An Introduction to Platform Security

Spring 2018 UEFI Seminar and Plugfest
March 26-30, 2018
Presented by Brent Holtsclaw and John Loucaides (Intel)
Legal Notice

No computer system can be absolutely secure.

Intel, the Intel logo are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.

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

© Intel Corporation.
Building a Threat Model...

Note: Contents are meant as examples. This does not represent an exhaustive analysis.
Why Attack Firmware?

- **Persistent Compromise**
  - Update firmware image with malicious content
- **Stealthy Compromise**
  - System Management Mode (SMM) code injection
- **Bypass of Security Features**
  - Hypervisor / Virtual Machine Monitor (VMM) Bypass
- **Denial of Service**
  - Corrupt/Delete critical configuration settings
“These draft guidelines promote resiliency in the platform by describing security mechanisms for protecting the platform against unauthorized changes, detecting unauthorized changes that occur, and secure recovery from attacks.”
Standards for a highly secure Windows 10 device

These standards are for general purpose desktops, laptops, tablets, 2-in-1's, mobile workstations, and desktops. This topic applies specifically and uniquely for Windows 10 version 1709, Fall Creators Update. Windows security features are enabled when you meet or exceed these standards and your device is able to provide a highly secure experience.

https://docs.microsoft.com/en-us/windows-hardware/design/device-experiences/oem-highly-secure

**Firmware-related features**

Systems must have firmware that implements Unified Extension Firmware Interface (UEFI) version 2.4+

Systems must have firmware that implements UEFI Class 2 or UEFI Class 3

System's firmware must support UEFI Secure Boot and must have UEFI Secure Boot enabled by default

System's firmware must implement Secure MOR revision 2

Systems must support the Windows* UEFI Firmware Capsule Update specification
Attacks and Platform Assets

- **Persistent Compromise**
  - Update firmware image with malicious content

- **Stealthy Compromise**
  - SMM code injection

- **Bypass of Security Features**
  - VMM Bypass

- **Denial of Service**
  - Corrupt/Delete critical configuration settings

**Boot Media** (e.g. SPI Flash)
- Firmware code
- NVRAM data

**Runtime Firmware** (e.g. SMM)

**HW Configuration** (e.g. locked registers)

These are examples. Not an exhaustive list.
Classes of Attacker

- **Unlimited**
  - Physical
- **Limited**
  - Physical
- **Privileged**
  - Software
- **Unprivileged**
  - Software

Decreasing Attacker Power
Resilient Defense

Boot Media
Runtime Firmware (eg. SMM)
HW Configuration

Detect
Protect
Recover

Decreasing Attacker Power

Unlimited
Limited
Privileged
Unprivileged

Physical
Software
Attack Surface (Interfaces to access/attack assets)

- Boot Media
- Runtime Firmware (eg. SMM)
- HW Configuration

- SPI programmer
- HW/SW Sequencing
- UEFI Variables
- SW SMI
- Device Drivers
- Register Access

Decreasing Attacker Power:
- Unlimited
- Limited
- Privileged
- Unprivileged

Physical
Software

Note: Contents are meant as examples. This does not represent an exhaustive analysis.
Do Firmware Attacks Require Kernel Privileges?

A matter of finding legitimate signed kernel driver which can be used on behalf of user-mode exploit as a *confused deputy*.

*RWEverything* driver signed for Windows 64bit versions (co-discovered with researchers from MITRE)
Securing the Platform

Defending the Boot Media Asset
## Boot Media Resiliency

<table>
<thead>
<tr>
<th>Attack</th>
<th>Protect Mechanism</th>
<th>Detect Mechanism</th>
<th>Recover Mechanism</th>
</tr>
</thead>
</table>
| Direct Write to Boot Media (eg. unlocked SPI, Speed Racer, etc.) | • SPI Controller Config  
• SMM-based Protection  
• TCB reduction | • UEFI Secure (Verified) Boot  
• Measured Boot  
• HW Root of Trust | • Capsule Update and Recovery  
• Independent hardware |

These are examples. Not an exhaustive list.
Boot Media Protections

• SPI Controller
  – Descriptor regions and permissions
  – Protected Range Registers
• SMM-Based BIOS Write Protection
  – SMI when enabling write access
  – Enable write access from SMM
• Reducing the TCB
Detection: Verified and Measured Boot
Detection: Hardware Root of Trust

Move the root of trust from FW to HW by having HW check FW integrity.

Signed BIOS Update

Driver Signing

Signature Check and Hashing

TPM

Drive

Signed BIOS Update

Driver Signing

Signature Check and Hashing

TPM
Securing the Platform

Defending the Runtime Firmware Assets
## Runtime Firmware Resiliency

<table>
<thead>
<tr>
<th>Attack</th>
<th>Protect Mechanism</th>
<th>Detect Mechanism</th>
<th>Recover Mechanism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Call-Outs</td>
<td>• Limited Page Table Access</td>
<td>• Debugger</td>
<td>• Firmware Update</td>
</tr>
<tr>
<td></td>
<td>• No Execute Pages</td>
<td>• Fuzzing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Hardware Check</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Confused Deputy</td>
<td>• Limited Page Table Access</td>
<td>• Fuzzing (e.g. CHIPSEC)</td>
<td>• Firmware Update</td>
</tr>
<tr>
<td>Malicious DMA</td>
<td>• TSEG</td>
<td></td>
<td>• Reboot</td>
</tr>
<tr>
<td></td>
<td>• IOMMU</td>
<td></td>
<td>• Firmware Update</td>
</tr>
</tbody>
</table>

These are examples. Not an exhaustive list.

**System Management Mode (SMM)**

- CPU enters System Management Mode (SMM) upon receiving System Management Interrupt (SMI#) from the chipset or other logical CPU
- CPU (OS) state is saved in SMRAM upon entry to SMM and restored upon exit from SMM. SMRAM is a range of DRAM reserved by BIOS and protected from other runtime code.
- CPU exits SMM to the interrupted OS when SMI handler executes **RSM** instruction (“Resume from SMM”)

UEFI Plugfest – Spring 2018  www.uefi.org
SMI “Confused Deputy” Attacks

Attacker can target SMM itself or bypass VMM protections, writing to VMM or other Guest VM memory
SMI Handler Memory Map Restriction

Phys Memory

SMRAM

Comm Buffer

OS/VMM Memory

SMI Handler Access

SMI Handler Access

Phys Memory

SMRAM

Comm Buffer

OS/VMM Memory
Finding SMM “Pointer” Vulnerabilities

[*][ Module: Testing SMI handlers for pointer validation vulnerabilities

...[*] Allocated memory buffer (to pass to SMI handlers) : 0x00000000DAAC3000
[*] >>>> Testing SMI handlers defined in 'smm_config.ini'..

...[*] testing SMI# 0x1F (data: 0x00) SW SMI 0x1F
[*] writing 0x500 bytes at 0x00000000DAAC3000

> SMI 1F (data: 00)
RAX: 0x5A5A5A5A5A5A5A5A
RBX: 0x00000000DAAC3000
RCX: 0x0000000000000000
RDX: 0x5A5A5A5A5A5A5A5A
RSI: 0x5A5A5A5A5A5A5A5A
RDI: 0x5A5A5A5A5A5A5A5A
< checking buffers contents changed at 0x00000000DAAC3000 +[29,32,33,34,35]
[!] DETECTED: SMI# 1F data 0 (rax=5A5A5A5A5A5A5A5A rbx=DAAC3000 rcx=0 rdx=...)

[-] <<< Done: found 2 potential occurrences of unchecked input pointers

https://www.youtube.com/watch?v=z2Qf45nUeaA
Software/DMA Access to SMRAM

CPU software access to SMRAM is blocked by SMRR

DMA access to SMRAM is blocked due to TSEG covering SMRAM

SMRAM

4GB

SMRR

CPU
Preboot DMA Protection

A Tour Beyond BIOS: Using IOMMU for DMA Protection in UEFI Firmware

This paper presents the idea of using an input-output memory management unit (IOMMU) to resist Direct Memory Access (DMA) attacks in firmware. The example presented uses Intel® Virtualization Technology (Intel® VT) for Directed I/O (Intel® VT-d), and the concept can be applied to other IOMMU engines.

Securing the Platform

Defending the HW Configuration Assets
## Hardware Configuration Resiliency

<table>
<thead>
<tr>
<th>Attack</th>
<th>Protect Mechanism</th>
<th>Detect Mechanism</th>
<th>Recover Mechanism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Reconfiguration (e.g. Remap)</td>
<td>• Configuration Guidance &amp; Locking</td>
<td>• CHIPSEC</td>
<td>• Reboot</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Firmware Update</td>
</tr>
<tr>
<td>Controller Reconfiguration (e.g. SPI)</td>
<td>• Configuration Guidance &amp; Locking</td>
<td>• CHIPSEC</td>
<td>• Reboot</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Firmware Update</td>
</tr>
<tr>
<td>Feature Enable/Disable or Reconfiguration (e.g. IOMMU, instructions, etc)</td>
<td>• Configuration Guidance &amp; Locking</td>
<td>• CHIPSEC</td>
<td>• Reboot</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Firmware Update</td>
</tr>
</tbody>
</table>

These are examples. Not an exhaustive list.
CHIPSEC: Platform Security Assessment Framework

CHIPSEC is a framework for analyzing the security of PC platforms including hardware, system firmware (e.g. BIOS/UEFI), and the configuration of platform components.

Research ➔ Testing ➔ Risk Assessment

**Research**
- Access to hardware from the OS
- Reusable Python based framework

**Testing/Validation**
- Implement test modules that support multiple platforms
- Ability to provide both positive and negative test cases

**Risk Assessment**
- Evaluate new systems for vulnerabilities and mitigations
- Evaluate the state of existing systems
CHIPSEC Architecture

*Other names and brands may be claimed as the property of others.*
Testing Against Known Issues

- CHIPSEC - Framework for Platform Security Assessment
  - Tests for known security issues (ex: locking SPI ROM at runtime)
  - Runs under Microsoft Windows, Linux, Mac OS X, and the UEFI Shell
  - chipsec@intel.com
- Open Source (GPLv2 License)
  - https://github.com/chipsec/chipsec
  - Released in 2014
  - Part of Intel’s Linux UEFI Validation (LUV) suite: https://01.org/linux-uefi-validation
## Examples: Checking Locks with CHIPSEC

<table>
<thead>
<tr>
<th>HW Configuration</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Controller</td>
<td>memconfig</td>
</tr>
<tr>
<td>SPI Descriptor</td>
<td>spi_access</td>
</tr>
<tr>
<td>SPI Controller</td>
<td>spi_lock</td>
</tr>
<tr>
<td>BIOS Write Protection</td>
<td>bios_wp</td>
</tr>
<tr>
<td>Debug Enable/Disable</td>
<td>debug_interface</td>
</tr>
<tr>
<td>Architectural Features</td>
<td>ia32cfg</td>
</tr>
</tbody>
</table>

These are examples. Not an exhaustive list.
Resilient Defense

- Boot Media
- Runtime Firmware (eg. SMM)
- HW Configuration

Decreasing Attacker Power

- Unlimited
- Limited
- Unprivileged
- Privileged

Physical
Software
Thanks for attending the Spring 2018 UEFI Plugfest

For more information on the UEFI Forum and UEFI Specifications, visit http://www.uefi.org

presented by

intel