Platform Security Assessment
with CHIPSEC

https://github.com/chipsec/chipsec

John Loucaides, Yuriy Bulygin
Introduction to Platform Security
What is Platform Security?

Hardware Implementation and Configuration
- Available Security Features
- Correct Configuration of HW Components
- Testing/Demonstration of HW Security Mechanisms

Firmware Implementation and Configuration
- Access Controls on Firmware Interfaces
- Correct Settings of Lock Bits
- Testing/Demonstration of FW Security Mechanisms
Example: System Management Mode

CanSecWest 2006 “Security Issues Related to Pentium System Management Mode” – Duflot

Are lock bits set?

“Attacking SMM Memory via Intel CPU Cache Poisoning” – ITL (Rutkowska, Wojtczuk)

“Getting into the SMRAM: SMM Reloaded” – Duflot, Levillain, Morin, Grumelard

Are SMRRs configured?
Example: BIOS Write Protection

Black Hat USA 2013 “BIOS Security” – MITRE (Kovah, Butterworth, Kallenberg)

NoSuchCon 2013 “BIOS Chronomancy: Fixing the Static Root of Trust for Measurement” – MITRE (Kovah, Butterworth, Kallenberg)

Is BIOS correctly protected?
Motivating Platform Security Assessment...

- BIOS/FW Exploits (BH USA 07, PoC 2007, BH USA 09, DEFCON 16)
- BIOS/FW Rootkits (BH EU 06, BH DC 07, Phrack66)
- SMM Exploits (CSW 2006, Phrack65, Phrack66, BH USA 08, bugtraq, CSW 2009)
- Mebromi malware
- (U)EFI Bootkits (BH USA 2012 @snare, SaferBytes 2012 Andrea Allievi, HITB 2013)
- Intel/McAfee - Evil Maid Just Got Angrier (CSW 2013)
- Intel/McAfee – “A Tale of One Software Bypass of Windows 8 Secure Boot” (BlackHat 2013)
- MITRE - Xeno Kovah - “Defeating Signed BIOS Enforcement” (PacSec 2013)
- ANSSI - Pierre Chifflier – “UEFI and PCI BootKist” (PacSec 2013)
- Dragos Ruiu - “Meet ‘badBIOS’ the mysterious Mac and PC malware that jumps airgaps (#badBios)
- Kaspersky Lab / Absolute Software
- Microsoft Technical Advisory 2871690
- Intel Security/MITRE - All Your Boot Are Belong To Us (CanSecWest 2014)
- Upcoming: MITRE - Setup for Failure (Syscan 2014)
When Is Secure Boot Actually Secure?

When all platform manufacturers...
When Is Secure Boot Actually Secure?

When all platform manufacturers...

- protect the UEFI BIOS from programmable SPI writes by malware,
- allow only signed UEFI BIOS updates,
- protect authorized update software,
- correctly program and protect SPI Flash descriptor,
- protect Secure Boot persistent configuration variables in NVRAM,
- implement authenticated variable updates,
- protect variable update API,
- disable Compatibility Support Module,
- don’t allow unsigned legacy Option ROMs,
- configure secure image verification policies,
- don’t reinvent image verification functionality,
- ...


When Is Secure Boot Actually Secure?

When all platform manufacturers...

• protect the UEFI BIOS from programmable SPI writes by malware,
• allow only signed UEFI BIOS updates,
• protect authorized update software,
• correctly program and protect SPI Flash descriptor,
• protect Secure Boot persistent configuration variables in NVRAM,
• implement authenticated variable updates,
• protect variable update API,
• disable Compatibility Support Module,
• don’t allow unsigned legacy Option ROMs,
• configure secure image verification policies,
• don’t reinvent image verification functionality,
• ...

and don’t introduce a single bug in all of this, of course.
Weakest Link
Introduction to CHIPSEC
How do we raise the bar?

Empowering End-Users to Make a Risk Decision
*Other names and brands may be claimed as the property of others.
## Known Threats and CHIPSEC modules

<table>
<thead>
<tr>
<th>Issue</th>
<th>CHIPSEC Module</th>
<th>Public Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMRAM Locking</td>
<td>common.smm</td>
<td>CanSecWest 2006</td>
</tr>
<tr>
<td>BIOS Keyboard Buffer Sanitization</td>
<td>common.bios_kbrd_buffer</td>
<td>DEFCON 16 2008</td>
</tr>
<tr>
<td>SMRR Configuration</td>
<td>common.smrr</td>
<td>ITL 2009 CanSecWest 2009</td>
</tr>
<tr>
<td>BIOS Protection</td>
<td>common.bios_wp</td>
<td>BlackHat USA 2009 CanSecWest 2013</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Black Hat 2013 NoSuchCon 2013</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Flashrom</td>
</tr>
<tr>
<td>SPI Controller Locking</td>
<td>common.spi_lock</td>
<td>Flashrom Copernicus</td>
</tr>
<tr>
<td>BIOS Interface Locking</td>
<td>common.bios_ts</td>
<td>PoC 2007</td>
</tr>
<tr>
<td>Access Control for Secure Boot Keys</td>
<td>common.secureboot.keys</td>
<td>UEFI 2.4 Spec</td>
</tr>
<tr>
<td>Access Control for Secure Boot Variables</td>
<td>common.secureboot.variables</td>
<td>UEFI 2.4 Spec</td>
</tr>
</tbody>
</table>
Example: System Management Mode

“Attacking SMM Memory via Intel CPU Cache Poisoning” – ITL (Rutkowska, Wojtczuk)

“Getting into the SMRAM: SMM Reloaded” – Duflot, Levillain, Morin, Grumelard

Are SMRRs configured?

common.smrr

[+] imported chipsec.modules.common.smrr

[x] Module: CPU SMM Cache Poisoning / SMM Range Registers (SMRR)

...[+] OK. SMRR are supported in IA32_MTRRCAP_MSR

...[+] OK so far. SMRR Base is programmed

...[+] OK so far. SMRR are enabled in SMRR_MASK MSR

...[+] OK so far. SMRR MSRs match on all CPUs

[+] PASSED: SMRR protection against cache attack seems properly configured
Example: System Management Mode

CanSecWest 2006 “Security Issues Related to Pentium System Management Mode” – Duflot

Are lock bits set?

common.smm

[+] imported chipsec.modules.common.smm
[x][ ==============================================================
[x][ Module: SMM memory (SMRAM) Lock
[x][ ==============================================================
[*] SMRAM register = 0x1A ( D_LCK = 1, D_OPEN = 0 )
[+] PASSED: SMRAM is locked
Example: BIOS Write Protection

Black Hat USA 2013 “BIOS Security” – MITRE (Kovah, Butterworth, Kallenberg)

NoSuchCon 2013 “BIOS Chronomancy: Fixing the Static Root of Trust for Measurement” – MITRE (Kovah, Butterworth, Kallenberg)

Is BIOS correctly protected?


```python
[+] imported chipsec.modules.common.bios_wp
[x][ =======================================================================
[x][ Module: BIOS Region Write Protection
[x][ =======================================================================
BIOS Control (BDF 0:31:0 + 0xDC) = 0x2A
[05] SMM_BWP = 1 (SMM BIOS Write Protection)
[04] TSS = 0 (Top Swap Status)
[01] BLE = 1 (BIOS Lock Enable)
[00] BIOSWE = 0 (BIOS Write Enable)

[+] BIOS region write protection is enabled (writes restricted to SMM)

[*] BIOS Region: Base = 0x00500000, Limit = 0x00FFFFFF
SPI Protected Ranges
------------------------------------------------------------
PRx (offset) | Value    | Base     | Limit    | WP? | RP?
------------------------------------------------------------
PR0 (74)     | 00000000 | 00000000 | 00000000 | 0   | 0
PR1 (78)     | 8FFF0F40 | 00F40000 | 00FFF000 | 1   | 0
PR2 (7C)     | 8EDF0EB1 | 00EB1000 | 00EDF000 | 1   | 0
PR3 (80)     | 8EB00EB0 | 00EB0000 | 00EB0000 | 1   | 0
PR4 (84)     | 8EAF0C00 | 00C00000 | 00EAF000 | 1   | 0

[!] SPI protected ranges write-protect parts of BIOS region (other parts of BIOS can be modified)

[+] PASSED: BIOS is write protected
```
Manual Analysis and Forensics
Direct HW Access for Manual Testing

Examples:

```bash
chipsec_util msr 0x200
chipsec_util mem 0x0 0x41E 0x20
chipsec_util pci enumerate
chipsec_util pci 0x0 0x1F 0x0 0xDC byte
chipsec_util io 0x61 byte
chipsec_util mmcfg 0 0x1F 0 0xDC 1 0x1
chipsec_util cmos dump
chipsec_util ucode id
chipsec_util smi 0x01 0xFF
chipsec_util idt 0
chipsec_util cpuid 1
chipsec_util spi read 0x700000 0x100000 bios.bin
chipsec_util decode spi.bin
chipsec_util uefi var-list
...
```
Forensics

**Live system firmware analysis**

```
chipsec_util spi info
chipsec_util spi dump rom.bin
chipsec_util spi read 0x700000 0x100000 bios.bin
chipsec_util uefi var-list
chipsec_util uefi var-read db D719B2CB-3D3A-4596-A3BC-DAD00E67656F db.bin
```  

**Offline system firmware analysis**

```
chipsec_util uefi keys PK.bin
chipsec_util uefi nvram vss bios.bin
chipsec_util uefi decode rom.bin
chipsec_util decode rom.bin
```
Writing New Modules
Directory Structure

chipsec_main.py  runs modules (see modules dir below)
chipsec_util.py  runs manual utilities (see utilcmd dir below)
/chipsec
  /cfg  platform specific information/offsets
  /hal  all the HW stuff you can interact with
  /helper support for OS/environments
  /modules tests go here
  /utilcmd manual utility commands for chipsec_util
def check_spi_lock():
    logger.start_test( "SPI Flash Controller Configuration Lock" )

    spi_locked = 0
    hsfsts_reg_value = cs.mem.read_physical_mem_dword(
                        get_PCH_RCBA_SPI_base(cs) + SPI_HSFSTS_OFFSET )
    logger.log( '[*] HSFSTS register = 0x%08X' % hsfsts_reg_value )
    logger.log( '    FLOCKDN = %u' % ((hsfsts_reg_value &
                               SPI_HSFSTS_FLOCKDN_MASK)>>15) )

    if 0 != (hsfsts_reg_value & SPI_HSFSTS_FLOCKDN_MASK):
        spi_locked = 1
        logger.log_passed_check( "SPI Flash Controller
configuration is locked\n"
    else:
        logger.log_failed_check( "SPI Flash Controller
configuration is not locked\n"

    return spi_locked==1

def run( module_argv ):
    return check_spi_lock()
What’s next?

See for yourself

• Run CHIPSEC
• Understand the Risks

Special thank you to all CHIPSEC contributors at Intel and partners who have provided excellent feedback!

Contribute to Platform Security Research

https://github.com/chipsec/chipsec