Google

Securing the Unseen: Vulnerability Research in Confidential Computing

Hardwear.io USA 2023

Cfir Cohen

Josh Eads

Agenda

- Confidential Computing
  - Background & Threat Model
- AMD SEV-SNP Security Review
  - Methods, Tools & Findings
- Intel TDX Security Review
  - Methods, Tools & Findings
- Summary

Reviews Background

- Motivation: Auditor role critical in building trust
  - See “sealed computation framework” paper
- Whitebox security audit
  - Close collaboration with CPU vendors
  - Access to hardware, design docs and source code
- Impactful
  - >30 CVEs & confirmed issues
- Transparent
  - 2022: AMD SNP security report
  - 2023: Intel TDX security report
- Acknowledgements
  - **James Forshaw, Jann Horn, Mark Brand, Felix Wilhelm, Erdem Aktas**
  - **Awesome AMD & Intel engineers**

2021

SNP Review Kickoff

TDX Review Kickoff

2022

SNP Review Published

2023

TDX Review Published

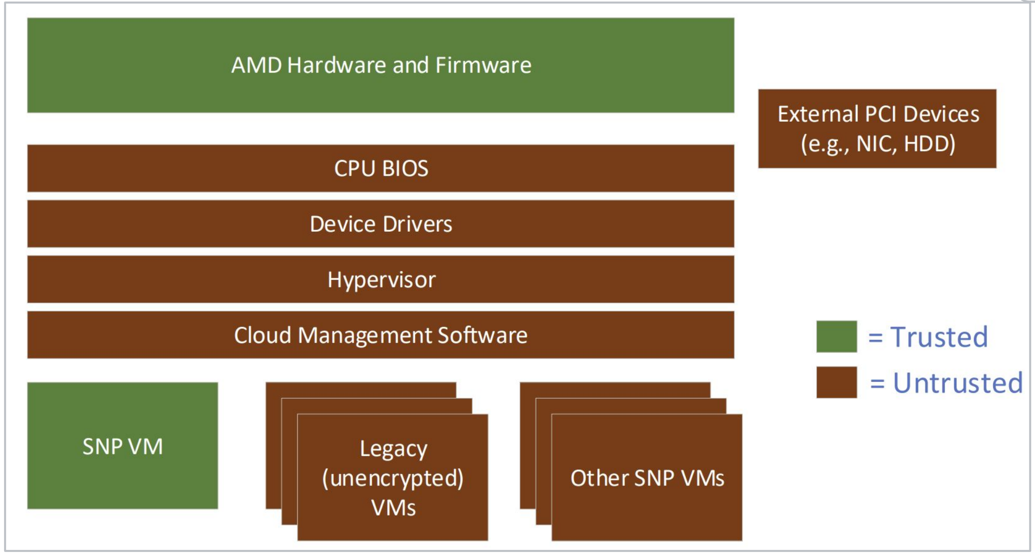
Confidential Computing

Confidential, Protected, Sealed Computing

- Hardware-based trusted execution
  - **Remote attestation**: code identity w/ quotes rooted in hardware.
  - **Strong isolation**: from hypervisor, peripheral devices, colocated tenants.

Initial state security guarantees

Runtime security guarantees



AMD SEV-SNP "Strengthening VM Isolation" whitepaper

Google

Confidential, Protected, Sealed Computing

- Hardware-based trusted execution
  - **Remote attestation**: code identity w/ quotes rooted in hardware.
  - **Strong isolation**: from hypervisor, peripheral devices, colocated tenants.
- Paradigm shift
  - **Give control** back to customer: platform TCB is auditable, verifiable and therefore **trusted**.
  - Protection of **data in use**: policy centered around authentic code & secure processing env.

Confidential, Protected, Sealed Computing

- Hardware-based trusted execution
  - **Remote attestation**: code identity w/ quotes rooted in hardware.
  - **Strong isolation**: from hypervisor, peripheral devices, colocated tenants.
- Paradigm shift
  - **Give control** back to customer: platform TCB is auditable, verifiable and therefore **trusted**.
  - Protection of **data in use**: policy centered around authentic code & secure processing env.
- Innovation trigger
  - **Usable**: VM based compute model supports lift-and-shift
  - **Available**: support across CPU vendors and Cloud providers
  - **Enabler**: data clean room and Multi-Party Compute solutions

Threat Model: Powerful Adversary In A Target Rich Env

- Deprivileged host OS:
  - Trusted for **resource management**, not **resource access**
- Adversarial host system has many capabilities:
  - Bad system configurations
  - Large API surface
  - DMA from peripheral devices
  - Control over scheduling
  - Memory access oracles
- Physical access attacks not in-scope
  - No DRAM interposers
  - No voltage glitching, etc.

Host firmware is untrusted

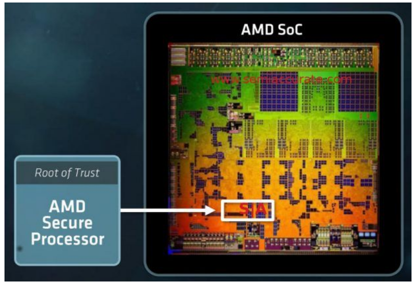
Architectural limitations

Google

AMD SEV-SNP Security Review

AMD SEV-SNP Components

- AMD Secure Processor (ASP)
  - Isolated execution environment; Exposes mailbox interface
  - Hosts security sensitive components e.g SEV firmware
- SEV firmware
  - Manages guest state, **nested page tables**, **page ownership table (RMP)**
  - Programs in-line memory encryption keys
  - Signs attestations
- CPU uCode
  - Checks page ownership on every store
- IOMMU
  - Checks page ownership during IOVA translations
- Memory Controller
  - Inline AES memory encryption for PAs with C-bit set



Ownership & Access Checks: Integrity guarantees

Encryption: Confidentiality guarantees

Google

Research Methodologies & Tools

- There is a **systematic** way to find high quality bugs.
- Things that worked well for us:
    - “Invariant analysis”
    - Layered crypto reviews
    - Emphasis on performance - security tradeoffs
    - Emphasis on security checks
    - Interaction between components

- Wycheproof for crypto tests
- PCIe screamer for hardware tests

Layered Crypto Review

Implementation	Correctness, random selection bias, secret dep ops, etc. Attackers target the implementation, not the math.
Algorithms	Choice of building blocks: encryption, digest, signing algo and schemes. Weak algorithms (MD5), weak modes (ECB), weak schemes (unauthenticated ciphertext).
Protocols	How building blocks assembled to form a secure channel. Looking for secrecy, authenticity, freshness, strong identity binding.

Persistent storage key reuse

- Authenticated encryption scheme: AES-CTR, HMAC
- Fixed IV leads to “two time pad”

```
$ diff <(xxd psp_nv_data0) <(xxd psp_nv_data1)
1,4c1,4
< 00000000: 6d53 fc56 9643 4a38 7445 6672 03cf 55d4  mS.V.CJ8tEfr..U.
< 00000010: b3dc 03a1 235d 308b 85f4 d680 041a b8aa  ....#]0.........
< 00000020: 274c 6959 8236 577e 83b9 ddc1 0fbc 4581  'LiY.6W~......E.
< 00000030: 809b 5ef5 5741 a3d1 59d4 9190 46e1 8cba  ..^.WA..Y...F...
---
> 00000000: 6d53 fc56 9643 4a38 7445 6672 3983 70da  mS.V.CJ8tEfr9.p.
> 00000010: 049a 6fa0 a7ef 5414 3191 5c98 6054 8980  ..o...T.1.\.`T..
> 00000020: 239f 1262 a710 1377 d676 9250 b20a 3e88  #..b...w.v.P..>.
> 00000030: c686 cbfa a911 08d2 d068 0f3f 46e1 8cba  .........h.?F...
```

Persistent storage key reuse

- Authenticated encryption scheme: AES-CTR, HMAC
- Fixed IV leads to “two time pad”
- Bypasses fix for 2019 [key recover vul](#).

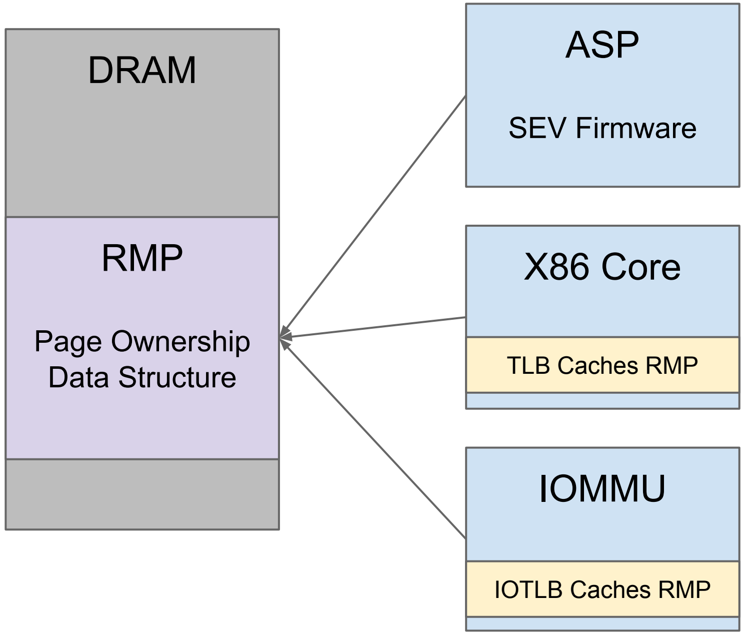
Somewhere here is the encrypted private key. If you recover it using “invalid curve” bug from 2019, you can XOR recover the storage key stream, then recover the private key of a patched system.

```
$ diff <(xxd psp_nv_data0) <(xxd psp_nv_data1)
1,4c1,4
< 00000000: 6d53 fc56 9643 4a38 7445 6672 03cf 55d4  mS.V.CJ8tEfr..U.
< 00000010: b3dc 03a1 235d 308b 85f4 d680 041a b8aa  ....#]0.........
< 00000020: 274c 6959 8236 577e 83b9 ddc1 0fbc 4581  'LiY.6W~......E.
< 00000030: 809b 5ef5 5741 a3d1 59d4 9190 46e1 8cba  ..^.WA..Y...F...
---
> 00000000: 6d53 fc56 9643 4a38 7445 6672 3983 70da  mS.V.CJ8tEfr9.p.
> 00000010: 049a 6fa0 a7ef 5414 3191 5c98 6054 8980  ..o...T.1.\.`T..
> 00000020: 239f 1262 a710 1377 d676 9250 b20a 3e88  #..b...w.v.P..>.
> 00000030: c686 cbfa a911 08d2 d068 0f3f 46e1 8cba  .........h.?F...
```

Google

IOMMU TLB not flushed on SNP-INIT

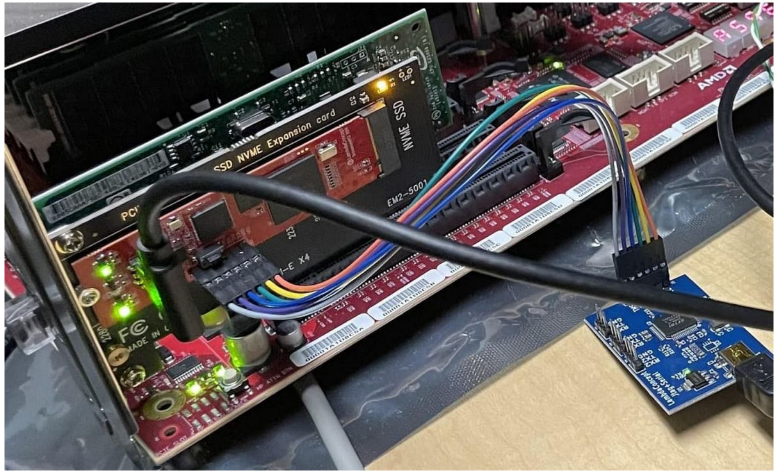
- Invariant: “**Page ownership information is coherent across components**”
- IOMMU caches RMP entries => IOTLB should be flushed on RMP updates.
- Firmware updates RMP on init, but failed to flush IOTLB.



IOMMU TLB not flushed on SNP-INIT

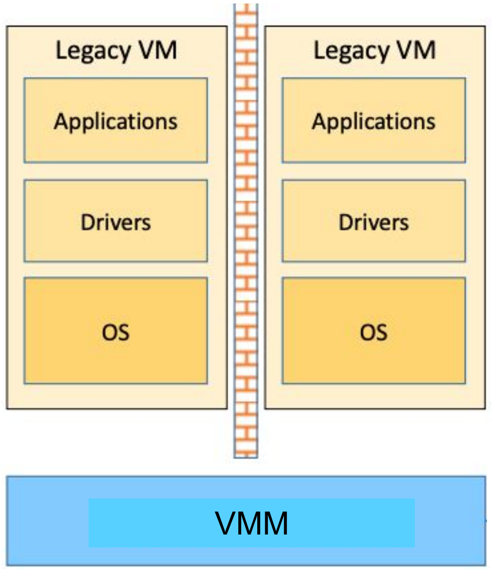
- DMA aggressor (screamer) poisons IOTLB
- Stale IOTLB entries lead to unchecked write

PCI Screamer:

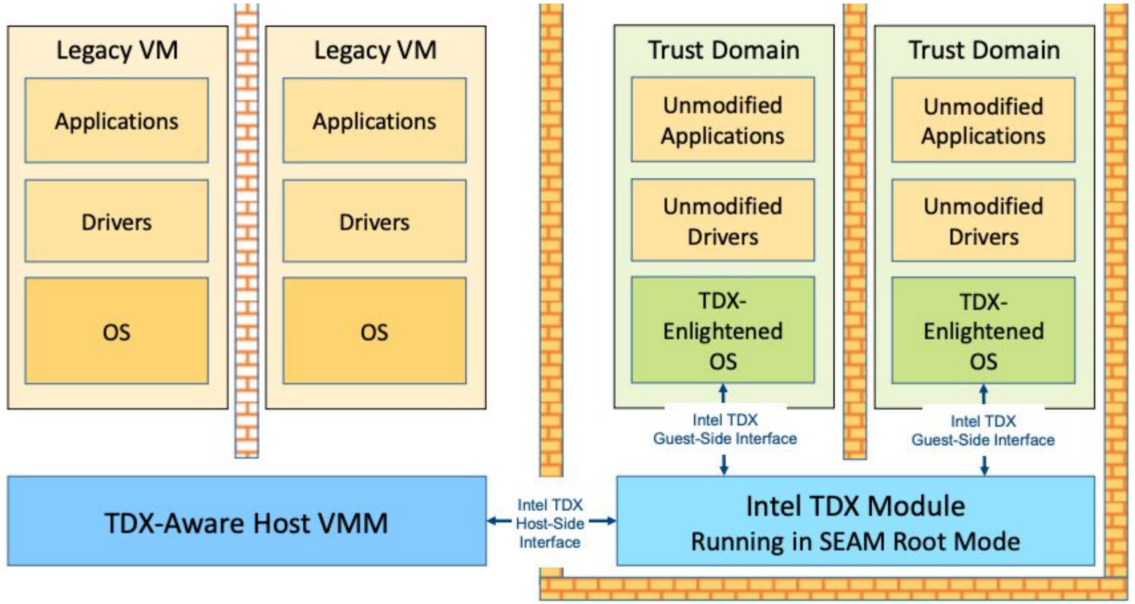


Intel TDX 1.0 Security Review

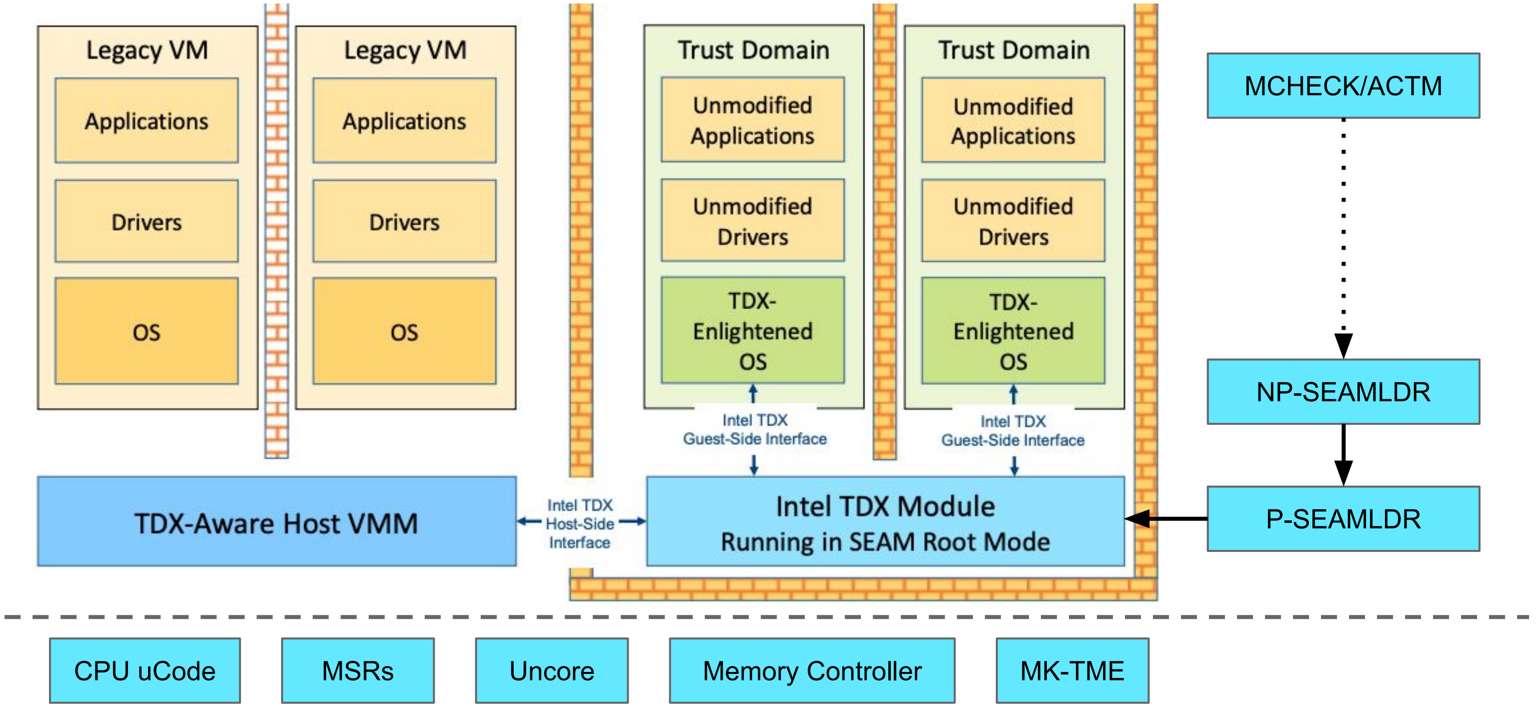
VMX in a Nutshell



TDX in a Nutshell

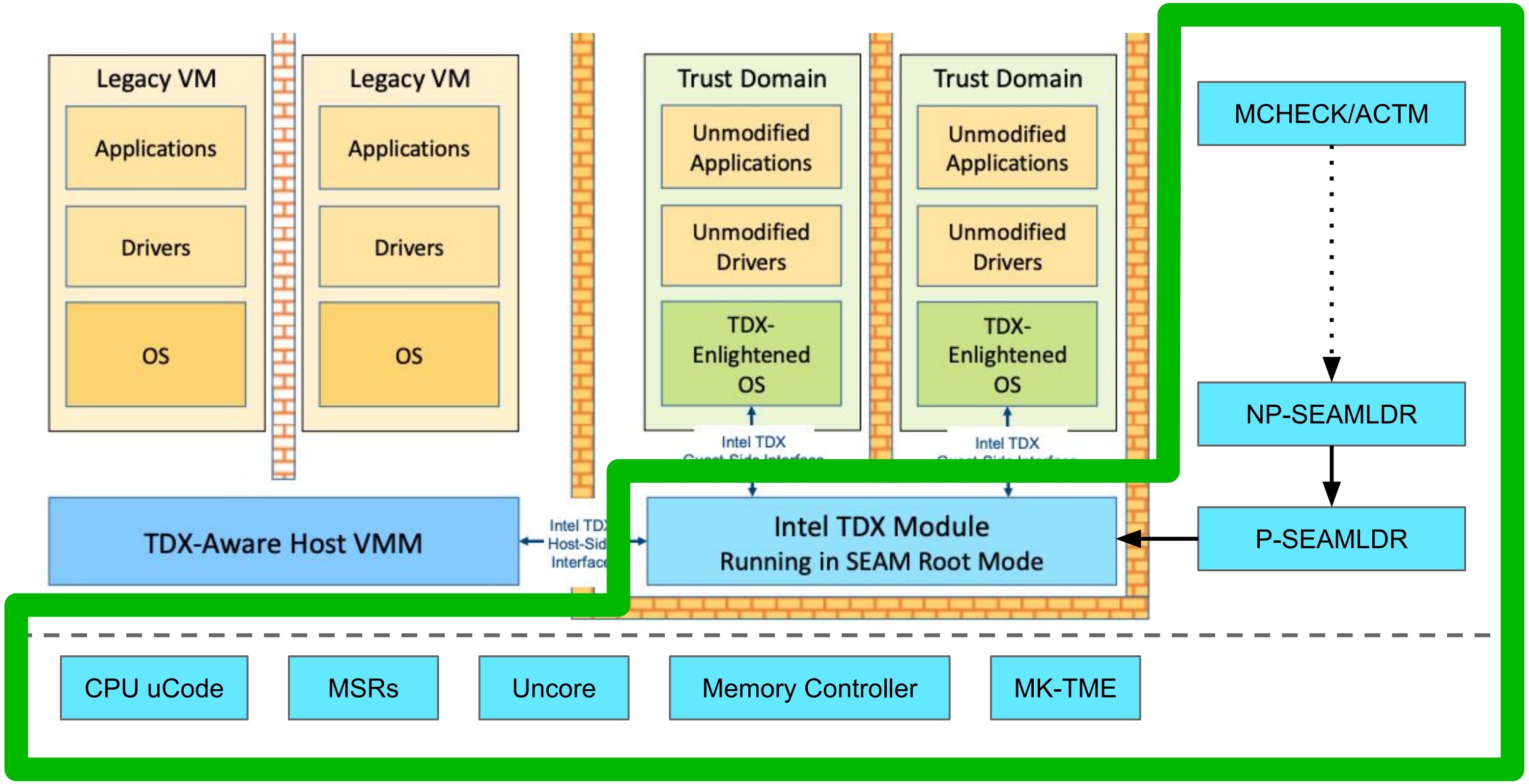


TDX in a Nutshell - Supporting Cast



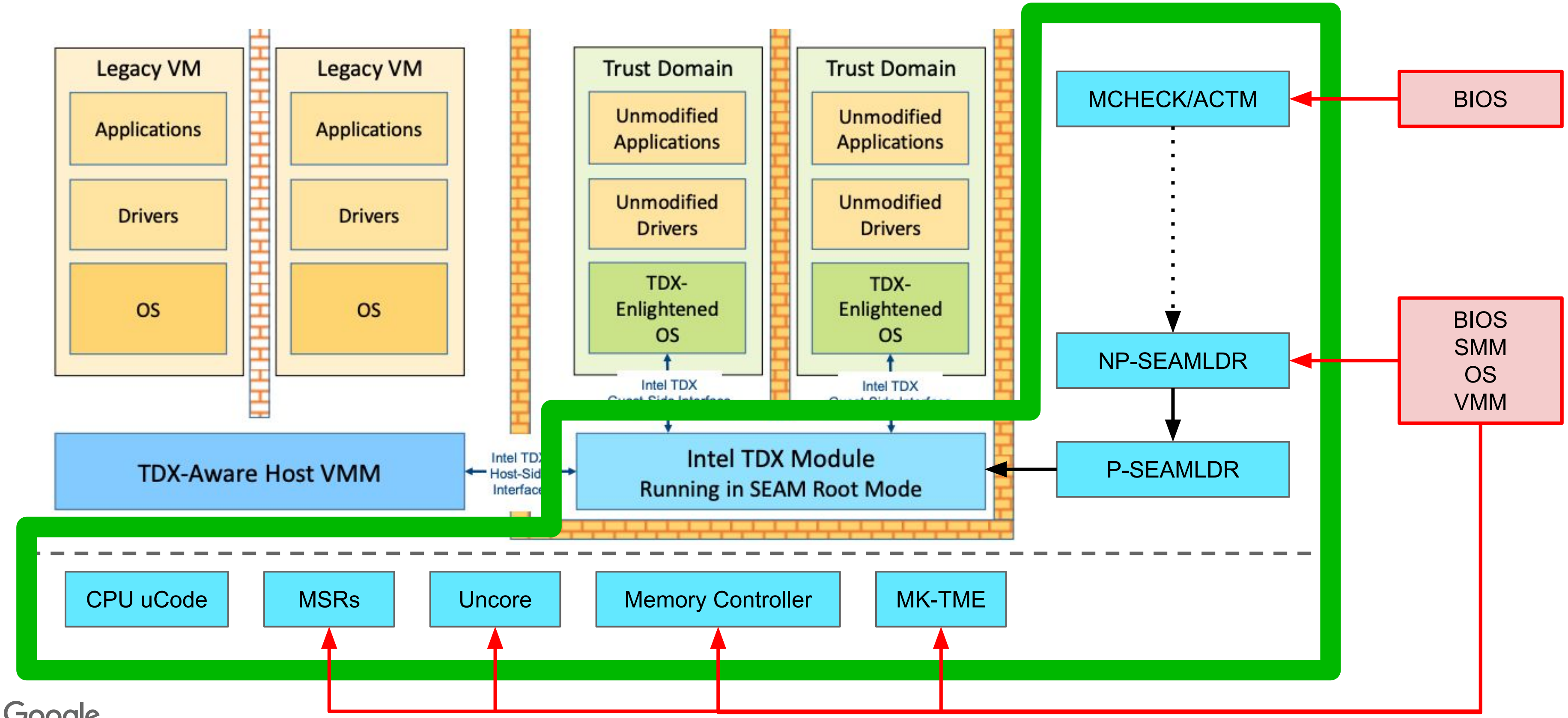
Google

TDX in a Nutshell - Trust Boundary



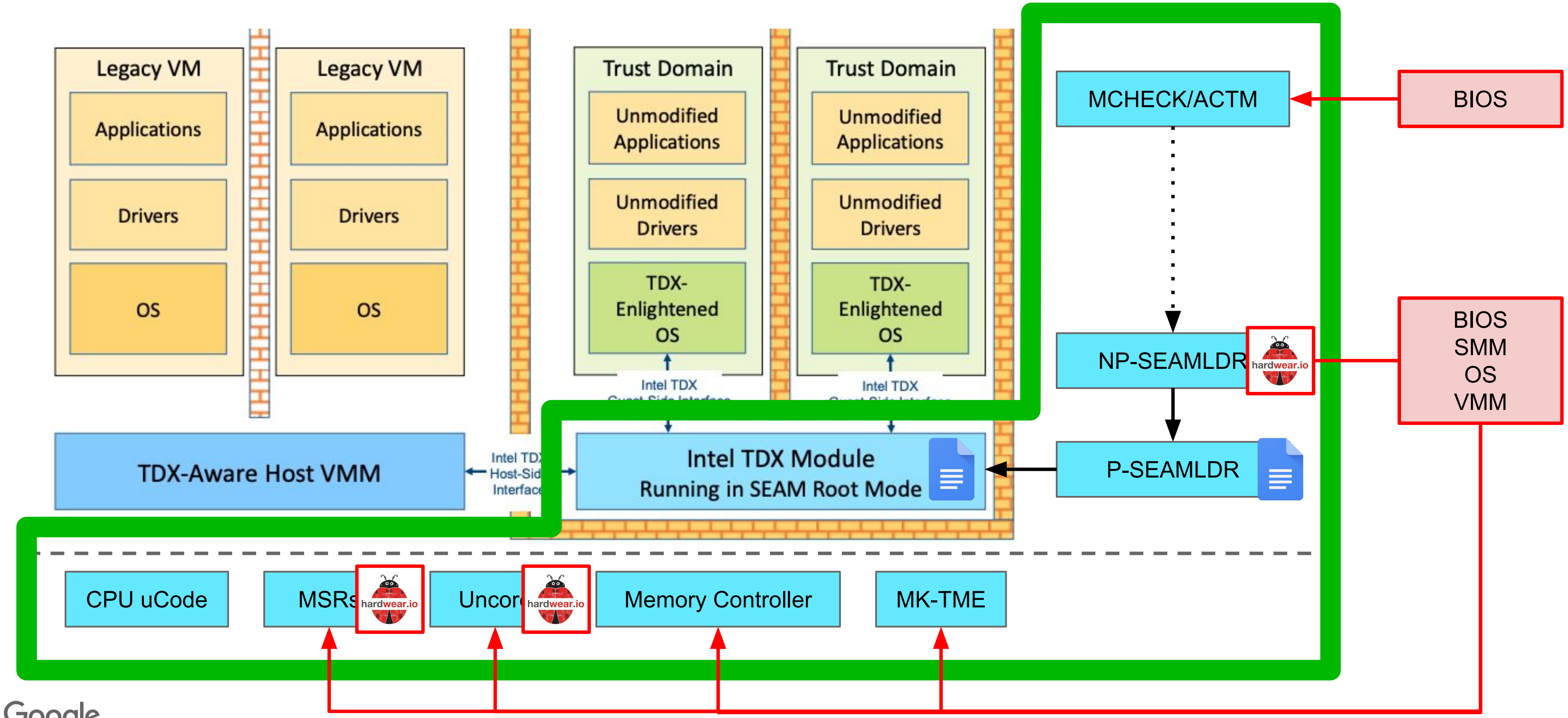
Google

TDX in a Nutshell - Attack Vectors



Google

TDX in a Nutshell - Vulnerabilities Discovered



Google

Security Review Strategy

A comprehensive review of TDX is a large task, we need to focus our resources.

- Review the high level architecture and **changes from legacy VMX**
- Develop an understanding of the different attack vectors available
- Apply a **variety of techniques** to ensure adequate coverage:
  - **Manual source code review**
  - **Test against the hardware implementation** - we relied on simulation as HW wasn't ready
  - **Lessons the SNP review**: security invariant analysis, wycheproof to scan for known crypto bugs

Read the full report for more details and results: Project Zero: Release of a Technical Report into Intel Trust Domain Extensions

TDX Initialization

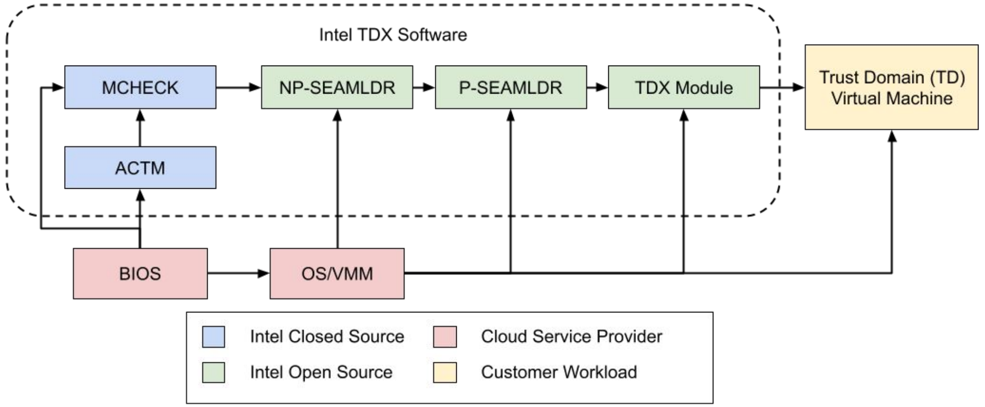
Trust rooted in silicon-fused public keys which verify MCHECK and ACTM module.

- Ensure the foundational system configuration is secure before continuing

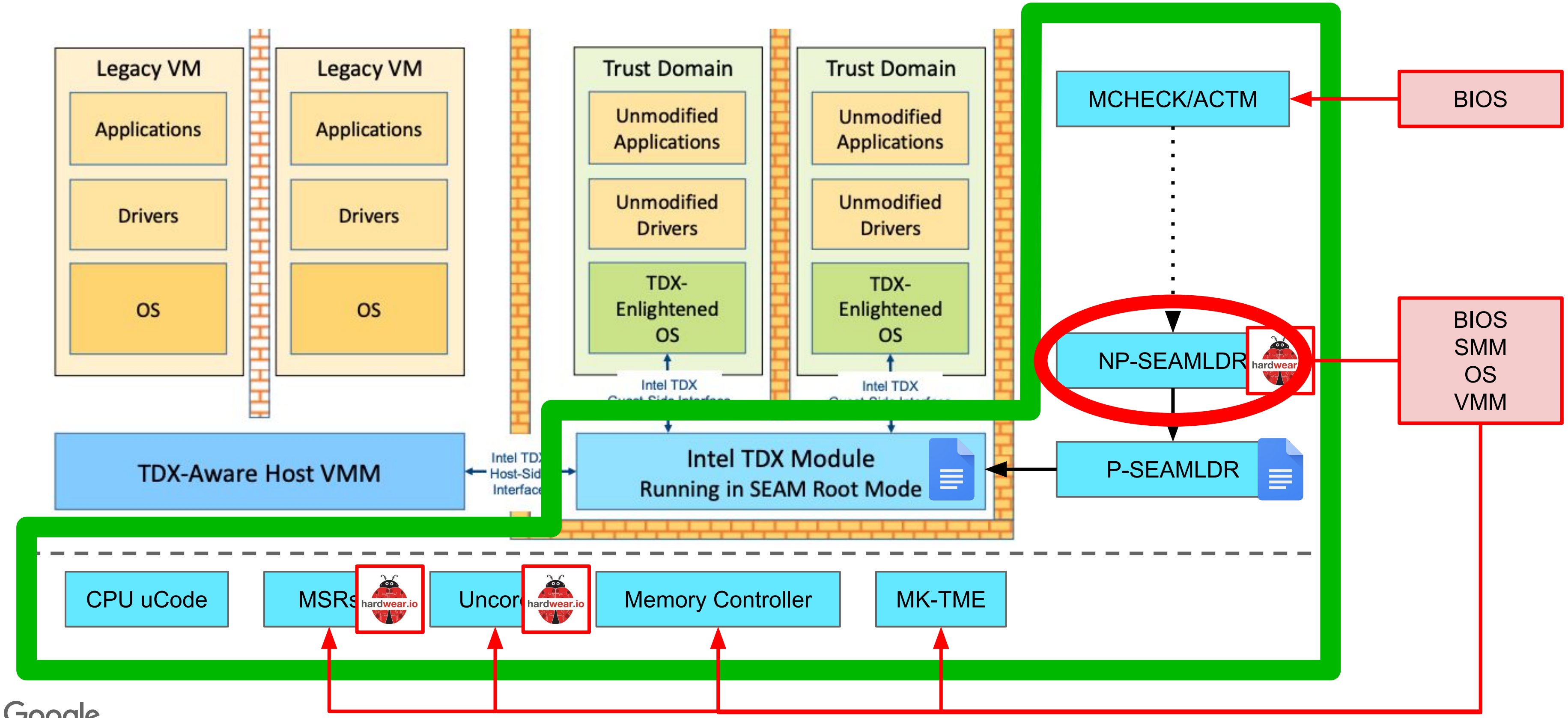
From there, a **verified chain of software** leads to the TDX module.

- Any post-verification compromise along this chain leads to a full system compromise

This chain and metadata are **measured and attested by a key provisioned to the SGX enclave**.



TDX in a Nutshell - NP-SEAMLDR



Google

Nonpersistent (NP)-SEAMLDR

- Part of the TDX boot flow; first stage **loaded by the OS or VMM**
  - Only input related to the CPU state to restore afterward
- Bundled as an Authenticated Code Module (ACM)
  - Legacy tech; code modules signed by Intel and used for system initialization
- **All ACMs run with the same, elevated privileges**. These privileges include:
  - Write access to the register used to measure the TDX boot chain
  - Write access to the SEAMRR protected memory range (TDX module, etc.)
- **Compromise of NP-SEAMLDR breaks the foundation for the rest of TDX**

Overall, it sounds like a fairly small attack surface…

Searching for attack vectors…

- GETSEC[ENTERACCS] **transitions from the host OS execution environment into the ACM** (NP-SEAMLDR)
  - CPU validates RSA signature of the ACM, transitions to 32b mode, and initializes registers
  - ACMs were designed to be called by 32b BIOS code, so **NP-SEAMLDR has new entry/exit paths to help transition between 64b host and 32b ACM**

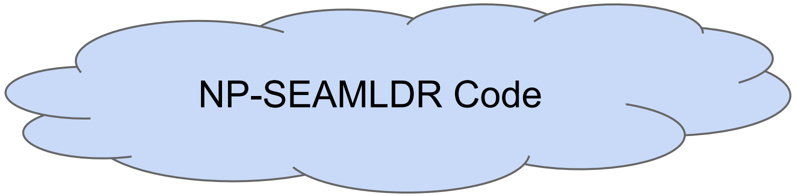
- ACMs run in a **constrained environment** which should negate some attacks
  - SMT is disabled, executes entirely from the cache, **interrupts & exceptions are disabled**

How exactly are interrupts and exceptions disabled?

Interrupts disabled!

```
AcmEntryPoint PROC NEAR
  sidt    fword ptr ds:[ebp + stackStart + 4*6]

  ; Make sure that Null IDTR is actually zero
  mov     dword ptr ds:[ebp + stackStart + 4], 0
  mov     dword ptr ds:[ebp + stackStart + 8], 0
  lidt    fword ptr ds:[ebp + stackStart + 4]    ; Load NULL IDTR
```

```
*(UINT16 *)SeamldrCom64Data.OriginalGdtr = (UINT16)(OriginalECX >> 16);
```

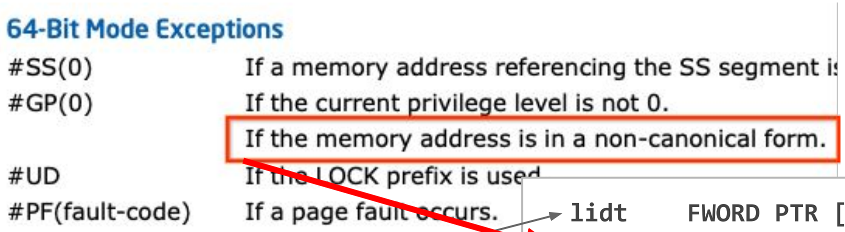


Interrupts re-enabled!

```
  lidt    FWORD PTR [rcx].SEAMLDR_COM64_DATA.NewIDTR    ; Load attacker IDTR
  lgdt    FWORD PTR [rcx].SEAMLDR_COM64_DATA.OriginalGdtr
DoExitAC:
  ; uCode restores the RIP, CR3, and error code from these GPRs
  mov     rbx, QWORD PTR [rcx].SEAMLDR_COM64_DATA.ResumeRip
  mov     r8, QWORD PTR [rcx].SEAMLDR_COM64_DATA.OriginalCR3
  mov     r9, QWORD PTR [rcx].SEAMLDR_COM64_DATA.RetVal

  ; <code truncated for slides>

  GETSEC[EXITAC]  ; drop privileges and return to host x86 code
```

Exit Path Interrupt Hijacking Vulnerability

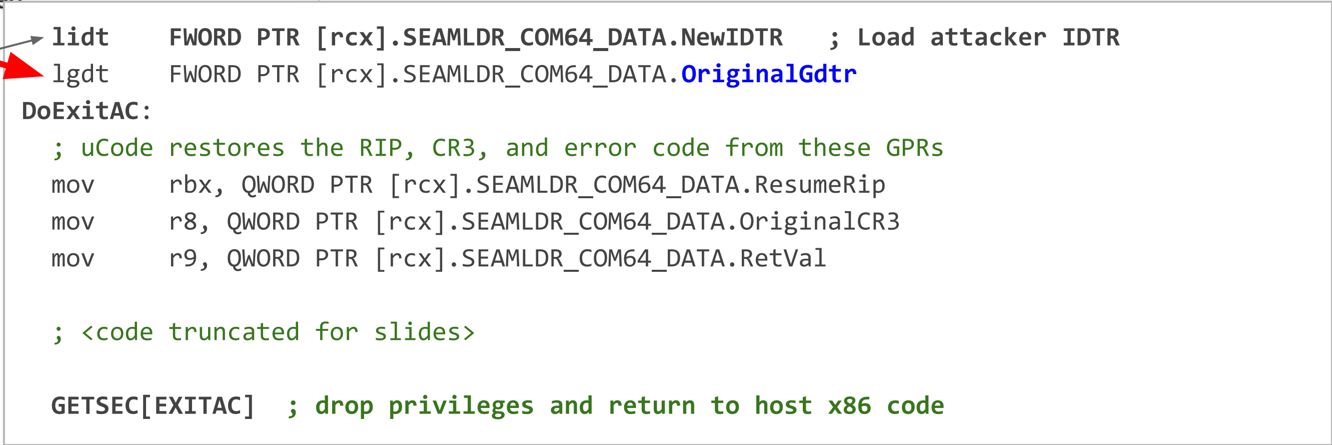
- There is a critical window where interrupts are enabled on ACM entry and exit
- Additionally, the attacker controls the value of IDTR on the exit path
- All of the exit path instructions are exception-safe… except one!



64-Bit Mode Exceptions

#SS(0)	If a memory address referencing the SS segment i
#GP(0)	If the current privilege level is not 0.
	If the memory address is in a non-canonical form.
#UD	If the LOCK prefix is used.
#PF(fault-code)	If a page fault occurs.

Interrupts re-enabled!



```
  lidt    FWORD PTR [rcx].SEAMLDR_COM64_DATA.NewIDTR   ; Load attacker IDTR
  lgdt    FWORD PTR [rcx].SEAMLDR_COM64_DATA.OriginalGdtr
DoExitAC:
  ; uCode restores the RIP, CR3, and error code from these GPRs
  mov     rbx, QWORD PTR [rcx].SEAMLDR_COM64_DATA.ResumeRip
  mov     r8, QWORD PTR [rcx].SEAMLDR_COM64_DATA.OriginalCR3
  mov     r9, QWORD PTR [rcx].SEAMLDR_COM64_DATA.RetVal

  ; <code truncated for slides>

  GETSEC[EXITAC]  ; drop privileges and return to host x86 code
```

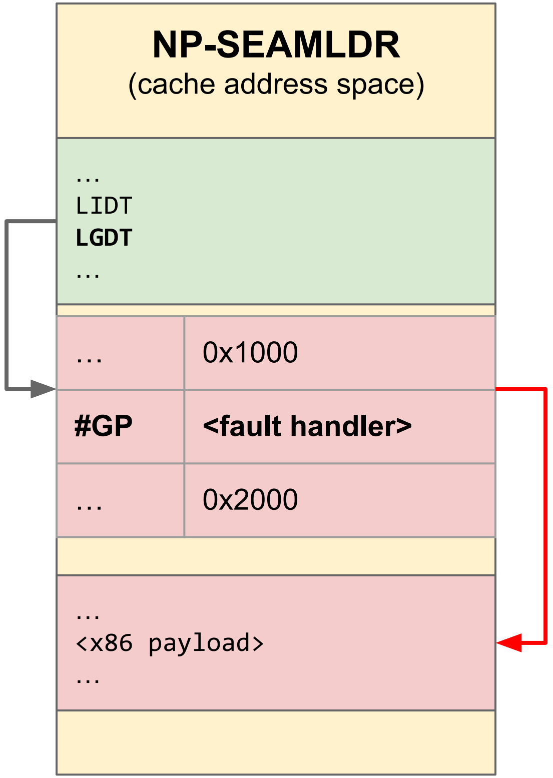
Google

Exploitation

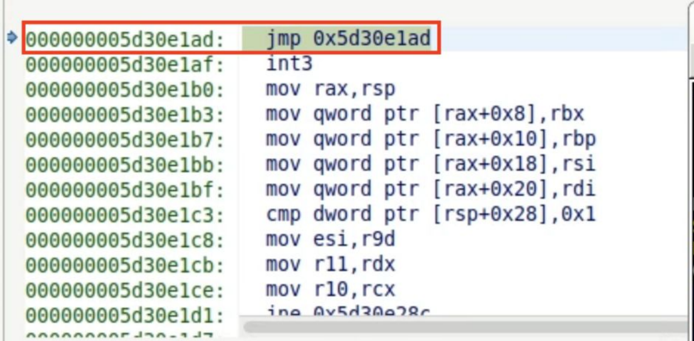
- Since we control the IDTR we can **redirect the #GP to any address**
- However, we're still running from cache and can't access RAM. We need to find a place for our IDT and payload.
- The ACM image is loaded into cache on entry, but its contents are verified through an RSA signature…
- Except for the scratch space!
  - It turns out there's ~800 unverified bytes, of which a subset are used for RSA scratch.
  - Since we didn't have hardware at the time, we proved out the exploit using Intel's Simics simulator which models ACM and NP-SEAMLDR behavior.

Sequence of Events

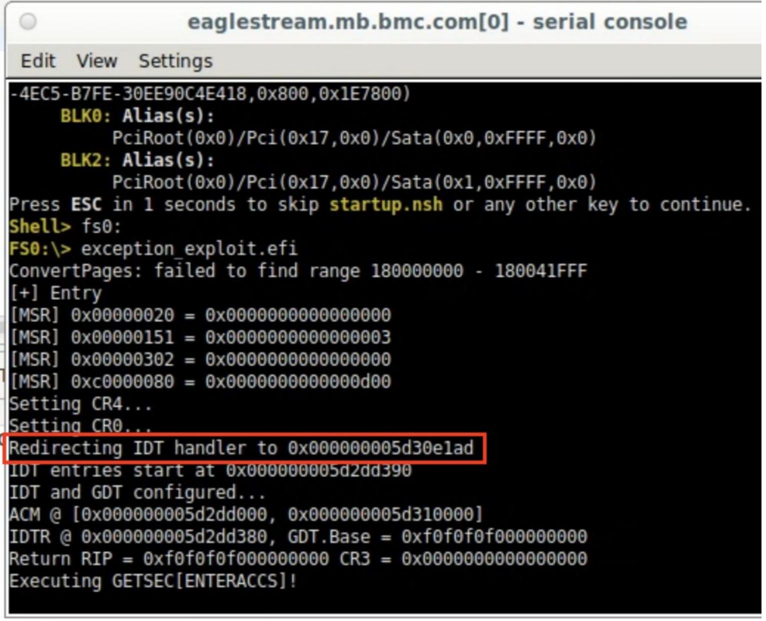
1. Insert fake IDT into ACM binary blob scratch space
    a. Scratch space is not covered by RSA signature
2. OS/VMM loads modified ACM
    a. Pass in invalid arguments for CPU state restoration
    b. IDTR => fake IDT offset in ACM binary
    c. GDTR => non-canonical address
3. ACM runs as usual
4. ACM exit path is reached
    a. Interrupts are enabled
    b. LGDT triggers a #GP



Google

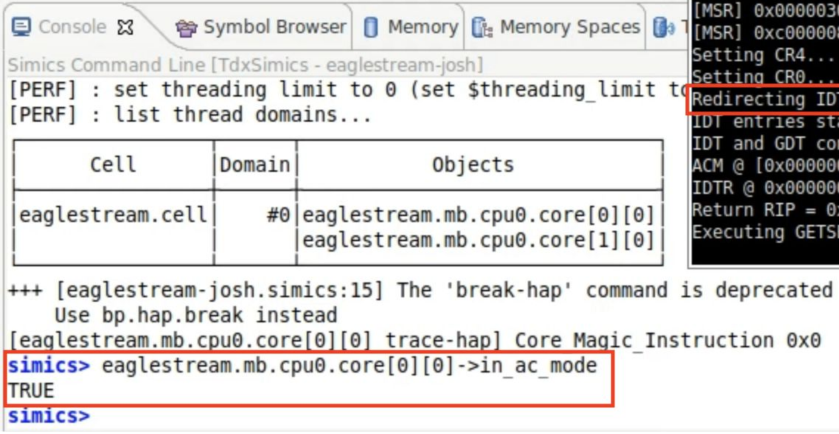


```
000000005d30e1ad:   jmp 0x5d30e1ad
000000005d30e1af:   int3
000000005d30e1b0:   mov rax,rsp
000000005d30e1b3:   mov qword ptr [rax+0x8],rbx
000000005d30e1b7:   mov qword ptr [rax+0x10],rbp
000000005d30e1bb:   mov qword ptr [rax+0x18],rsi
000000005d30e1bf:   mov qword ptr [rax+0x20],rdi
000000005d30e1c3:   cmp dword ptr [rsp+0x28],0x1
000000005d30e1c8:   mov esi,r9d
000000005d30e1cb:   mov r11,rdx
000000005d30e1ce:   mov r10,rcx
000000005d30e1d1:   jne 0x5d30e28c
```



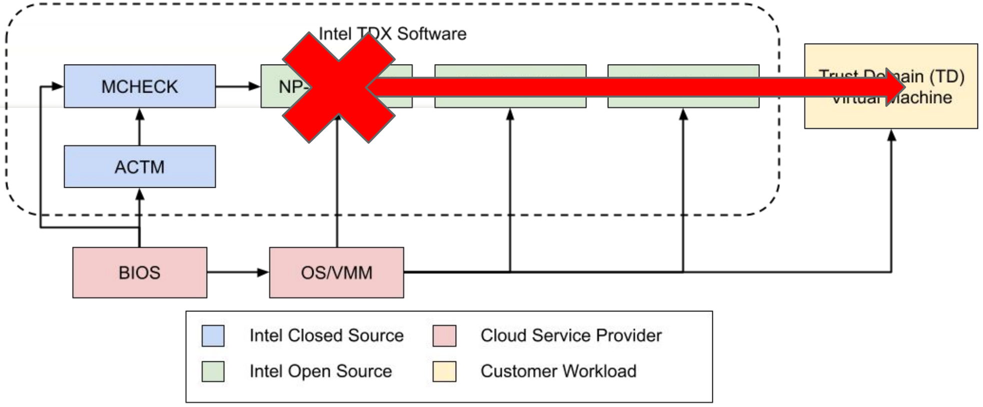
eaglestream.mb.bmc.com[0] - serial console

```
-4EC5-B7FE-30EE90C4E418,0x800,0x1E7800)
     BLK0: Alias(s):
          PciRoot(0x0)/Pci(0x17,0x0)/Sata(0x0,0xFFFF,0x0)
     BLK2: Alias(s):
          PciRoot(0x0)/Pci(0x17,0x0)/Sata(0x1,0xFFFF,0x0)
Press ESC in 1 seconds to skip startup.nsh or any other key to continue.
Shell> fs0:
FS0:\> exception_exploit.efi
ConvertPages: failed to find range 180000000 - 180041FFF
[+] Entry
[MSR] 0x00000020 = 0x0000000000000000
[MSR] 0x00000151 = 0x0000000000000003
[MSR] 0x00000302 = 0x0000000000000000
[MSR] 0xc0000080 = 0x0000000000000d00
Setting CR4...
Setting CR0...
Redirecting IDT handler to 0x000000005d30e1ad
IDT entries start at 0x000000005d2dd390
IDT and GDT configured...
ACM @ [0x000000005d2dd000, 0x000000005d310000]
IDTR @ 0x000000005d2dd380, GDT.Base = 0xf0f0f0f000000000
Return RIP = 0xf0f0f0f000000000 CR3 = 0x0000000000000000
Executing GETSEC[ENTERACCS]!
```



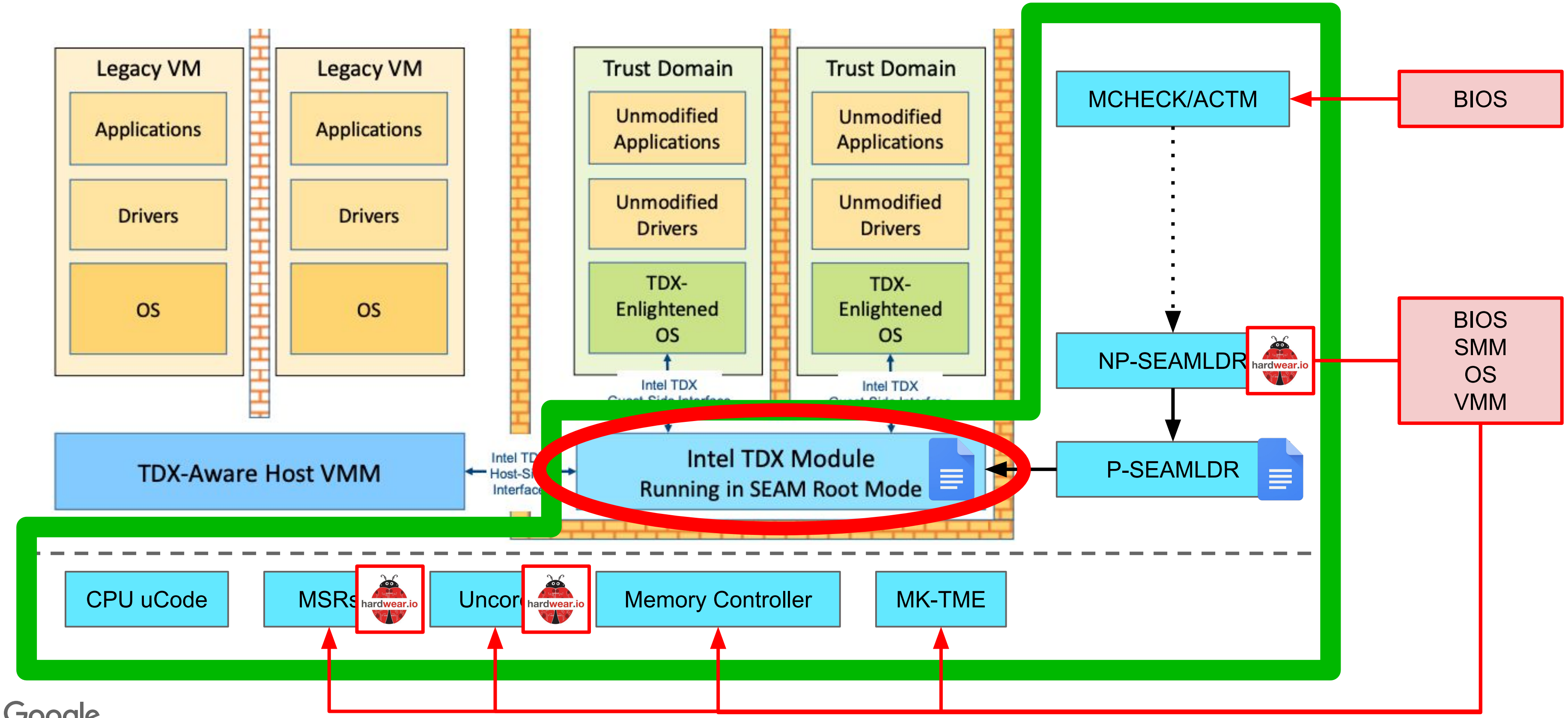
Simics Command Line [TdxSimics - eaglestream-josh]

```
[PERF] : set threading limit to 0 (set $threading_limit to
[PERF] : list thread domains...

 ┌───────────────┬──────┬──────────────────────────┐
 │     Cell      │Domain│         Objects          │
 ├───────────────┼──────┼──────────────────────────┤
 │eaglestream.cell│   #0│eaglestream.mb.cpu0.core[0][0]│
 │               │      │eaglestream.mb.cpu0.core[1][0]│
 └───────────────┴──────┴──────────────────────────┘

+++ [eaglestream-josh.simics:15] The 'break-hap' command is deprecated
    Use bp.hap.break instead
[eaglestream.mb.cpu0.core[0][0] trace-hap] Core Magic_Instruction 0x0
simics> eaglestream.mb.cpu0.core[0][0]->in_ac_mode
TRUE
simics>
```



Google

TDX in a Nutshell - Vulnerabilities Discovered



Google

TDX Module

This persistent module is **responsible for TD management** throughout their lifecycle. It runs in elevated SEAM mode, compromise of this code leads to compromise of TDX on the system.

Two APIs:

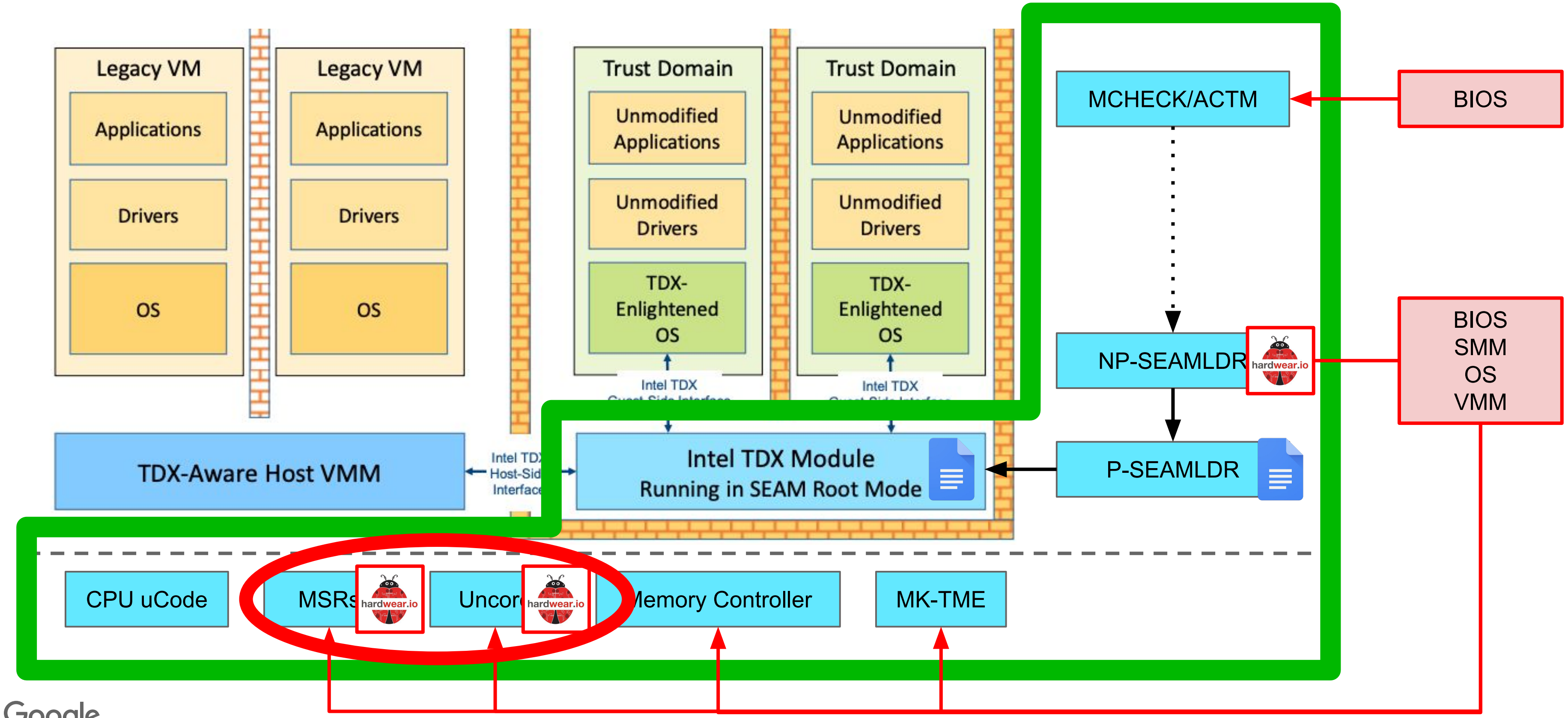
- **9 APIs available directly to a TD** (new TDCALL instruction)
- **44 APIs available to the host** (new SEAMCALL instruction)

Review strategy:

- Large focus on manual code review throughout the team
- Additionally used wycheproof for crypto library (Intel IPP) validation, weggli for variant analysis, and Frama-C for limited static analysis

4 bugs discovered and fixed, details in the full report

TDX in a Nutshell - Vulnerabilities Discovered



Google

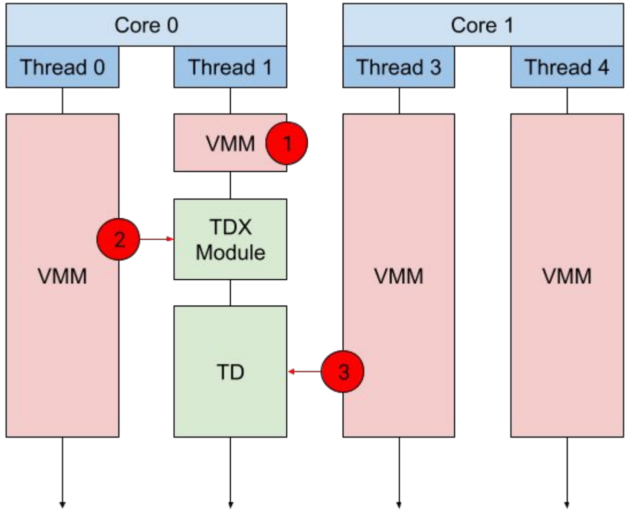
MSR Attack Vector

A malicious host, SMM, or BIOS has a large selection of control registers which may weaken system security.

We focused on per-core and per-platform MSRs:

- The TDX module can't prevent their modification
- Affects TDX module and TDs during runtime

Overall, there is a **privilege inversion** with MSRs and TDX. The privileged TDX code can't trap or limit MSR values set by the untrusted host.



Google

TDX Private Memory Protections

MK-TME **encrypts and protects** the TDX private memory contents

- Plaintext in cache, ciphertext + integrity-check in DRAM
- AES encryption, 128-bit blocks

Two modes CPU uses to verify private memory is private

- Cryptographic (TDX-CI): 28-bit HMAC stored beside the content in RAM
- Logical (TDX-LI): 1-bit indicator alternative
- Mode selection based on DDR5 capacity characteristics

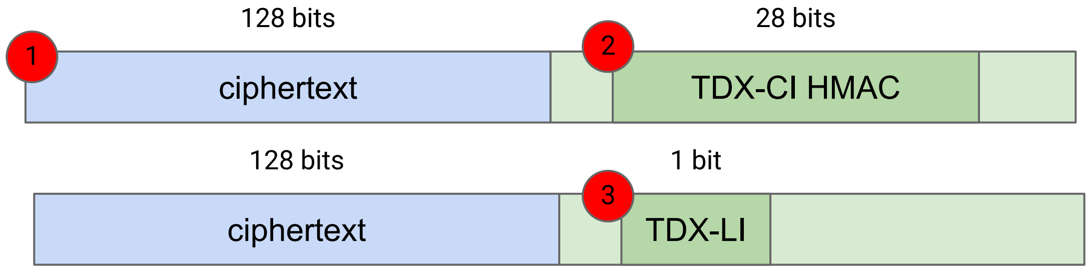
Breaking the integrity check allows an attacker with DRAM control to **create false private memory**.

What about Rowhammer?

Breaking the integrity check allows an attacker with DRAM control to **create false private memory**.

Targeting options:

1. **Ciphertext** - essentially randomize 128 bits; invalidate integrity check
2. **TDX-CI HMAC** - need to brute force 28-bit HMAC but only 1 shot
3. **TDX-LI** - only need to flip a single bit



Does ECC mitigate this attack on TDX-LI?

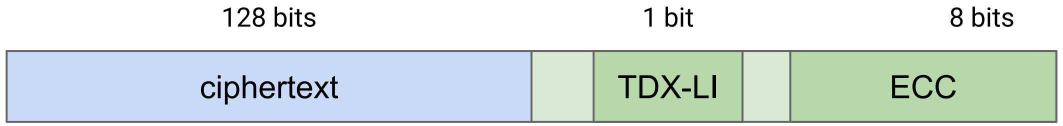
ECC significantly raises the bar for successful silent data corruption.

However, **ECC must first be configured by the BIOS** via the Uncore registers

Who verifies that ECC is enabled before TDX-LI is allowed?

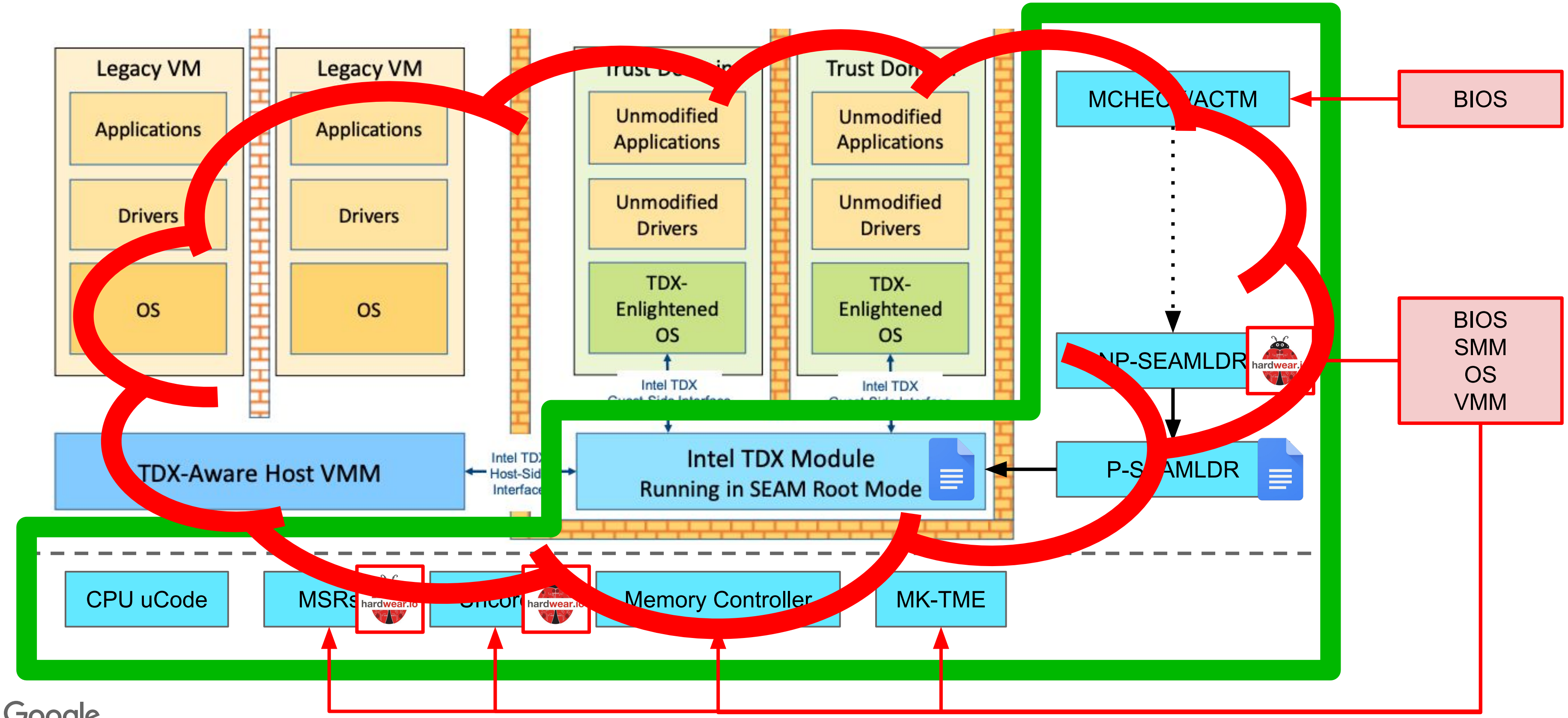
It turns out nobody, in the pre-release version! A malicious BIOS with Rowhammer primitives could create false private memory by disabling ECC.

- Intel fixed the bug before public release by securely validating ECC settings.



Google

TDX in a Nutshell - Vulnerabilities Discovered



Google

Side Channel Attack Vectors

We focused on three categories of SCAs:

- **Transient execution-based**
  - TDX software uses best practices re: HW/SW mitigations
  - Manually identified critical points insert speculation barriers and flush branch predictors
- **Traditional timing-based**
  - Hardened crypto library used (additionally verified with Wycheproof test suite)
  - RAPL energy reporting feature limits power analysis granularity
- **Access oracles**
  - Host-visible effects that disclose sensitive TD behavior

Side Channels - Access Oracles

Witnessing **TD memory access patterns can reveal sensitive info** depending on the workload

Oracle Primitive	Side Effects	Address Resolution
TDH.MEM.RANGE.BLOCK	Guest fault (recoverable)	Page-level
Poisoning TD cache lines	Guest termination	Cacheline-level
MONITOR/MWAIT (up to ~40 in parallel)	None	Cacheline-level

Confidential compute users should **utilize constant-access operations** for sensitive workloads.

TDX Security Review Summary

All discovered software bugs were remediated by Intel before the public launch!

- Vulnerability impact ranged from minor to full TDX compromise

Source code and many specifications are available online

- Spec & source: Intel® Trust Domain Extensions (Intel® TDX)
- Intel bug bounty: Intel® Bug Bounty Program Terms

However, TDX is only going to get more complex - **there is more work to be done!**

- TDX 1.5 (live migration) and 2.0 (TEE-IO) specifications published

Check the full report for more details on what we didn’t cover today.

Conclusions

The State of Confidential Compute Vulnerability Research

CC de-privileges the host OS/VMM - **significantly raising the cost of host=>VM attacks**.

We **encourage hardware vendors to embrace collaboration with security researchers**

- Users must trust the CPU security claims before staking their secrets on it.
- Open sourcing confidential compute firmware will build more confidence in the robustness of the design and implementation.

Side channel attacks take an elevated importance in systems touted as completely opaque - even from the most sophisticated attackers.

Due to the complexity and amount of new technology, **we'll likely see a long tail of issues**.

Confidential VMs may become ubiquitous as new hardware saturates the server market.