Behind the Scenes of Intel Security and Manageability Engine

Shai Hasarfaty
Principal Security Research Engineer, Intel Corp.
Yanai Moyal
Security Researcher, Intel Corp.
Intel provides these materials as-is, with no express or implied warranties.

All products, dates, and figures specified are preliminary, based on current expectations, and are subject to change without notice.

Intel, processors, chipsets, and desktop boards may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No product or component can be absolutely secure. Check with your system manufacturer or retailer or learn more at https://www.intel.com.

Some results have been estimated or simulated using internal Intel analysis or architecture simulation or modeling, and provided to you for informational purposes. Any differences in your system hardware, software or configuration may affect your actual performance.

Intel and the Intel logo are trademarks of Intel Corporation in the United States and other countries.

*Other names and brands may be claimed as the property of others.

© Intel Corporation
Scope Of This Talk

Latest CSME 12 Firmware & Hardware on Intel 8th and 9th Gen Core Processor based platforms (code name Coffee Lake and Whiskey Lake)
• Architecture & Boot flow
• OS Security Principles & Internals
• Hardening & Mitigations
• Pre & Post Manufacturing
• Update & Recoverability
• Wrap-up
What Is CSME?

CSME is an embedded subsystem in Platform Controller Hub (PCH)
- Stands for Converged Security & Manageability Engine
- Standalone low power Intel processor with dedicated Hardware (HW)
- CSME is Root of Trust of the platform
  - Provides an isolated execution environment protected from host SW running on main CPU
  -Executes CSME Firmware (FW)
What Is CSME?

CSME serves 3 main platform roles

- **Chassis**
  - Secure boot of the platform
  - Overclocking
  - Micro-code loading into PCH/CPU HW engines

- **Security**
  - Isolated & trusted execution of security services (TPM, DRM, DAL)

- **Manageability**
  - Platform management over out of band network (Intel AMT)
• **CPU**: Intel 32 bits processor (i486) supporting rings, segmentation and MMU for page management

• **SRAM**: Isolated RAM (~1.5 MB) from host

• **ROM**: HW root of trust of CSME Firmware

• **System Agent**: Allows CPU to securely access SRAM and enforce access control to SRAM from internal/external devices by using IOMMU (i.e. control DMA access)

• **OCS (Offload & Cryptography Subsystem)**: Crypto HW accelerator with DMA engine and Secure Key Storage (SKS)

• **Gasket**: interface to PCH fabric & CSME IO devices (TPM, HECI etc.)
• DRNG
  • Generates non-deterministic random numbers
  • Compliant to NIST SP800-90A, B and C

• Fuses – 2 types
  • Intel PCH Manufacturing Fuses – set by Intel before shipment to manufacturers
    • CSME configurations: which Intel CSME signing keys enabled, production silicon etc.
    • CSME security keys unique per chip and encrypted using CSME HW key
  • In Field Programmable Fuses (FPF) – set by manufacturers before shipment to end-users
    • Manufacturers’ secure settings: public key, Intel Boot Guard policy etc.
    • CSME FW Anti-Rollback Security Version Number (ARB SVN)

• DFX (debug)
  • Control CSME & other PCH micro-controllers debug interface (JTAG)
  • In debug (JTAG open), keys in fuses and secrets in NVM are not available & CSME SRAM is zeroed
CSME Role In Platform Boot

Legend:
- Load and authenticate
- Measured Boot w/ PTT
- Intel Boot Guard support

Host (CPU):
- PMC start
- Host out of reset
- DRAM Init Done
- BIOS/OS handoff

CSME:
- ROM
- RBE
- TCB OS
- Bringup (includes BTG support & PTT)

Drivers, Services and Applications:
- Sensors
- Audio
- Camera

IPs:
- PMC ROM
- PMC patch/data

Time
CSME ROM

- ROM is part of PCH HW with no patch mechanism after HW tape-in
  - ROM bypass disabled by Intel manufacturing fuse on production stepping
- Main responsibilities
  - Moves CSME CPU to protected mode & enable paging and segmentation
  - Generates CSME FW keys using chipset key and RBE Security Version Number (TCB SVN)
  - Loads, authenticates and executes IDLM (debug module) / RBE
  - Hashes of public keys embedded in ROM
  - Intel manufacturing fuse indicates which public key is enabled (debug signing key is disabled on production)
Key Derivation By ROM

1. HW key generation
2. Decryption of Chipset Key
3. Derivation of CSME FW Key
HW Secure Key Storage

**HW SKS:**
Protect CSME root keys during runtime. FW can only use keys.

Every SKS slot has set of attributes

- **Secure Mode**
  - Result of AES-CBC decrypt and HMAC using the key in this slot can be stored in SKS only

- **Privilege Level:** used for HW access control on SKS slot
  - The key in this slot is accessible if SKS slot privilege level is $\geq$ SKS privilege level
HW SKS:
Protect CSME root keys during runtime. FW can only use keys
Every SKS slot has set of attributes
- Secure Mode
  - Result of AES-CBC decrypt and HMAC using the key in this slot can be stored in SKS only
- Privilege Level: used for HW access control on SKS slot
  - The key in this slot is accessible if SKS slot privilege level is >= SKS privilege level
- Locked: key in this slot can be invalidated or replaced after CSME HW reset only
RBE (ROM Boot Extension)

- Extends ROM functionality in FW (can be updated on field)
- Bootloader of CSME OS
- Main responsibilities
  - Performs HW based anti-rollback check on CSME FW
  - Performs early chassis job – PMC patch
  - Loads, authenticates and executes CSME OS
Once all CSME modules have been loaded, Process Manager stores all CSME modules’ ICVs and ICV key in ICV Blob Partition (IVBP) in SPI flash, encrypted, integrity and replay protected.
CSME Secure Boot Flow With ICV Blob

1. Load, authenticate and execute
2. Load and authenticate
3. Execute using ICV
4. Execute using ICV
5. Execute using ICV
6. Execute using ICV
7. Execute using ICV

Legend
ICV: Integrity Check Value

- FW Ring0
- FW TCB Ring3
- HW Ring0
- FW OS Ring3
- FW App Ring3
- FW lib. Ring3

ICVs are used to speed-up CSME boot

OS & core drivers and services

- Kernel
- Syslib
- Process Manager
- VFS
- Crypto Driver
- Event Disp.
- Bringup
- Load Manager
- Storage Driver
- Bus Driver
- PRTC Driver
- FPF Driver

Driver & Services

- CSME Services
- CSME Drivers

Applications

- AMT
- PTT
- DAL
- PAVP
- ICC
- IP Loading

Code pages replacement using ICV done by kernel from SPI flash
Code and data pages replacement using ICV done by kernel from DRAM

Legend
- FW Ring0
- FW TCB Ring3
- HW Ring0
- FW OS Ring3
- FW App Ring3
- FW lib. Ring3
• Architecture & Boot flow
• OS Security Principles & Internals
• Hardening & Mitigations
• Pre & Post Manufacturing
• Update & Recoverability
• Wrap-up
CSME OS Main Security Principles

- Micro-Kernel OS based on Minix OS architecture
  - The micro-kernel is the only runtime component running at ring0. Application, Drivers and Services run at ring3
  - The micro-kernel implements the bare minimum required to implement an OS

- Minimal Trusted Compute Base (TCB)
  - Protects access to keys and HW (CSME assets)
  - Responsible for CSME FW code integrity at boot & runtime
  - Responsible for protection CSME modules from each other and their data in SPI flash
  - Enforces CSME modules’ minimum privileges
• Main responsibilities
  • Driver of the CPU
    • Enforces code execution is from SRAM only
    • Enforces process isolation using CPU rings and x86 segments
    • Sets up page attributes (RW and User bit). Enforcement done by MMU
    • Controls HW access via MMIO by ring3
  • Driver of the IOMMU
    • Controls DMA access to SRAM
  • Support standard kernel service
    • Inter-Process Communication (IPC)
    • Processes & threads management
    • Interrupts and exceptions handling
  • Handle page replacement between SRAM and DRAM/SPI flash to save SRAM utilization
    • Evicted pages to DRAM are encrypted and integrity protected
## CSME TCB OS

### Ring3 TCB Component

<table>
<thead>
<tr>
<th>Component</th>
<th>Main Security Role</th>
<th>Create CSME Process</th>
<th>Control access to CSME keys</th>
<th>Control access to CSME OS services</th>
<th>Control access to Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process Manager</td>
<td>1. Code authentication 2. Process creation &amp; termination</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Crypto Driver</td>
<td>1. Crypto &amp; DMA service 2. FW key management</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Virtual File System (VFS)</td>
<td>1. Secure storage service with data migration support 2. Enforce permission check on data files and special files exposed by drivers &amp; services</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Bus Driver</td>
<td>1. Allow CSME drivers to configure their own device configuration space 2. Enforce access control</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Minimal</td>
</tr>
</tbody>
</table>

### Diagram:

- **Applications**
- **Services**
- **Drivers**
- **Bringup**
- **TCB OS**
- **uKernel**
- **RBE**
- **ROM**
- **Flash**
- **Boot flow**
- **Ring 0**
- **Ring 3**
Bringup (BUP)

- Support early platform boot and configuration

- Reduced its privileges starting CSME 12
  - Can’t create CSME processes
  - No access to root keys and attestation keys
  - No access to crypto accelerator and DFX HW
• Driver and Services are running at ring 3
• Drivers have access only to HW they need to manage via Memory Mapped IO (MMIO)
• Access to drivers and services are jointly controlled by VFS & uKernel
• CSME applications are running at ring 3
• CSME TCB ensure CSME applications are isolated from each others including their data kept in NVM
• Architecture & Boot flow
• OS Security Principles & Internals
• Hardening & Mitigations
• Pre & Post Manufacturing
• Update & Recoverability
• Wrap-up
Hardening & Exploitation Mitigations

• Kernel (Ring0)
  • Kernel system call filtering
  • Applied stack protector for kernel
  • Data execution prevention
  • Activated Supervisor Mode Execution Prevention (SMEP)
  • Use CR0.Write.Protect
    • Prevent corruption of read only pages by kernel
  • ACL on Ring3 inter-process communication
Hardening & Exploitation Mitigations

- Example of authorized IPC (Abstracted)

<table>
<thead>
<tr>
<th>PAVP - DRM</th>
<th>VFS</th>
<th>CryptoDrv</th>
<th>Kernel</th>
</tr>
</thead>
</table>

#BHUSA @BLACKHATEVENTS
• Example of authorized IPC (Abstracted)

open() - IPC_SEND_RCV

PAVP - DRM  |  VFS  |  Crypto Drv  |  Kernel
• Example of authorized IPC (Abstracted)

```plaintext
PAVP - DRM
VFS
CryptoDrv
Kernel

open() - IPC_SEND_RCV

VERIFY_ACL

WRITE_BUFFER
RETURN_SUCCESS
RETURN_SUCCESS
```
• Example of authorized IPC (Abstracted)
• Example of authorized IPC (Abstracted)

```
open() - IPC_SEND_RCV
VERIFI_REQ_ACL
FILE_DESCRIPTOR_HANDLE
VERIFI_ACL
IOCTL_AES_ENC
IOCTL_CHECK_ACL
WRITE BUFFER
RETURN_SUCCESS
RETURN_SUCCESS
```
• Example of authorized IPC (Abstracted)

```
PAVP - DRM  VFS  Crypto Drv  Kernel

open() - IPC_SEND_SEND

VERIFY_ACL

FILE_DESCRIPTOR_HANDLE

FILE_DESCRIPTOR_HANDLE

VERIFY_REQ_ACL

IOCTL_AES_ENC

IOCTL_CHECK_ACL

WRITE BUFFER

RETURN_SUCCESS

RETURN_SUCCESS
```
• Example of authorized IPC (Abstracted)
• Example of authorized IPC (Abstracted)

```
open() - IPC_SEND_RECV

VERIFI_REQ_ACL

FILE_DESCRIPTOR_HANDLE

IOCTL(FD, AES_ENC, &BUF)

IOCTL_CHECK_ACL
```

Diagram:
- PAVP - DRM
- VFS
- CryptoDrv
- Kernel

- `open()` - IPC_SEND_RECV
- `VERIFI_REQ_ACL`
- `FILE_DESCRIPTOR_HANDLE`
- `IOCTL(FD, AES_ENC, &BUF)`
- `IOCTL_CHECK_ACL`
• Example of authorized IPC (Abstracted)
• Example of authorized IPC (Abstracted)

```
open() - IPC_SEND_RCV
    |        |
    V

FILE_DESCRIPTOR_HANDLE

FILE_DESCRIPTOR_HANDLE

FILE_DESCRIPTOR_HANDLE

ioctl( FD , AES_ENC , &BUF )

ioctl AES_ENC

WRITE BUFFER

ioctl_CHECK_ACL

RETURN_SUCCESS

RETURN_SUCCESS
```

```
PAVP - DRM
VFS
Crypto Drv
Kernel
```
• Example of authorized IPC (Abstracted)

```
open() - IPC_SEND_RCV

VERIFI_REQ_ACL

VERIFI_ACL

FILE_DESCRIPTOR_HANDLE

FILE_DESCRIPTOR_HANDLE

IOCTL (FD, AES_ENC, &BUF)

IOCTL AES_ENC

IOCTL_CHECK_ACL

WRITE BUFFER

RETURN_SUCCESS

RETURN_SUCCESS
```
• Example of unauthorized IPC

```
open() - IPC_SEND_RCV

VERIFY_REQ_ACL

VERIFY_ACL

IOCTL AES_ENC

IOCTL_CHECK_ACL

WRITE BUFFER

RETURN_SUCCESS

RETURN_SUCCESS
```
• Exploitation mitigations in Ring3
• Exploitation mitigations in Ring3
  • Syslib context pointer moved to a read only page (not on stack anymore)
Hardening & Exploitation Mitigations

- Exploitation mitigations in Ring3
  - Syslib context pointer moved to a read only page (not on stack anymore)
  - Return Control Flow Integrity via modified Stack Canary “XOR-RET-ALL” on the majority of the Ring3 functions.
• Exploitation mitigations in Ring3
  • Syslib context pointer moved to a read only page (not on stack anymore)
  • Return Control Flow Integrity via modified Stack Canary “XOR-RET-ALL” on the majority of the Ring3 functions.
Exploitation mitigations in Ring3

- Syslib context pointer moved to a read only page (not on stack anymore)
- Return Control Flow Integrity via modified Stack Canary “XOR-RET-ALL” on the majority of the Ring3 functions.
Exploitation mitigations in Ring3

- Syslib context pointer moved to a read only page (not on stack anymore)
- Return Control Flow Integrity via modified Stack Canary “XOR-RET-ALL” on the majority of the Ring3 functions.

Regular Stack-Protector:

<table>
<thead>
<tr>
<th>LOCAL VAR</th>
<th>Canary</th>
<th>OLD EBP</th>
<th>RETURN</th>
<th>ARG 1</th>
<th>ARG 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Buffer overflow</td>
<td>Random</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Exploitation mitigations in Ring3
- Syslib context pointer moved to a read only page (not on stack anymore)
- Return Control Flow Integrity via modified Stack Canary “XOR-RET-ALL” on the majority of the Ring3 functions.

Regular Stack-Protector:
• Exploitation mitigations in Ring3
  • Syslib context pointer moved to a read only page (not on stack anymore)
  • Return Control Flow Integrity via modified Stack Canary “XOR-RET-ALL” on the majority of the Ring3 functions.

• Stack-Protector XORed with Return address:

```
LOCAL VAR | Canary = Random ^ Ret | OLD EBP | RETURN | ARG 1 | ARG 2
```
Hardening & Exploitation Mitigations

- Exploitation mitigations in Ring3
  - Syslib context pointer moved to a read only page (not on stack anymore)
  - Return Control Flow Integrity via modified Stack Canary “XOR-RET-ALL” on the majority of the Ring3 functions.

- Stack-Protector XORed with Return address:

- Attacker will now require to have stack/canary info leak or to leverage a data corruption (if possible)
• Exploitation mitigations in Ring3
  • Syslib context pointer moved to a read only page (not on stack anymore)
  • Return Control Flow Integrity via modified Stack Canary “XOR-RET-ALL” on the majority of the Ring3 functions.
  • SW Forward Edge Control Flow Integrity
Exploitation mitigations in Ring3
- Syslib context pointer moved to a read only page (not on stack anymore)
- Return Control Flow Integrity via modified Stack Canary “XOR-RET-ALL” on the majority of the Ring3 functions.
- SW Forward Edge Control Flow Integrity
• Exploitation mitigations in Ring3
  • Syslib context pointer moved to a read only page (not on stack anymore)
  • Return Control Flow Integrity via modified Stack Canary “XOR-RET-ALL” on the majority of the Ring3 functions.
  • SW Forward Edge Control Flow Integrity
Exploitation mitigations in Ring3
- Syslib context pointer moved to a read only page (not on stack anymore)
- Return Control Flow Integrity via modified Stack Canary “XOR-RET-ALL” on the majority of the Ring3 functions.
- SW Forward Edge Control Flow Integrity
Exploitation mitigations in Ring3
- Syslib context pointer moved to a read only page (not on stack anymore)
- Return Control Flow Integrity via modified Stack Canary “XOR-RET-ALL” on the majority of the Ring3 functions.

SW Forward Edge Control Flow Integrity
• Exploitation mitigations in Ring3
  • Syslib context pointer moved to a read only page (not on stack anymore)
  • Return Control Flow Integrity via modified Stack Canary “XOR-RET-ALL” on the majority of the Ring3 functions.
  • SW Forward Edge Control Flow Integrity
Exploitation mitigations in Ring3

- Syslib context pointer moved to a read only page (not on stack anymore)
- Return Control Flow Integrity via modified Stack Canary “XOR-RET-ALL” on the majority of the Ring3 functions.
- SW Forward Edge Control Flow Integrity

```assembly
mov eax, [ebp + CBFuncPtr]
mov ecx, eax
mov ecx, [ecx - 4]
dec ecx
#compare with endbr32-1 to create "cmp" without endbr32 bytecode
cmp ecx, 0FB1E0FP2h  #endbr32 = f3 0f 1e fb
jz .L_valid_function_pointer
__CFI_FAIL__
.L_valid_function_pointer:
call eax
```
• Exploitation mitigations in Ring3
  • Syslib context pointer moved to a read only page (not on stack anymore)
  • Return Control Flow Integrity via modified Stack Canary “XOR-RET-ALL” on the majority of the Ring3 functions.
  • SW Forward Edge Control Flow Integrity
  • Heap protections
Hardening & Exploitation Mitigations

• Exploitation mitigations in Ring3
  • Syslib context pointer moved to a read only page (not on stack anymore)
  • Return Control Flow Integrity via modified Stack Canary “XOR-RET-ALL” on the majority of the Ring3 functions.
  • SW Forward Edge Control Flow Integrity
• Heap protections
  • Double free protection
  • Malloc of zero size return NULL
  • Cookie protection enforced during free of an allocated busy chunk
    • A Marker surround every Busy Block
    • Value of random “Cookie” field in the Marker is compared with the original Cookie value. Mismatch is handled as an overflow attack (or bug).
• Exploitation mitigations in Ring3
  • Syslib context pointer moved to a read only page (not on stack anymore)
  • Return Control Flow Integrity via modified Stack Canary “XOR-RET-ALL” on the majority of the Ring3 functions.
  • SW Forward Edge Control Flow Integrity
  • Heap protections
  • Data execution prevention
Applying Security Development Lifecycle (SDL) through the CSME development phases

• Security Architecture and Design Review
  • Threat analysis
  • Challenging FW design results into product changes

• Security Code Review
  • Manual and Static Code Analysis (SCA) tools

• Penetration testing
  • Manual
  • Automation
Using latest industry techniques on silicon
• Using latest industry techniques on silicon
  • Address Sanitization
• Using latest industry techniques on silicon
  • Address Sanitization
    • Doesn’t work out of the box since requires glibc
    • Sanitizer requires 8 bytes aligned memory and CSME a 4 bytes aligned memory
    • Sanitizer write to a “shadow” memory at a fixed address of “0x20000000”
    • Making the code too big won’t fit into flash or won’t fit into SRAM
• Using latest industry techniques on silicon
  • Address Sanitization
    • Doesn’t work out of the box since requires glibc.
    • Fixed it by creating stub functions that are missing and implement them
    • Sanitizer requires 8 bytes aligned memory and CSME a 4 bytes aligned memory
    • Each time we enter a “error function” make sure is it in a 4 byte aligned memory and validate that address with the “shadow” if it’s an issue or not
    • Using: “-fsanitize-recover=address” so calling code path won’t change
    • Sanitizer write to a “shadow” memory at a fixed address of “0x20000000”
    • Patch GCC to make it accept “-fsanitize=kernel-address” by removing “SANITIZE_KERNEL_ADDRESS” in “opts-global.c”
    • Was asked many times but none done it as a feature in GCC:
      • https://groups.google.com/forum/#!topic/address-sanitizer/ZL14un1NyoE
    • Making the code too big won’t fit into flash or won’t fit into SRAM
    • Apply only on a single process and not on the entire system at once.
• Using latest industry techniques on silicon
  • Address Sanitization ✓
• Using latest industry techniques on silicon
  • Address Sanitization
  • Fuzzing with Coverage guided
• Using latest industry techniques on silicon
  • Address Sanitization ✓
  • Fuzzing with Coverage guided
    • Based on AFL Fuzzer logic
• Using latest industry techniques on silicon
  • Address Sanitization
  • Fuzzing with Coverage guided
    • Based on AFL Fuzzer logic

Issue #1: BitMap size
• Using latest industry techniques on silicon
  • Address Sanitization
  • Fuzzing with Coverage guided
    • Based on AFL Fuzzer logic

Issue #1: BitMap size

```c
/* Map size for the traced binary (2^MAP_SIZE POW2). Must be greater than 2; you probably want to keep it under 18 or so for performance reasons (adjusting AFL_INST_RATIO when compiling is probably a better way to solve problems with complex programs). You need to recompile the target binary after changing this - otherwise, SEGVs may ensue. */

#define MAP_SIZE POW2 16
```
• Using latest industry techniques on silicon
  • Address Sanitization
  • Fuzzing with Coverage guided
    • Based on AFL Fuzzer logic

  Issue #2: Memory pipe for getting test feedback
Using latest industry techniques on silicon
- Address Sanitization
- Fuzzing with Coverage guided
  - Based on AFL Fuzzer logic

Issue #2: Memory pipe for getting test feedback
- Easy to solve by calling a test firmware API (not exist in production) to get the internal array that hold all feedback
- Modify AFL instrumentation to set the global BITMAP array inside of the FW
• Using latest industry techniques on silicon
  • Address Sanitization
  • Fuzzing with Coverage guided
    • Based on AFL Fuzzer logic
• Using latest industry techniques on silicon
  • Address Sanitization
  • Fuzzing with Coverage guided
    • Based on AFL Fuzzer logic
• Using latest industry techniques on silicon
  • Address Sanitization
  • Fuzzing with Coverage guided
    • Based on AFL Fuzzer logic
• Using latest industry techniques on silicon
  • Address Sanitization
  • Fuzzing with Coverage guided
    • Based on AFL Fuzzer logic
• Using latest industry techniques on silicon
• Address Sanitization
• Fuzzing with Coverage guided
  • Based on AFL Fuzzer logic
• Using latest industry techniques on silicon
  • Address Sanitization ✓
  • Fuzzing with Coverage guided ✓
- Architecture & Boot flow
- OS Security Principles & Internals
- Hardening & Mitigations
- Pre & Post Manufacturing
- Update & Recoverability
- Wrap-up
• Features are configurable by manufacturers
  • Manageability support – corporate / consumer
  • HW Anti-rollback support
  • Manufacturer public key for secure micro-code loading and Intel Boot Guard
  • Intel Boot Guard enable/disable & policy
  • PTT enable/disable

• End Of Manufacturing (EOM)
  • Required by manufacturers before shipping platforms to end-users
  • Write and lock manufacturers’ settings into FPF and CSME data partition in SPI flash
  • Close SPI flash descriptor – SPI controller enforces access control on BIOS, CSME and other SPI regions
Some CSME features can still be configured after EOM by end-users

- Manageability can be configured in BIOS menus
  - AMT out of band network interface enable/disable
  - AMT USB provisioning enable/disable
  - AMT Host Based Provisioning enable/disable
  - AMT redirection enable/disable
• Architecture & Boot flow
• OS Security Principles & Internals
• Hardening & Mitigations
• Pre & Post Manufacturing
• Update & Recoverability
• Wrap-up
CSME FW verifies digital signature and version of new CSME FW image before updating it in SPI flash on end-user system

• Two levels of CSME FW anti-rollback supported in CSME 12
  1. SW rollback to old CSME FW is prevented using Version Control Number (VCN)
  2. Physical rollback is prevented using Anti-Rollback (ARB) SVN
     • ARB SVN is kept in field programmable fuse (FPF)
     • Require manufacturer support

• Once FW update is done to a higher TCB SVN, CSME will perform data migration and initiate the re-creation of attestation keys (EPID and PTT Endorsement Key)
1. If not latest iCLS (Intel Capability Licensing Service) SW service is already used, update SW.

2. Manufacturers update to new CSME FW with **higher SVN**. At next boot, CSME FW performs CSME data migration from previous CSME storage key to new one derived by ROM.

3. Intel iCLS SW service connects securely using Intel SIGMA protocol over internet to Intel backend servers to complete TCB recovery and retrieve new EPID key and Intel certificate for new PTT Endorsement Key (TPM EK).

4. At some point, Intel revokes EPID keys and PTT EK (i.e. publish CRL on Intel server). Once revocation is done, Content Providers can halt streaming content to non-updated systems.
• Architecture & Boot flow
• OS Security Principles & Internals
• Hardening & Mitigations
• Pre & Post Manufacturing
• Update & Recoverability
• Wrap-up
Wrap-up

• Secure design with defense In-depth
  • Secure boot and execution enforced by minimal TCB
  • Least privileges and process isolation
  • Exploitation mitigations

• Secure Update & Recovery
  • Secure FW update
  • FW and HW Anti-rollback
  • Data migration with online renewal of attestation keys (TCB Recovery)

• Evaluating Future Enhancements
  • Further reduce privileges
  • Adding ASLR support given CSME OS memory limitations
  • HW Control-Flow Enforcement Technology (CET) support in CSME CPU
Thank You!

Special thanks to CSME architecture, development, validation and security research teams for their contribution to this presentation.