classes (1/2)

- We do not claim it to be a comprehensive list, but it is comprehensive enough to cover all PSIRTs received by Intel
  - Many bugs might fit multiple classes (since one problem might lead to another, we use the first in the chain to aggregate)
  - **Help evolve this list**
- Notice that we grouped together things that are a bit more classical software cases (for example, not properly validating untrusted input)
  - But we still called out ones that have some specificities due to the expectations for the whole platform security
- The main intention: be able to easily define a class for a given issue and to remove the sentiment (shared by many researchers) that BIOS security is something obscure
  - Notice that when we group the classes per the number of reported issues in each ☺

The main complexity is the sheer number of platform-specific configurations

Proposal of Bug Classes (2/2)

- **Inconsistent power-transition checks**
  - Bad Initialization State (secrets kept in memory, assumption of certain system state)
  - S3 Boot Script
- **Race Condition**
  - Enabling protection mechanisms (including disabling/enabling certain HW elements, including wrong module launching order)
  - Time of Check/Time of Use (TOCTOU) Race Conditions
- **Trusting input** (from less privileged entities in the platform, or having different assets configured by different threat vectors)
  - Could lead to race conditions (double fetches and TOCTOUs, for example if a pointer is pointing to unprotected memory area and therefore can be arbitrarily modified after any checks)
- **Measurement failures**
  - Not measuring certain modules
  - Accepting signed updates, but for the wrong platform due to using same keys
  - Accepting to load certain modules in the wrong time (when certain assets are not properly protected)
  - Failing to identify a measurement error/problem
- **Platform capability not properly configured**
  - Locks not set, devices not properly initialized, features not disabled, etc
- **Security of meaningful assets exposed to untrusted entities**
  - E.g.: UEFI variables
- **HW Misbehavior**
  - Wrongly defined architecture, micro-architecture, bad design, etc
  - Platform components behaving differently than specified

Examples (1/7)

- Inconsistent power-transition checks
  - Bad Initialization State (secrets kept in memory, assumption of certain system state)
  - S3 Boot Script
- Many talks mention the S3 Boot Script vulnerability and public exploits/write-ups are available (http://blog.cr4.sh/2015/02/exploiting-uefi-boot-script-table.html)
- Some of the bugs in this category can be very subtle, like when in a system is turned on a given code path is taken which copies certain data structures inside SMRAM for parsing, while when the system is coming back from resume the SMM code consumes external tables directly (therefore, wrongly trusting the runtime OS)

Examples (2/7)

- Race Condition
  - Enabling protection mechanisms (including disabling/enabling certain HW elements, including wrong module launching order)
  - Time of Check / Time of Use TOCTOU Race Condition

- A good example we saw was a third party option ROM being launched before EndOfDxe (lots of assets unprotected)

- Another common case is SMM code validating input *before* copying it to protected memory (which could be altered after the checks)
  - We experimenting with extending Simics to catch some of those (See: “Windows Kernel Race Condition Analysis While Accessing User-mode Data” paper in PoC || GTFO for some of our tests/experiments reproducing Project Zero’s work on Windows for similar problem) https://www.alchemistowl.org/pocorgtfo/pocorgtfo15.pdf

Examples (3/7)

- Trusting input (from less privileged entities in the platform, or having different assets configured by different threat vectors)
  - Could lead to race conditions (double fetches and TOCTOUs, for example if a pointer is pointing to unprotected memory area and therefore can be arbitrarily modified after any checks)
  - Traditional code bugs apply here (like buffer and integer overflows)
- Lots presented/found along the years by Intel's ATR, SeCoE and other internal and external researchers
- Many caught by our Excite Project (BIOS symbolic execution) (https://github.com/REhints/Publications/blob/master/Conferences/ZeroNights_2016/Excite_Project_ZN.pdf)

Examples (4/7)

- Measurement failures
  - Not measuring certain modules
  - Accepting signed updates, but for the wrong platform due to using same keys
  - Accepting to load certain modules in the wrong time (when certain assets are not properly protected)
  - Failing to identify a measurement error/problem
- Two real examples we've seem:
  - Client & Server parts using the same keys, meaning that an image from one would load in the other, but obviously bricking the platform
  - TE (Terse Executables) are loaded correctly, but not measured since deeper parts check for specific PE header presence (but the boot succeeds)

Examples (5/7)

- Platform capability not properly configured
  - Locks not set, devices not properly initialized, features not disabled, etc
  - Lots of times guidance is just not being followed
  - Chipsec use by OEMs and consumers would identify most of these
  - See the Chipsec Platform Security Assessment Framework

https://github.com/chipsec/chipsec

Watch the talk at 5pm: "Betraying the BIOS: Where the guardians of the BIOS are failing" for real life recent examples ☺. Callout to Alex Matrosov

Examples (6/7)

- Security of meaningful assets exposed to untrusted entities
  - E.g.: UEFI variables
- At least one case we've seem included UEFI variables that control PK/KEK not requiring physical presence to be modified
  - That essentially undermines Secure Boot
- Some times, sample code has validation features that should be disabled/removed from production versions, such as a UEFI variable that sets the policy responsible for configuring EISS (also known as SMM_BWP, which essentially opens the flash for writes from outside of SMM) bit in the SPI controller to be false

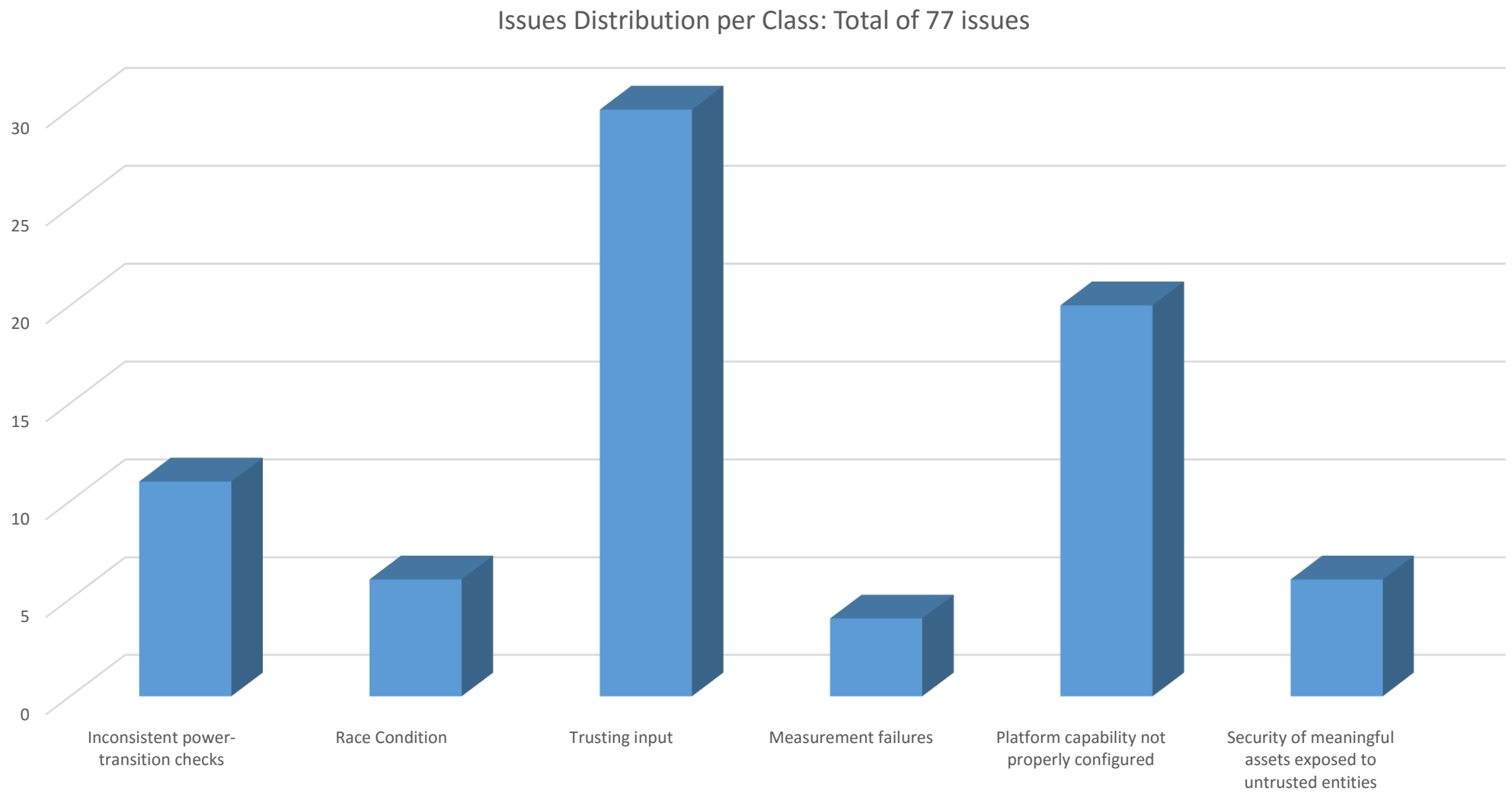
Examples (7/7)

- HW Misbehavior
  - Wrongly defined architecture, micro-architecture, bad design, etc
  - Platform components behaving differently than specified
- See Intel's Erratas ☹ We do our best, but things fail. Complexity and composition are security's worst enemies

- Examples of those:
  - The Memory Sinkhole (by Christopher Domas in Black Hat USA 2015) https://www.blackhat.com/docs/us-15/materials/us-15-Domas-The-Memory-Sinkhole-Unleashing-An-x86-Design-Flaw-Allowing-Universal-Privilege-Escalation-wp.pdf
  - Phrack article "Using SMM for Other Purposes" (http://phrack.org/issues/65/7.html) release of a weird cache behavior, followed by Joanna's and Rafal's work proving it was an exploitable vulnerability in "Attacking SMM Memory via Intel CPU Cache Poisoning" (http://invisiblethingslab.com/resources/misc09/smm_cache_fun.pdf)
  - McAfee ATR work on "BARing the System" at Recon2017 (http://www.intelsecurity.com/advanced-threat-research/content/data/REConBrussels2017_BARing_the_system.pdf)

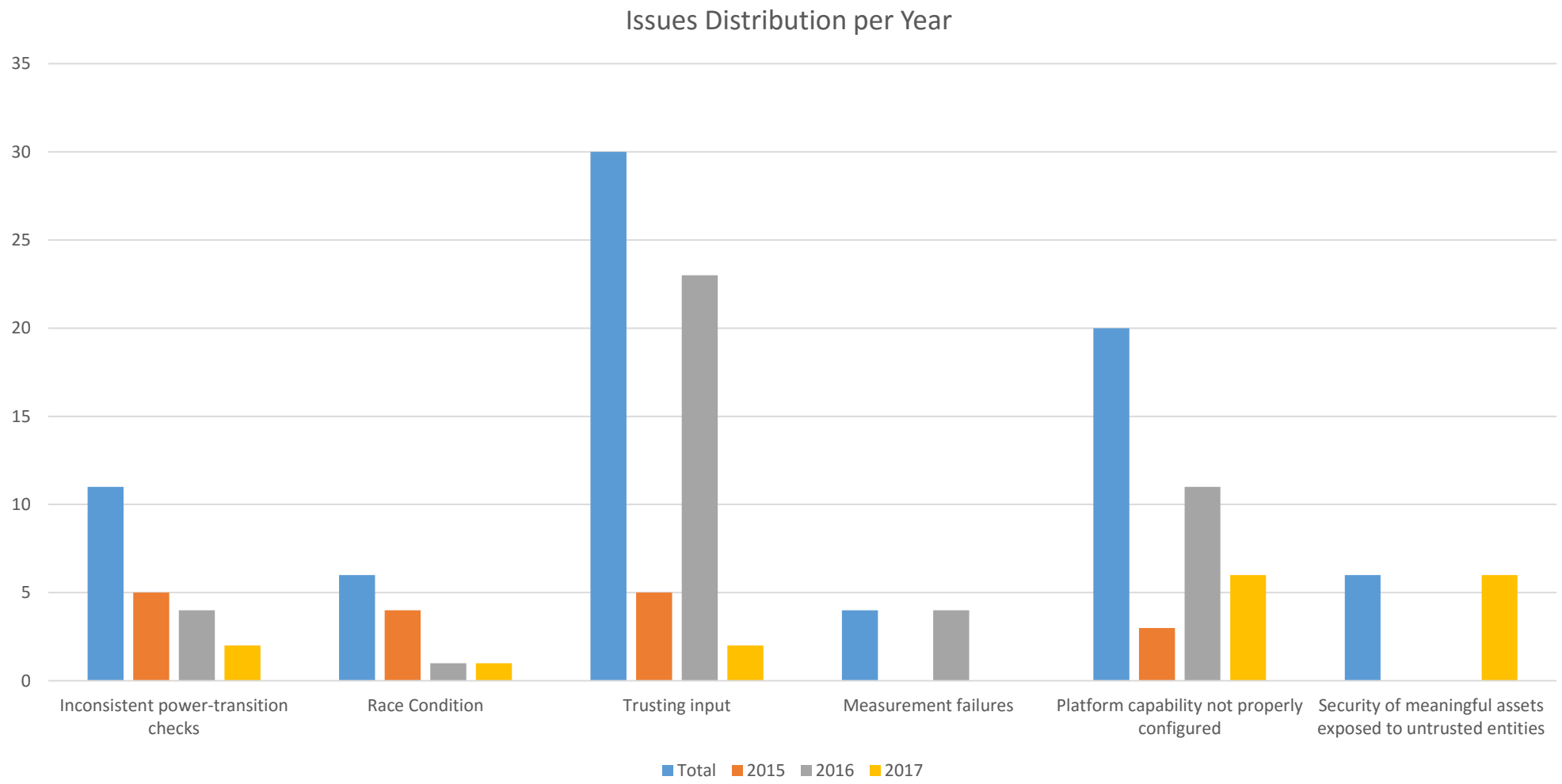
Bug Class Distribution

2015-2017 (Past Week – Middle July)



Bug Class Distribution per Year

2015-2017 (Past Week – Middle July)



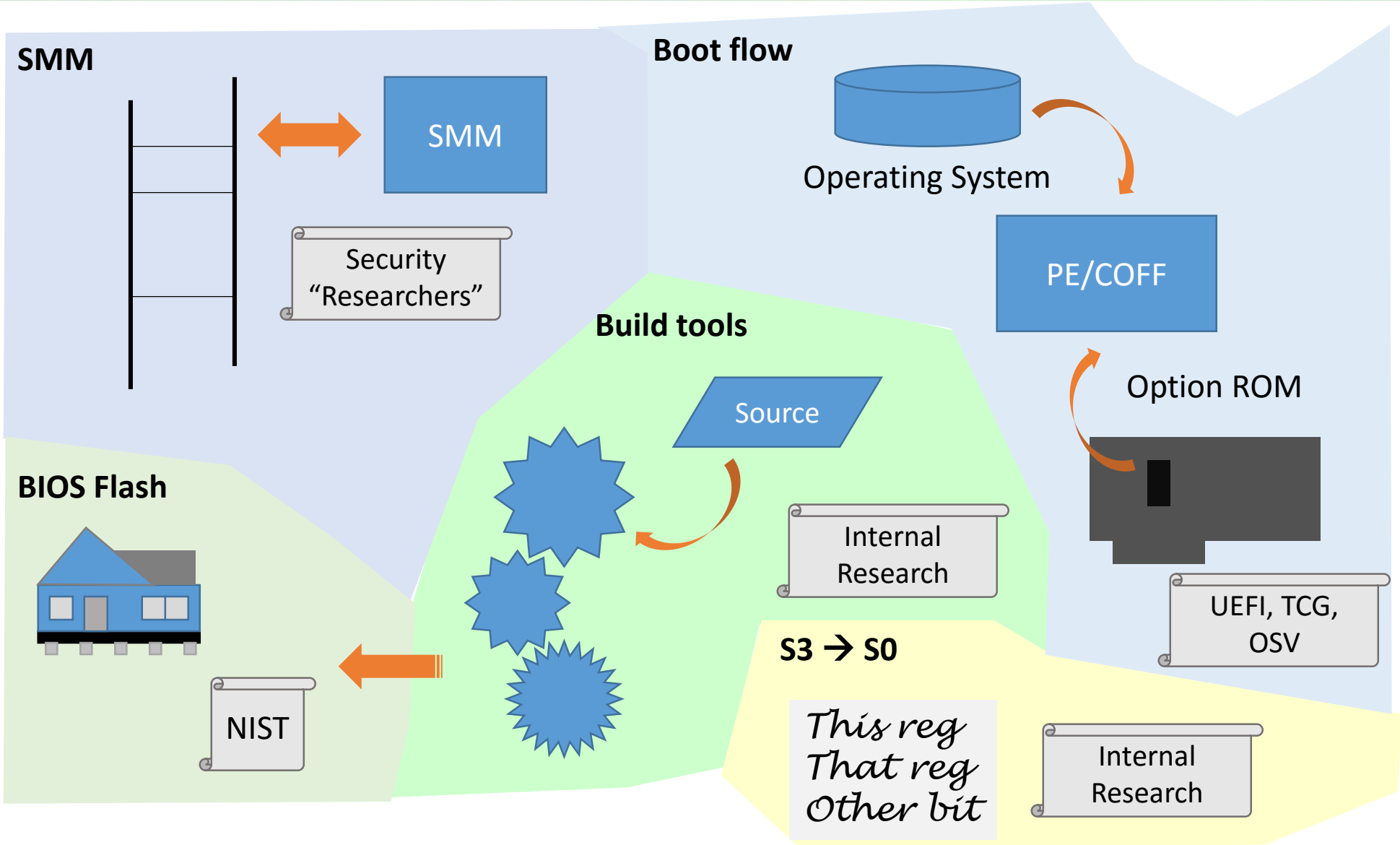
Rationale for a Threat Modeling

- "My house is secure" is almost meaningless
  - Against a burglar? Against a meteor strike? A thermonuclear device?
- "My system is secure" is almost meaningless
  - Against what? To what extent?
- Threat modeling is a process to define the goals and constraints of a (software) security solution
  - Translate user requirements to security requirements
- We used threat modeling for our UEFI / PI codebase
  - We believe the process and findings are applicable to driver implementations as well as UEFI implementations in general

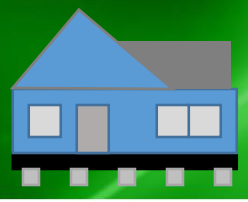
Defining, using a Threat Modeling

- A Threat Model (TM) defines the security assertions and constraints for a product
  - Assets: What we're protecting
  - Threats: What we're protecting it against
  - Mitigations: How we're protecting our Assets
- Use TM to narrow subsequent mitigation efforts
  - Don't only perform a secure review, fuzz test all interfaces
  - Select the ones that are critical
- TM is part science, part art, part experience, part nuance, part preference
  - Few big assets vs lots of focused assets

We don’t always get to choose our assets



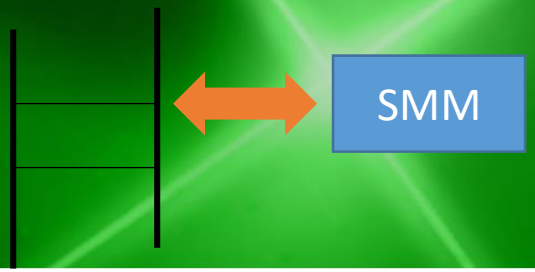
Flash**



- NIST SP800-147 says
  - Lock code flash except for update before Exit Mfg Auth (Common Bugs: HW Misbehavior, Platform capability not properly configured)
  - Signed update (>= RSA2048, SHA256) (Common Bugs: Security meaningful assets exposed to untrusted entities, Trusting input, Measurement Failures)
  - High quality signing servers (Outside of the scope of the BIOS itself, but signing keys must be secret, etc)
  - Without back doors (“non-bypassability”) (Common Bugs: A group of the above)
- Threats
  - PDoS – Permanent Denial of Service
    - System into inefficient room heater
  - Elevation of privileges
    - Owning the system at boot is an advantage to an attacker
- Known attacks
  - CIH / Chernobyl 1999-2000
  - Mebroni 2010
  - Hacking Team 2015
  - Various reputed government leaks (2016, 2017…)
- Mitigations include
  - Reexamining flash protection methods – use the best even if its new
  - Using advanced techniques to locate and remove vulnerabilites

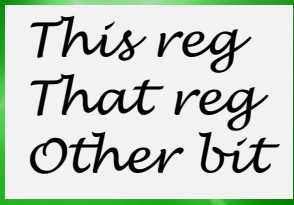
** or tomorrow’s equivalent NV storage

SMM



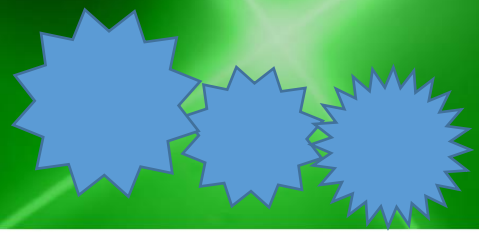
- SMM is valuable because
  - It's invisible to System Software (including Anti Virus, etc)
  - SMM sees all of system RAM (standard notion, without STM)
  - Not too different from PCI adapter device firmware
  - Most BIOS bugs are somehow related to getting SMM access (so all classes apply)
- Threats
  - Elevation of privileges
    - View secrets or own the system by subverting RAM
- Known attacks
  - See e.g: Duflot, "Phrack Using SMM for Other Purposes and others", Legbacore, McAfee's Advanced Threat Research, Cr4sh
- Mitigations include
  - Validate "external" / "untrusted" input
  - Remove calls from inside SMM to outside SMM

Resume from S3



- ACPI says that we return the system to the S5→S0 configuration at S3→S0
  - Must protect the data structures we record the cold boot config in (Common Bugs: Inconsistent power-transition checks, Race Condition, Security of meaningful assets exposed to untrusted entities)
- Threats
  - Changing data structures could cause security settings to be incorrectly configured leaving S3
  - Reopen the other assets' mitigated threats
- Known attacks (McAfee ATR, Legbacore, cr4sh)
- Mitigations include
  - Store data in SMM -or-
  - Store hash of data structures and refuse to resume if the hashes don't compare

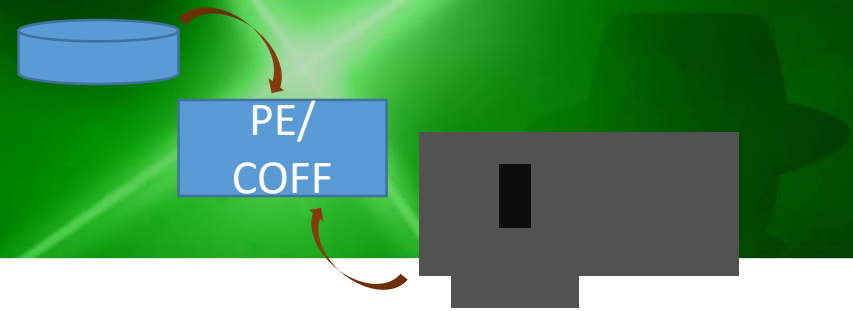
Tool chain



- Tools create the resulting firmware
  - Rely on third party tools and home grown tools
  - Incorrect or attacked tools leave vulnerabilities
  - We did not add a class for those (instead we tried to tie the final bug to a class)
- Threats
  - Disabled signing, for example
- Known attacks
  - See e.g. *Reflections on Trust*, Ken Thompson**
- Mitigation
  - Difficult: For most tools, provided as source code
  - Review for correct implementation
  - Use static, dynamic code analysis tools
    - PyLint for Python, for example

** CACM, Vol 27, No 8, Aug, 1984, pp. 761-763

Boot flow



- Secure boot (Common Bugs: Measurement failures, Platform capability not properly configured, Security of meaningful assets exposed to untrusted entities, Race Condition)
  - Authenticated variables
  - Based on the fundamental crypto being correct
  - Correct location for config data
- Threats
  - Run unauthorized op roms, boot loaders
  - PDoS systems with bad config variables
- Known attacks
- Mitigations include
  - Sanity check config vars before use, use defaults
  - Reviews, fuzz checking, third party reviews, etc.

Future

- We can't foresee the future, but there are lots of things that Intel is planning and that the overall industry can help!
  - The release of a Platform Security Configuration Guide, independent of the BIOS Writer Guide (which currently already have a section for security)
  - Release of more Chipsec and HSTI security checks
  - Increase in mitigating technologies as part of the platform (some were discussed in this talk), to evolve the state-of-the-art
  - Share of other initiatives, as we did for example with Excite (symbolic execution tool, currently used to automate finding SMM call-outs and having heavy investment)

- What ***you*** can do to help? ***OUR* CALL FOR ACTIONS!**
  - The ecosystem is complex, help naming the names and demonstrating the value to whoever cares about security
  - Always question if attack demonstrated are against the state-of-the-art available, or are just one more configuration mistake (and mention the state-of-the-art if you are a researcher)
  - Help us evolving such state-of-the-art: propose new attack classes (as some of you are already doing), propose architectural improvements: Dream high! Let us care about making it a reality ☺
  - If you are an end-user: Those attacks are *NOT* complex. Understand and define your threat model, run the available tools and pressure your vendor!

Acknowledgements

- Lots of different individuals (and teams) inside and outside of Intel contribute to the evolution of UEFI security, some names from Intel (knowing very well we forgot many):
  - Bulygin, Yuriy; Govindarajan, Sugumar; Mathews, John; Loucaides, John; Delgado, Brian; Yao, Jiewen; Frinzell, Aaron; Brannock, Kirk; Wolman, Ayellet; Rosenbaum, Lee

- Without them, this talk would not have been possible and we strongly appreciate their help, continuous feedback and work

- To all the researchers that collaborate with Intel, directly or indirectly (yes, any form of report, including full-disclosure contributed to actions, decisions and direction changes) -> **Thank you!**

THE END! IS IT?

QUESTIONS?

Bruce Monroe & Rodrigo Rubira Branco (@bsdaemon) & Vincent Zimmer (@vincentzimmer)
{ bruce.monroe || rodrigo.branco || vincent.zimmer } @ intel.com