CHIPSEC

Platform Security Assessment Framework

https://github.com/chipsec/chipsec
@CHIPSEC

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

Is Compatible SMRAM Protected?

"Attacking SMM Memory via Intel CPU Cache Poisoning" – Wojtczuk, Rutkowska

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

Is SMRAM Vulnerable to Cache Poisoning Attack?

Example: BIOS Write Protection

Persistent BIOS Infection – Sacco, Ortega

CanSecWest 2013 "Evil Maid Just Got Angrier" – Bulygin

Black Hat USA 2013 "BIOS Security" – Butterworth, Kallenberg, Kovah

"BIOS Chronomancy: Fixing the Core Root of Trust for Measurement" – Butterworth, Kallenberg, Kovah

BlackHat USA 2013 "A Tale Of One Software Bypass Of Windows 8 Secure Boot" – Bulygin, Furtak, Bazhaniuk

Is BIOS Protected in SPI Flash?

Motivating Platform Security Assessment…

- Security Issues Related to Pentium System Management Mode (CSW 2006)
- Implementing and Detecting an ACPI BIOS Rootkit (BlackHat EU 2006)
- Implementing and Detecting a PCI Rootkit (BlackHat DC 2007)
- Programmed I/O accesses: a threat to Virtual Machine Monitors? (PacSec 2007)
- Hacking the Extensible Firmware Interface (BlackHat USA 2007)
- BIOS Boot Hijacking And VMWare Vulnerabilities Digging (PoC 2007)
- Bypassing pre-boot authentication passwords (DEF CON 16)
- Using SMM for "Other Purposes" (Phrack65)
- Persistent BIOS Infection (Phrack66)
- A New Breed of Malware: The SMM Rootkit (BlackHat USA 2008)
- Preventing & Detecting Xen Hypervisor Subversions (BlackHat USA 2008)
- A Real SMM Rootkit: Reversing and Hooking BIOS SMI Handlers (Phrack66)
- Attacking Intel BIOS (BlackHat USA 2009)
- Getting Into the SMRAM: SMM Reloaded (CSW 2009, CSW 2009)
- Attacking SMM Memory via Intel Cache Poisoning (ITL 2009)
- BIOS SMM Privilege Escalation Vulnerabilities (bugtraq 2009)
- System Management Mode Design and Security Issues (IT Defense 2010)
- Analysis of building blocks and attack vectors associated with UEFI (SANS Institute)
- (U)EFI Bootkits (BlackHat USA 2012 @snare, SaferBytes 2012 Andrea Allievi, HITB 2013)
- Evil Maid Just Got Angrier: Why Full-Disk Encryption With TPM Is Insecure On Many Systems (CSW 2013)
- A Tale of One Software Bypass of Windows 8 Secure Boot (BlackHat USA 2013)
- BIOS Chronomancy (NoSuchCon 2013, BlackHat USA 2013, Hack.lu 2013)
- Defeating Signed BIOS Enforcement (PacSec 2013, Ekoparty 2013)
- UEFI and PCI BootKit (PacSec 2013)
- Meet 'badBIOS' the mysterious Mac and PC malware that jumps airgaps (#badBios)
- All Your Boot Are Belong To Us (CanSecWest 2014 Intel and MITRE)
- Setup for Failure: Defeating Secure Boot (Syscan 2014)
- Setup for Failure: More Ways to Defeat Secure Boot (HITB 2014 AMS)
- Analytics, and Scalability, and UEFI Exploitation (INFILTRATE 2014)
- PC Firmware Attacks, Copernicus and You (AusCERT 2014)
- Extreme Privilege Escalation (BlackHat USA 2014)
- Summary of Attacks Against BIOS and Secure Boot (DEF CON 22)

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?

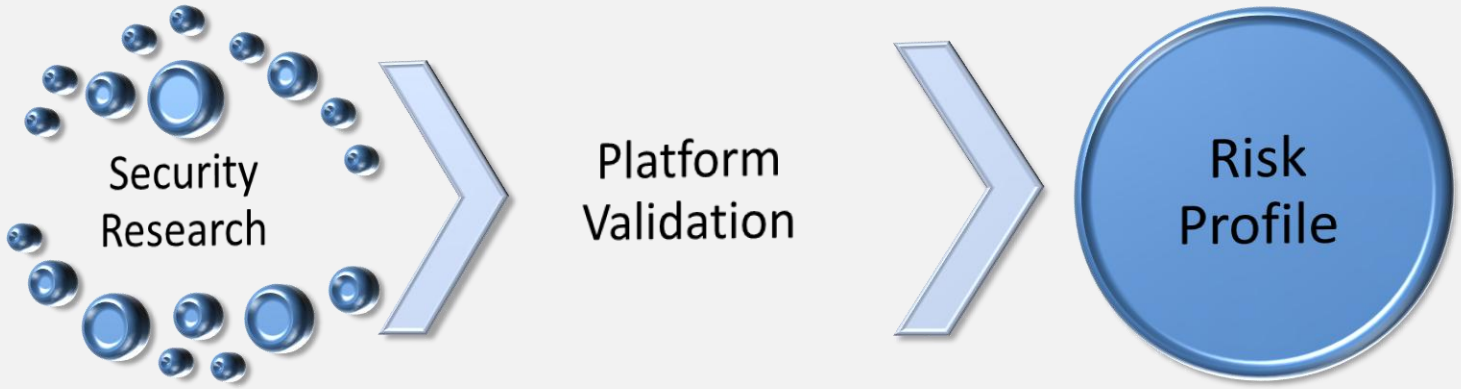
When all platform manufacturers…

- 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.

Introduction to CHIPSEC

How do we raise the bar?

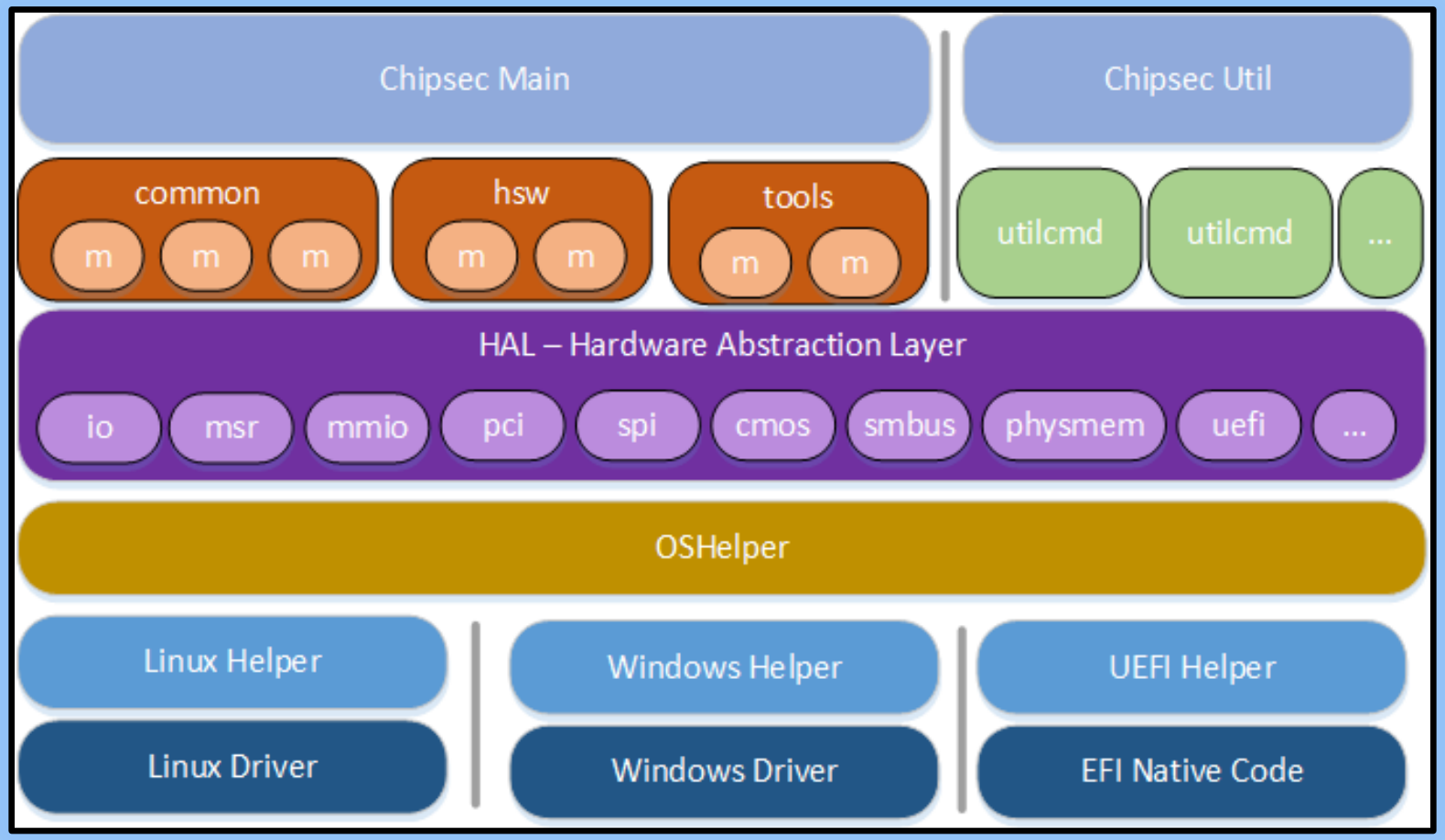


New Attacks

Test Modules for OEMs/IBVs

End-user Risk

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

Issue	CHIPSEC Module	References
SMRAM Locking	common.smm	CanSecWest 2006
BIOS Keyboard Buffer Sanitization	common.bios_kbrd_buffer	DEFCON 16 2008
SMRR Configuration	common.smrr	ITL 2009 CanSecWest 2009
BIOS Protection	common.bios_wp	BlackHat USA 2009 CanSecWest 2013 Black Hat 2013 NoSuchCon 2013 Flashrom
SPI Controller Locking	common.spi_lock	Flashrom Copernicus
BIOS Interface Locking	common.bios_ts	PoC 2007
Access Control for Secure Boot Keys	common.secureboot.keys	UEFI 2.4 Spec
Access Control for Secure Boot Variables	common.secureboot.variables	UEFI 2.4 Spec

Example: System Management Mode

Is SMRAM Vulnerable to Cache Poisoning Attack?

- **common.smrr**

```
[+] imported chipsec.modules.common.smrr
[x][ ============================================================
[x][ Module: CPU SMM Cache Poisoning / SMM Range Registers (SMRR)
[x][ ============================================================
    …
[+] 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

Is Compatibility SMRAM Protected?

- **`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

Is BIOS Protected in SPI Flash?

- **common.bios_wp**

```
[+] 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
```

Structure

`chipsec_main.py`	runs modules (see modules dir below)
`chipsec_util.py`	runs manual utilities (see utilcmd dir below)
`/chipsec`	
`/cfg`	platform specific configuration
`/hal`	all the HW stuff you can interact with
`/helper`	support for OS/environments
`/modules`	modules (tests/tools/PoCs) go here
`/utilcmd`	utility commands for chipsec_util

Writing a Module Example

```
 def check_spi_lock(self):
        self.logger.start_test
        ( "SPI Flash Controller Configuration Lock" )

        spi_locked = 0
        hsfsts_reg_value = self.spi.spi_reg_read( SPI_HSFSTS_OFFSET)
..

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

        return spi_locked==1

    def run( self, module_argv ):
        return self.check_spi_lock()
```

Defined in HAL

Module Starts Here

Manual Analysis and Forensics

BIOS/Firmware 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
```

Manual Access to HW Resources

```
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 mmio list
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
..
```