CHIPSEC GUIDE v0.1 (04/2019)

```
A Help to interpret modules output
23 modules
```

Module chipsec.modules.common.spi_fdopss

Description

This module checks for SPI Controller Flash Descriptor Security Override Pin Strap (FDOPSS). On some systems, this may be routed to a jumper on the motherboard and allow to override Flash Descriptor Security or to enable Intel ME Debug mode

Technical informations

- HSFS[FLOCKDN] is not checked but in chipsec.modules.common.spi_lock module.
- HSFS[FDOPSS] is status bit (Read only)

Valid output

```
[x][ Module: SPI Flash Descriptor Security Override Pin-Strap
[x][ ============================================================================
HSFS = 0xE008 << Hardware Sequencing Flash Status Register (SPIBAR + 0x4)
    [00] FDONE            = 0 << Flash Cycle Done
    [01] FCERR            = 0 << Flash Cycle Error
    [02] AEL              = 0 << Access Error Log
    [03] BERASE           = 1 << Block/Sector Erase Size
    [05] SCIP             = 0 << SPI cycle in progress
    [13] FDOPSS           = 1 << Flash Descriptor Override Pin-Strap Status
    [14] FDV              = 1 << Flash Descriptor Valid
    [15] FLOCKDN          = 1 << Flash Configuration Lock-Down

[+] PASSED: SPI Flash Descriptor Security Override is disabled
```

Module chipsec.modules.common.spi_desc

Description

This module checks that software cannot write to the flash descriptor. The SPI Flash Descriptor (Region 0 of SPI Flash) indicates read/write permissions for

devices to access regions of the flash memory.

Technical informations

- If software can write to the Flash Descriptor, then software could bypass any protection defined by it. While often used for debugging, this should not be the case on production systems.
- FRAP[BMRAG] and FRAP[BMWAG] can grant one or more masters read access to the BIOS region 1 overriding the read permissions in the Flash Descriptor.
- FRAP[BRRA] and FRAP[BRWA] are not checked but in chipsec.modules.common.spi_access module.
- if FRAP[FLOCKDN] = 1 then FRAP register is locked.

Valid output

```
[x][ Module: SPI Flash Region Access Control
[x][ ========================================================================
FRAP = 0x00000A0B << SPI Flash Regions Access Permissions Register (SPIBAR + 0x50)
    [00] BRRA             = B << BIOS Region Read Access
    [08] BRWA             = A << BIOS Region Write Access

    [16] BMRAG            = 0 << BIOS Master Read Access Grant
    [24] BMWAG            = 0 << BIOS Master Write Access Grant

    Software access to SPI flash regions: read = 0x0B, write = 0x0A
    [+] PASSED: SPI flash permissions prevent SW from writing to flash descriptor
```

Module chipsec.modules.common.ia32cfg

Description

This module checks that IA-32/IA-64 architectural features are configured and locked, including IA32 Model Specific Registers (MSRs).

Technical informations

- For each CPU, Bit 0 (Lock) of IA32_Feature_Control register defines if features are locked or not from software access.
- CPU Register IA32_Feature_Control is used to control/enable/disable options (BIOS software uses this register) :
  - VMX (bits 1, 2) : enable extensions in or outside SMX operation, on each cpu

  - enable SGX (code/data memory encryption and isolation, bits 17, 18) on each cpu
- Safer Mode Extensions (SMX) provide a programming interface for system software to establish a measured environment within the platform to support trust decisions by end users. SMX functionality is provided in an Intel 64 processor through the GETSEC instruction via leaf functions.

Valid output

```
[x][ Module: IA32 Feature Control Lock
[x][ ===========================================================================
Verifying IA32_Feature_Control MSR is locked on all logical CPUs..
     cpu0: IA32_Feature_Control Lock = 1
     cpu1: IA32_Feature_Control Lock = 1
     cpu2: IA32_Feature_Control Lock = 1
     cpu3: IA32_Feature_Control Lock = 1
     cpu4: IA32_Feature_Control Lock = 1
     cpu5: IA32_Feature_Control Lock = 1
     cpu6: IA32_Feature_Control Lock = 1
     cpu7: IA32_Feature_Control Lock = 1

    [+] PASSED: IA32_FEATURE_CONTROL MSR is locked on all logical CPUs
```

Module chipsec.modules.common.spi_lock

Description

This module checks that the SPI Flash Controller configuration is locked.

Technical informations

- The configuration of the SPI controller, including protected ranges (PR0-PR4), is locked by HSFS[FLOCKDN] until reset. If not locked, the controller configuration may be modify by reprogramming these registers.
- Only HSFS[FLOCKDN] is checked.

Valid output

```
[x][ Module: SPI Flash Controller Configuration Locks
 [x][ ==========================================================================
HSFS = 0xE008 << Hardware Sequencing Flash Status Register (SPIBAR + 0x4)
    [00] FDONE            = 0 << Flash Cycle Done
    [01] FCERR            = 0 << Flash Cycle Error
```

```
[02] AEL              = 0 << Access Error Log
[03] BERASE           = 1 << Block/Sector Erase Size
[05] SCIP             = 0 << SPI cycle in progress
[13] FDOPSS           = 1 << Flash Descriptor Override Pin-Strap Status
[14] FDV              = 1 << Flash Descriptor Valid
[15] FLOCKDN          = 1 << Flash Configuration Lock-Down
SPI Flash Controller configuration is locked

[+] PASSED: SPI Flash Controller locked correctly.
```

Module chipsec.modules.common.me_mfg_mode

Description

This module checks that specific ME (Management Engine) mode : Manufacturing mode. This mode allows configuring critical platform settings stored in one-time-programmable memory (FUSEs) and some of them are called Field Programmable Fuses (FPFs). FPFs are typically used to store platform parameters.

Technical informations

Setting FPFs requires Intel's ME to be in the Manufacturing Mode. As part of a two-step process, the FPFs are first stored to temporary memory and are then burned when the Manufacturing Mode is closed. If a system remains in Manufacturing Mode, that means the FPFs have never been initialized because the process hasn't been completed. If manufacturers somehow forget to set the FPFs that they need to set for their products and the Manufacturing Mode remains enabled, that could allow attackers to set their own FPFs, and, thus, control the platform. For instance, the attackers could set their own values for Intel BootGuard or other security features. The Intel platform would then automatically load with the attackers' malicious code, regardless of the steps the user would take to protect their machine against malware

Exploitation of misconfiguration

- https://blog.ptsecurity.com/2018/10/intel-me-manufacturing-mode-macbook.html

Valid output

```
[x][ Module: ME Manufacturing Mode
[x][ ================================================================================
```

```
[+] PASSED: ME is not in Manufacturing Mode
```

Module chipsec.modules.common.bios_ts

Description

This module checks for BIOS Interface Lock including Top Swap Mode. Top-Block Swap mode is used to allow for safe update of the Boot Block even when a power failure occurs.

Technical informations

3 bits of 3 differents registers are checked :

- General Control and Status Register / GCS[0] : BIOS Interface Lock-Down (BILD)
- BIOS Control Register BIOS_CNTL[4] : Top Swap Status (TSS)
- Backed Up Control Register / BUC[0] : Top Swap (TS)

Exploitation of misconfiguration

- BIOS Boot Hijacking And VMware Vulnerabilities Digging : http://powerofcommunity.net/poc2007/sunbing.pdf

Valid output

```
[x][ Module: BIOS Interface Lock (including Top Swap Mode)
[x][ ================================================================================
  [*] BiosInterfaceLockDown (BILD) control = 1
  [*] BIOS Top Swap mode is disabled (TSS = 0)
  [*] RTC TopSwap control (TS) = 0
  [+] PASSED: BIOS Interface is locked (including Top Swap Mode)
```

Module chipsec.modules.common.smrr

Description

This module checks to see that SMRRs are enabled and configured.

Technical informations

- If ring 0 software can make SMRAM cacheable and then populate cache lines at SMBASE with exploit code, then when an SMI is triggered, the CPU could execute the exploit code from cache.
- To avoid this attack, System Management Mode Range Registers (SMRRs) force non-cachable behavior and block access to SMRAM when the CPU is not in SMM. These registers need to be enabled/configured by the BIOS and define the protected range of memory and the type of memory for NVRAM (cacheable or not).

Exploitation of misconfiguration

Researchers demonstrated a way to use CPU cache to effectively change values in SMRAM in :

- Attacking SMM Memory via Intel CPU Cache Poisoning : http://www.invisiblethingslab.com/resources/misc09/smm_cache_fun.pdf
- Getting into the SMRAM: SMM Reloaded : http://cansecwest.com/csw09/csw09-duflot.pdf

Valid output

```
[x][ Module: CPU SMM Cache Poisoning / System Management Range Registers
[x][ ===========================================================================
  [+] OK. SMRR range protection is supported

  [*] Checking SMRR range base programming..
  [*] IA32_SMRR_PHYSBASE = 0xCF800006 << SMRR Base Address MSR (MSR 0x1F2)
      [00] Type             = 6 << SMRR memory type
      [12] PhysBase         = CF800 << SMRR physical base address
  [*] SMRR range base: 0x00000000CF800000
  [*] SMRR range memory type is Writeback (WB)
  [+] OK so far. SMRR range base is programmed

  [*] Checking SMRR range mask programming..
  [*] IA32_SMRR_PHYSMASK = 0xFF800800 << SMRR Range Mask MSR (MSR 0x1F3)
      [11] Valid            = 1 << SMRR valid
      [12] PhysMask         = FF800 << SMRR address range mask
  [*] SMRR range mask: 0x00000000FF800000
  [+] OK so far. SMRR range is enabled

  [*] Verifying that SMRR range base & mask are the same on all logical CPUs..
  [CPU0] SMRR_PHYSBASE = 00000000CF800006, SMRR_PHYSMASK = 00000000FF800800
  [CPU1] SMRR_PHYSBASE = 00000000CF800006, SMRR_PHYSMASK = 00000000FF800800
```

```
[CPU2] SMRR_PHYSBASE = 00000000CF800006, SMRR_PHYSMASK = 00000000FF800800
[CPU3] SMRR_PHYSBASE = 00000000CF800006, SMRR_PHYSMASK = 00000000FF800800
[CPU4] SMRR_PHYSBASE = 00000000CF800006, SMRR_PHYSMASK = 00000000FF800800
[CPU5] SMRR_PHYSBASE = 00000000CF800006, SMRR_PHYSMASK = 00000000FF800800
[CPU6] SMRR_PHYSBASE = 00000000CF800006, SMRR_PHYSMASK = 00000000FF800800
[CPU7] SMRR_PHYSBASE = 00000000CF800006, SMRR_PHYSMASK = 00000000FF800800
[+] OK so far. SMRR range base/mask match on all logical CPUs
[*] Trying to read memory at SMRR base 0xCF800000..

[+] PASSED: SMRR reads are blocked in non-SMM mode
[+] PASSED: SMRR protection against cache attack is properly configured
```

Module chipsec.modules.common.smm

Description

This This module simply reads SMRAMC and checks that D_LCK is set.

Technical informations

If SMRAMC[D_LCK] is not set by the BIOS, SMRAM can be accessed even when the CPU is not in SMM. SMRACMC[D_LCK] allow to set in read only mode for several important bits of SMRAM registers (D_OPEN, G_SMRARE, C_BASE_SEG, H_SMRAM_EN, GMS, TOLUD, TOM, TSEG_SG, TSEG_EN)

Exploitation of misconfiguration

- Using CPU SMM to Circumvent OS Security Functions : http://fawlty.cs.usfca.edu/~cruse/cs630f06/duflot.pdf
- Using SMM for Other Purposes : http://phrack.org/issues/65/7.html

Valid output

```
[x][ Module: Compatible SMM memory (SMRAM) Protection
[x][ ================================================================================
  [*] PCI0.0.0_SMRAMC = 0x1A << System Management RAM Control (b:d.f 00:00.0 + 0x88)
    [00] C_BASE_SEG       = ? << SMRAM Base Segment = ?b
    [03] G_SMRAME         = 1 << SMRAM Enabled
    [04] D_LCK            = 1 << SMRAM Locked
    [05] D_CLS            = 0 << SMRAM Closed
    [06] D_OPEN           = 0 << SMRAM Open
    [*] Compatible SMRAM is enabled
```

```
    [+] PASSED: Compatible SMRAM is locked down
```

Module chipsec.modules.common.memlock

Description

This module checks if memory configuration is locked to protect SMM

Valid output

```
[x][ Module: Check MSR_LT_LOCK_MEMORY
[x][ ================================================================================
  [X] Checking MSR_LT_LOCK_MEMORY status
  [*]   cpu0: MSR_LT_LOCK_MEMORY[LT_LOCK] = 1
  [*]   cpu1: MSR_LT_LOCK_MEMORY[LT_LOCK] = 1
  [*]   cpu2: MSR_LT_LOCK_MEMORY[LT_LOCK] = 1
  [*]   cpu3: MSR_LT_LOCK_MEMORY[LT_LOCK] = 1
  [*]   cpu4: MSR_LT_LOCK_MEMORY[LT_LOCK] = 1
  [*]   cpu5: MSR_LT_LOCK_MEMORY[LT_LOCK] = 1
  [*]   cpu6: MSR_LT_LOCK_MEMORY[LT_LOCK] = 1
  [*]   cpu7: MSR_LT_LOCK_MEMORY[LT_LOCK] = 1

  [+] PASSED: Check have successfully
```

Module chipsec.modules.common.rtclock

Description

This module checks for RTC memory locks. RTC stands for Real Time Clock, which is the crystal oscillator controlled timer that maintains the time and date in the computer when switched off. Today, most computers have moved the settings from CMOS and integrated them into the southbridge or Super I/O chips.

Technical informations

Since we do not know what RTC memory will be used for on a specific platform, WARNING (rather than FAILED) is returned if the memory is not locked.

Valid output

```
[x][ Module: Protected RTC memory locations
[x][ ============================================================================
  [*] RC = 0x0000001C << RTC Configuration (RCBA + 0x3400)
    [02] UE               = 1 << Upper 128 Byte Enable
    [03] LL               = 1 << Lower 128 Byte Lock
    [04] UL               = 1 << Upper 128 Byte Lock
  [+] Protected bytes (0x38-0x3F) in low 128-byte bank of RTC memory are locked
  [+] Protected bytes (0x38-0x3F) in high 128-byte bank of RTC memory are locked

  [+] PASSED: Protected locations in RTC memory are locked
```

Module chipsec.modules.remap

Description

This module checks the Memory Remapping Configuration (if correct and locked).

Technical informations

- RAM Remapping must respect the condition : REMAPBASE <= REMAPLIMIT < TOUU
- Protection of registers TOUUD (Top of Upper Usable DRAM), TOLUD (Top of Lower Usable DRAM), REMAPBASE and REMAPLIMIT (Memory Remap Base Address) and REMAPLIMIT (Memory Remap Limit Address) are also checked by chipsec.modules.memconfig module.

Valid output

```
[x][ Module: Memory Remapping Configuration
[x][ ============================================================================
[*] Registers:
[*]   TOUUD     : 0x000000026F600001
[*]   REMAPLIMIT: 0x000000026F500001
[*]   REMAPBASE : 0x00000001FF000001
[*]   TOLUD     : 0x8FA00001
[*]   TSEGMB    : 0x8B000001

[*] Memory Map:
[*]   Top Of Upper Memory: 0x000000026F600000
[*]   Remap Limit Address: 0x000000026F5FFFFF
[*]   Remap Base Address : 0x00000001FF000000
```

```
[*]   4GB                : 0x0000000100000000
[*]   Top Of Low Memory  : 0x000000008FA00000
[*]   TSEG (SMRAM) Base  : 0x000000008B000000

[*] checking memory remap configuration..
[*]   Memory Remap is enabled
[+]   Remap window configuration is correct: REMAPBASE <= REMAPLIMIT < TOUUD
[+]   All addresses are 1MB aligned
[*] checking if memory remap configuration is locked..
[+]   TOUUD is locked
[+]   TOLUD is locked
[+]   REMAPBASE and REMAPLIMIT are locked
[+] PASSED: Memory Remap is configured correctly and locked
```

Module chipsec.modules.smm_dma

Description

This module examines the configuration and locking of SMRAM range configuration protecting from DMA attacks. If it fails, then DMA protection (TSEG) may not be securely configured to protect SMRAM.

Technical informations

- Just like SMRAM needs to be protected from software executing on the CPU, it also needs to be protected from devices that have direct access to DRAM (DMA). Protection from DMA is configured through proper programming of SMRAM memory range. If BIOS does not correctly configure and lock the configuration, then malware could reprogram configuration and open SMRAM area to DMA access, allowing manipulation of memory that should have been protected.
- TSEG is an SMRAM extension and define a memory addresses range to protect by the motherboard chipset (TSEG Memory Base Register for range memory, bits 31:20). All SMRAM memory must be a part of TSEG in order to be protected. The TSEG configuration must be protected against writing (TSEG Memory Base Register for range memory, LOCK <=> bit 0)

Valid output

```
[x][ Module: SMM TSEG Range Configuration Check
[x][ [x][ ============================================================================
[*] TSEG      : 0x00000000CF800000 - 0x00000000CFFFFFFF (size = 0x00800000)
```

```
[*] SMRR range: 0x00000000CF800000 - 0x00000000CFFFFFFF (size = 0x00800000)

[*] checking TSEG range configuration..
  [+] TSEG range covers entire SMRAM
  [+] TSEG range is locked

  [+] PASSED: TSEG is properly configured. SMRAM is protected from DMA attacks
```

Module chipsec.modules.memconfig

Description

This module verifies memory map secure configuration, i.e. that memory map registers are correctly configured and locked down.

Technical informations

Bit LOCK (0) of each register is checked.

Valid output

```
[x][ Module: Host Bridge Memory Map Locks
[x][ ============================================================================
[+] PCI0.0.0_BDSM        = 0x00000000D0000001 - LOCKED   - Base of Graphics Stolen Memory
[+] PCI0.0.0_BGSM        = 0x00000000D0000001 - LOCKED   - Base of GTT Stolen Memory
[+] PCI0.0.0_DPR         = 0x00000000CF800001 - LOCKED   - DMA Protected Range
[+] PCI0.0.0_GGC         = 0x0000000000000003 - LOCKED   - Graphics Control
[+] PCI0.0.0_MESEG_MASK  = 0x0000007FFE000C00 - LOCKED   - Manageability Engine Limit Addres
[+] PCI0.0.0_PAVPC       = 0x0000000000000004 - LOCKED   - PAVP Configuration
[+] PCI0.0.0_REMAPBASE   = 0x00000007FE000001 - LOCKED   - Memory Remap Base Address
[+] PCI0.0.0_REMAPLIMIT  = 0x000000082DF00001 - LOCKED   - Memory Remap Limit Address
[+] PCI0.0.0_TOLUD       = 0x00000000D0000001 - LOCKED   - Top of Low Usable DRAM
[+] PCI0.0.0_TOM         = 0x0000000800000001 - LOCKED   - Top of Memory
[+] PCI0.0.0_TOUUD       = 0x000000082E000001 - LOCKED   - Top of Upper Usable DRAM
[+] PCI0.0.0_TSEGMB      = 0x00000000CF800001 - LOCKED   - TSEG Memory Base
[+] PASSED: All memory map registers seem to be locked down
```

Module chipsec.modules.common.bios_wp

Description

This module common.bios_wp will fail if SMM-based protection is not correctly configured and SPI protected ranges (PR registers) do not protect the entire

BIOS region.

Technical informations

- BIOSWE allow to lock writing on Flash SPI Region, BLE to control (SMI interruption) modifications on bit BIOSWE, SMM_BWP prevent writing from kernel space.
- PR0<=>Desc BIOS Region, PR1<=>BIOS Region, PR2<=>ME Region, PR3<=>GbE Region, PR4<=>PDR Region

Valid output

```
[x][ Module: BIOS Region Write Protection
[x][ ================================================================================
[*] BC = 0x00 << BIOS Control (b:d.f 00:31.0 + 0xDC)
    [00] BIOSWE           = 0 << BIOS Write Enable
    [01] BLE              = 1 << BIOS Lock Enable
    [02] SRC              = 0 << SPI Read Configuration
    [04] TSS              = 0 << Top Swap Status
    [05] SMM_BWP          = 1 << SMM BIOS Write Protection
  [+] BIOS region write protection is enabled (writes restricted to SMM)

    [*] BIOS Region: Base = 0x????????, Limit = 0x????????
    SPI Protected Ranges
    ------------------------------------------------------------
    PRx (offset) | Value    | Base     | Limit    | WP? | RP?
    ------------------------------------------------------------
    PR0 (??)     | ???????? | ???????? | ???????? | 1   | 0
    PR1 (??)     | ???????? | ???????? | ???????? | 1   | 0
    PR2 (??)     | ???????? | ???????? | ???????? | 1   | 0
    PR3 (??)     | 00000000 | 00000000 | 00000000 | 0   | 0
    PR4 (??)     | 00000000 | 00000000 | 00000000 | 0   | 0

  [+] PASSED: BIOS is write protected (by SMM and SPI Protected Ranges)
```

Module chipsec.modules.common.bios_smi

Description

This module checks that SMI events configuration is locked. SMI events allow to stop the attempts to modify register like BIOS_CNTL with bit 0 (BIOSWE).

Technical informations

- Differents registers are checked:
  - SMI_EN register provides a lot of control over the generation of SMI#.
  - SMI_LOCK bit of GEN_PMCON_1 register allows to lock configuration of SMI_EN register.
  - TCO_LOCK bit of TCO1_CNT register allows to lock modification of SMI_EN[TCO_EN].
- The acronym TCO (Total Cost of Ownership) refer to a logic block in the Intel ICH products family.
- If the SMI configuration space is not locked in writing, these events can be modified or deleted to allow a modification of register BIOS_CNTL.

Exploitation of misconfiguration (SecureBoot by pass)

- Summary of Attacks Against BIOS and Secure Boot : https://www.defcon.org/images/defcon-22/dc-22-presentations/Bulygin-Bazhaniul-Furtak-Loucaides/DEFCON-22-Bulygin-Bazhaniul-Furtak-Loucaides-Summary-of-attacks-against-BIOS-UPDATED.pdf

Valid output

```
[x][ Module: SMI Events Configuration
[x][ ================================================================================
[*] Checking SMI enables..
    Global SMI enable: 1
    TCO SMI enable   : 1
    [+] All required SMI events are enabled
    [*] Checking SMI configuration locks..
      [+] TCO SMI configuration is locked (TCO SMI Lock)
      [+] SMI events global configuration is locked (SMI Lock)

    [+] PASSED: All required SMI sources seem to be enabled and locked
```

Module chipsec.modules.common.bios_kbrd_buffer

Description

This module checks for BIOS/HDD password exposure through BIOS keyboard buffer and checks for exposure of pre-boot passwords (BIOS/HDD/pre-boot authentication SW) in the BIOS keyboard buffer.

Technical informations

The BIOS API offers interruption 0x16to retrieve keystrokes from the keyboard and uses a buffer to work in order to use extended keystrokes (e.g.: Alt + Shift + Keystroke). This Buffer, into the memory physical, is not flushed after using and can be read by an attacker (from OS).

Exploitation of the vulnerability

- Bypassing Pre-boot Authentication Passwords by Instrumenting the BIOS Keyboard Buffer : https://www.defcon.org/images/defcon-16/dc16-presentations/brossard/defcon-16-brossard-wp.pdf

Valid output

```
[x][ Module: Pre-boot Passwords in the BIOS Keyboard Buffer
[x][ ===========================================================================
[*] Keyboard buffer head pointer = 0x0 (at 0x41A), tail pointer = 0x0 (at 0x41C)
[*] Keyboard buffer contents (at 0x41E):
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
[*] Checking contents of the keyboard buffer..

[+] PASSED: Keyboard buffer looks empty. Pre-boot passwords don't seem to be exposed
```

Module chipsec.modules.common.spi_access

Description

This module checks the SPI Flash Region Access Permissions programmed in the Flash Descriptor from Software access.

Technical informations

- If the bit is set for bytes HSFS[BRRA] and HSFS[BRWA], the specific master can erase and write that particular region through register accesses
- HSFS[BMRAG] and HSFS[BMWAG] are not checked but in chipsec.modules.common.spi_desc module
- if HSFS[FLOCKDN] = 1 then HSFS[FRAP] register is locked
- BRRA and BRWA are status bits (Read only) to indicate access rights on Flash Regions (7)
- Read Access is accepted and write access on FREG1_BIOS is accepted

Valid output

```
[x][ Module: SPI Flash Region Access Control
[x][ ============================================================================
SPI Flash Region Access Permissions
-------------------------------------------------------------
[*] FRAP = 0x00000A0B << SPI Flash Regions Access Permissions Register (SPIBAR + 0x50)
    [00] BRRA             = 03|07|0B|0F << BIOS Region Read Access
    [08] BRWA             = 02 << BIOS Region Write Access
    [16] BMRAG            = 0 << BIOS Master Read Access Grant
    [24] BMWAG            = 0 << BIOS Master Write Access Grant

BIOS Region Write Access Grant (00):
  FREG0_FLASHD: 0
  FREG1_BIOS  : 0
  FREG2_ME    : 0
  FREG3_GBE   : 0
  FREG4_PD    : 0
  FREG5       : 0
  FREG6       : 0
BIOS Region Read Access Grant (00):
  FREG0_FLASHD: 0
  FREG1_BIOS  : 0
  FREG2_ME    : 0
  FREG3_GBE   : 0
  FREG4_PD    : 0
  FREG5       : 0
  FREG6       : 0
BIOS Region Write Access (02):
  FREG0_FLASHD: 0
  FREG1_BIOS  : 1
  FREG2_ME    : 0
  FREG3_GBE   : 0
  FREG4_PD    : 0
  FREG5       : 0
  FREG6       : 0
BIOS Region Read Access (03|07|0B|0F):
  FREG0_FLASHD: 1
  FREG1_BIOS  : 1
  FREG2_ME    : ?
  FREG3_GBE   : ?
  FREG4_PD    : 0
  FREG5       : 0
  FREG6       : 0
```

```
[+] PASSED: SPI Flash Region Access Permissions in flash descriptor look ok
```

Module chipsec.modules.common.cpu.spectre_v2

Description

In 2018, reasearchers discovered that CPU data cache timing can be abused to efficiently leak information out of mis-speculated execution, leading to arbitrary virtual memory read vulnerabilities across local security boundaries in various contexts.

There are three known variants of the issue:

- Variant 1: bounds check bypass (CVE-2017-5753)
- Variant 2: branch target injection (CVE-2017-5715)
- Variant 3: rogue data cache load (CVE-2017-5754)

This module checks the variant 2 in verifying if system includes hardware mitigations for Speculative Execution Side Channel.

Technical informations

The module checks if the following hardware mitigations (with hardware registers) are supported by the CPU and enabled by the OS/software:

- Indirect Branch Restricted Speculation (IBRS) and Indirect Branch Predictor Barrier (IBPB):
  - CPUID.(EAX=7H,ECX=0):EDX[26] == 1 > enumerates support for IBRS and IBPB
- Single Thread Indirect Branch Predictors (STIBP):
  - CPUID.(EAX=7H,ECX=0):EDX[27] == 1 > enumerates support for STIBP
  - IA32_SPEC_CTRL[STIBP] == 1 > enable control for STIBP by the software/OS
- Enhanced IBRS:
  - CPUID.(EAX=7H,ECX=0):EDX[29] == 1 > enumerates support for the IA32_ARCH_CAPABILITIES MSR
  - IA32_ARCH_CAPABILITIES[IBRS_ALL] == 1 > enumerates support for enhanced IBRS
  - IA32_SPEC_CTRL[IBRS] == 1 > enable control for enhanced IBRS by the software/OS

Exploitation of misconfiguration

- Technical details of Spectre/Meldown : https://googleprojectzero.blogspot.com/2018/01/reading-privileged-memory-with-side.html
- Spectre Attacks (variant 1 and 2), Exploiting Speculative Execution : https://spectreattack.com/spectre.pdf
- Meltdown (variant 3), Reading Kernel Memory from User Space : https://meltdownattack.com/meltdown.pdf

Valid output

```
[x][ Module: Checks for Branch Target Injection / Spectre v2 (CVE-2017-5715)
[x][ ================================================================================
  [*] CPUID.7H:EDX[26] = 1 Indirect Branch Restricted Speculation (IBRS) & Predictor Barrier
  [*] CPUID.7H:EDX[27] = 1 Single Thread Indirect Branch Predictors (STIBP)
  [*] CPUID.7H:EDX[29] = 0 IA32_ARCH_CAPABILITIES
  [+] CPU supports IBRS and IBPB
  [+] CPU supports STIBP
  [*] checking enhanced IBRS support in IA32_ARCH_CAPABILITIES...
  [+] CPU supports enhanced IBRS (on all logical CPU)
  [+] OS enabled Enhanced IBRS (on all logical processors)

  [+] PASSED: CPU and OS support hardware mitigations
```

Module chipsec.modules.common.uefi.access_uefispec

Description

This module checks protection of UEFI variables defined in the UEFI spec to have certain permissions. Returns failure if variable attributes are not as defined in table 11 Global Variables http://uefi.org of the UEFI spec.

Technical informations

Possibilities for attributes are :

- Non-Volatile (NV) : Stored in SPI Flash based NVRAM
- Boot Service (BS) : Accessible to DXE drivers / Boot Loaders at boot time
- Run-Time (RT) : Accessible to the OS through run-time UEFI SetVariable/GetVariable API
- Time-Based Authenticated Write Access (TBAWS) : see chipsec.modules.common.secureboot.variables section
- Authenticated Write Access (AWS) : see chipsec.modules.common.secureboot.variables section

Valid output

```
[x][ Module: Access Control of EFI Variables
[x][ ============================================================================
[*] Testing UEFI variables ..
[*] Variable PlatformLangCodes (BS+RT)
[*] Variable BootOrder (NV+BS+RT)
[*] Variable GsetUefiIplDefaultValue (NV+BS+RT)
[*] Variable BootFlow (BS+RT)
[*] Variable Ar00000000 (NV+BS+RT)
[*] Variable ConOut (NV+BS+RT)
[*] Variable DefaultLegacyDevOrder (NV+BS+RT)
[*] Variable ProgressBarPolicyVar (BS+RT)
[*] Variable NBPlatformData (BS+RT)
[*] Variable TcgMonotonicCounter (NV+BS+RT)
[*] Variable DriverHealthCount (BS+RT)
[*] Variable ServiceTag (NV+BS+RT)
[*] Variable EsataBay (NV+BS+RT)
[*] Variable AssetTag (NV+BS+RT)
[*] Variable db (NV+BS+RT+TBAWS)
[*] Variable TdtAdvancedSetupDataVar (NV+BS+RT)
[*] Variable P (NV+BS+RT)
[*] Variable TPMPERBIOSFLAGS (NV+BS+RT)
[*] Variable GsetLegacyIplDefaultValue (NV+BS+RT)
[*] Variable BootOneDevice (BS+RT)
[*] Variable UsbMassDevNum (BS+RT)
[*] Variable ColdReset (BS+RT)
[*] Variable WdtPersistentData (NV+BS+RT)
[*] Variable ScramblerBaseSeed (NV+BS+RT)
[*] Variable ConOutDev (BS+RT)
[*] Variable SerialPortsEnabledVar (BS+RT)
[*] Variable TcgPPIVarAddr (NV+BS+RT)
[*] Variable DefaultBootOrder (NV+BS+RT)
[*] Variable UsbMassDevValid (BS+RT)
[*] Variable PK (NV+BS+RT+TBAWS)
[*] Variable BootFFFD (NV+BS+RT)
[*] Variable BootFFFE (NV+BS+RT)
[*] Variable BootFFFB (NV+BS+RT)
[*] Variable BootFFFC (NV+BS+RT)
[*] Variable Boot0001 (NV+BS+RT)
[*] Variable SetupSnbPpmFeatures (NV+BS+RT)
[*] Variable NvRamSpdMap (NV+BS+RT)
[*] Variable SetupPlatformData (BS+RT)
[*] Variable OsIndications (NV+BS+RT)
[*] Variable CurrentPriority (NV+BS+RT)
```

```
[*] Variable BootOptionSupport (BS+RT)
[*] Variable MonotonicCounter (NV+BS+RT)
[*] Variable ConInDev (BS+RT)
[*] Variable S3CpuThrottle (NV+BS+RT)
[*] Variable EDIDLastData (NV+BS+RT)
[*] Variable ErrOut (BS+RT + NV)
[*] Variable TxtFeatures (BS+RT)
[*] Variable OsType (NV+BS+RT)
[*] Variable PchS3Peim (BS+RT)
[*] Variable Lang (NV+BS+RT)
[*] Variable An00000000 (NV+BS+RT)
[*] Variable PchInit (NV+BS+RT)
[*] Variable GNVS_PTR (BS+RT)
[*] Variable OsIndicationsSupported (BS+RT)
[*] Variable FPDT_Variable (NV+BS+RT)
[*] Variable BootCurrent (BS+RT)
[*] Variable Timeout (NV+BS+RT)
[*] Variable Rd00000000 (NV+BS+RT)
[*] Variable SetupDptfFeatures (NV+BS+RT)
[*] Variable SignatureSupport (BS+RT+AWS)
[*] Variable KEK (NV+BS+RT+TBAWS)
[*] Variable SetupMode (BS+RT+AWS)
[*] Variable PreviousBootServiceDataInfo (NV+BS+RT)
[*] Variable Boot0000 (NV+BS+RT)
[*] Variable ErrOutDev (BS+RT)
[*] Variable Boot0003 (NV+BS+RT)
[*] Variable Boot0005 (NV+BS+RT)
[*] Variable MemoryOverwriteRequestControl (NV+BS+RT)
[*] Variable SecureBoot (BS+RT+AWS)
[*] Variable EfiTime (NV+BS+RT)
[*] Variable DFNS (NV+BS+RT)
[*] Variable DebuggerSerialPortsEnabledVar (BS+RT)
[*] Variable ConIn (NV+BS+RT)
[*] Variable Boot0012 (NV+BS+RT)
[*] Variable Boot0011 (NV+BS+RT)
[*] Variable Boot0010 (NV+BS+RT)
[*] Variable SaPegData (NV+BS+RT)
[*] Variable InSetup (BS+RT)
[*] Variable M (NV+BS+RT)
[*] Variable MBPDataPoint (BS+RT)
[*] Variable TxtOneTouch (NV+BS+RT)
[*] Variable OemCpuData (BS+RT)
[*] Variable NBGopPlatformData (BS+RT)
[*] Variable AMITCGPPIVAR (NV+BS+RT)
[*] Variable DriverHlthEnable (BS+RT)
[*] Variable LangCodes (BS+RT)
```

```
[*] Variable Boot000D (NV+BS+RT)
[*] Variable Boot000E (NV+BS+RT)
[*] Variable Boot000F (NV+BS+RT)
[*] Variable PlatformLang (NV+BS+RT)
[*] Variable Rc00000000 (NV+BS+RT)

[+] PASSED: All Secure Boot UEFI variables are protected
```

Module chipsec.modules.common.uefi.s3bootscript

Description

This module checks protections of the S3 resume boot-script implemented by the UEFI based firmware. UEFI Boot Script is a data structure interpreted by UEFI firmware during S3 resume.

Technical informations

- In some cases, an attacker with ring0 privileges can alter this data structure. As a result, by forcing S3 suspend/resume cycle, an attacker can run arbitrary code on a platform that is not yet fully locked (BIOS_CNTL not locked and SMRAM via DMA not locked with TSEGMB lock bit). The consequences include ability to overwrite the flash storage and take control over SMM.

Exploitation of misconfigurations

- Attacks on UEFI Security : https://events.ccc.de/congress/2014/Fahrplan/system/attachments/2557/original/AttacksOnUEFI_Slides.pdf
- Attacking UEFI Boot Script : https://bromiumlabs.files.wordpress.com/2015/01/venamis_whitepaper.pdf
- Technical details to exploit vulnerability : http://blog.cr4.sh/2015/02/exploiting-uefi-boot-script-table.html

Valid output

```
[x][ Module: S3 Resume Boot-Script Protections
[x][ ================================================================================
  [*] SMRAM: Base = 0x00000000CF800000, Limit = 0x00000000CFFFFFFF, Size = 0x00800000
  [+] Didn't find any S3 boot-scripts in EFI variables
  [!] WARNING: S3 Boot-Script was not found. Firmware may be using other ways to store/locat

  OR
```

```
[x][ Module: S3 Resume Boot-Script Protections
[x][ ============================================================================
  [*] SMRAM: Base = 0x00000000CF800000, Limit = 0x00000000CFFFFFFF, Size = 0x00800000
  [*] Found ? S3 boot-script(s) in EFI variables
  [*] Checking entry-points of Dispatch opcodes..
    [+] ????? > PROTECTED
    [+] ????? > PROTECTED
    [+] ????? > PROTECTED
    [+] ????? > PROTECTED
  [+] S3 boot-script is in SMRAM
```

Module chipsec.modules.common.secureboot.variables

Description

This module checks that all Secure Boot key/whitelist/blacklist UEFI variables are authenticated and protected from unauthorized modification (only from application not signed because it is possible to modify Secureboot variables from local access). Secureboot allows to protect against modification of critical codes used to start a computer and its operating system (UEFI binaries, drivers, Kernbel Grub, Windows Boot Loader, OS Boot Loader, ...). Without SecureBoot and Flash protection, UEFI BootKit can be installed easly.

Technical informations

SecureBoot UEFI variables are stored into SPI Flash into NVRAM volume and various key databases are used to configure SecureBoot :

- DB (aka, 'signature database'): contains the trusted keys used for authenticating any applications or drivers executed in the UEFI environment.
- DBX (aka, 'forbidden signature database' or 'signature database blacklist'): contains a set of explicitly untrusted keys and binary hashes. Any application or driver signed by these keys or matching these hashes will be blocked from execution.
- KEK (key exchange keys database): contains the set of keys trusted for updating DB and DBX.
- PK (platform key - while PK is often referred to simply as a single public key, it could be implemented as a database). Only updates signed with PK can update the KEK database.
- SecureBoot: Enables/disables image signature checks.
- SetupMode: SETUP_MODE allows updating KEK/db(x), self-signed PK.

Checked attibuts for each SecureBoot variables are :

- Authenticated Write Access – PK cert verifies PK/KEK update – KEK verifies db/dbx update – certdb verifies general authenticated EFI variable updates
- Time-Based Authenticated Write Access
  - Same attributs as Authenticated Write Access +
  - Signed with time-stamp (anti-replay)

Warning message :

- If Secureboot are been disabled from BIOS interface (in order to boot on linux live to execute Chipsec), the message : Secure Boot appears to be disabled will be returned.
- If no blacklist has been defined, the warning message : Some required Secure Boot variables are missing will be returned.

Valid output

```
[x][ Module: Attributes of Secure Boot EFI Variables
[x][ ================================================================================
[*] Checking protections of UEFI variable 8be4df61-93ca-11d2-aa0d-00e098032b8c:SecureBoot
[+] Variable 8be4df61-93ca-11d2-aa0d-00e098032b8c:PK is authenticated
\(AUTHENTICATED_WRITE_ACCESS)
[*] Checking protections of UEFI variable 8be4df61-93ca-11d2-aa0d-00e098032b8c:SetupMode
[+] Variable 8be4df61-93ca-11d2-aa0d-00e098032b8c:PK is authenticated
\(AUTHENTICATED_WRITE_ACCESS)
[*] Checking protections of UEFI variable 8be4df61-93ca-11d2-aa0d-00e098032b8c:PK
[+] Variable 8be4df61-93ca-11d2-aa0d-00e098032b8c:PK is authenticated
\(TIME_BASED_AUTHENTICATED_WRITE_ACCESS)
[*] Checking protections of UEFI variable 8be4df61-93ca-11d2-aa0d-00e098032b8c:KEK
[+] Variable 8be4df61-93ca-11d2-aa0d-00e098032b8c:KEK is authenticated
\(TIME_BASED_AUTHENTICATED_WRITE_ACCESS)
[*] Checking protections of UEFI variable d719b2cb-3d3a-4596-a3bc-dad00e67656f:db
[+] Variable d719b2cb-3d3a-4596-a3bc-dad00e67656f:db is authenticated \
(TIME_BASED_AUTHENTICATED_WRITE_ACCESS)
[*] Checking protections of UEFI variable d719b2cb-3d3a-4596-a3bc-dad00e67656f:dbx
[+] Variable d719b2cb-3d3a-4596-a3bc-dad00e67656f:dbx is authenticated \
(TIME_BASED_AUTHENTICATED_WRITE_ACCESS)

[+] PASSED: All Secure Boot UEFI variables are protected
```

Module chipsec.modules.debugenabled

Description

This module checks if the system has debug features turned on, specifically the Direct Connect Interface (DCI).

Technical informations

- With activated DCI interface and debug options, an attacker can simply plug into an external USB port to install a persistent rootkit, bypassing secure boot and many other security features.
- The module checks if the following hardware are well configurated :
  - P2SB_DCI.DCI_CONTROL_REG[HDCIEN] > to enable/disable DCI Direct Connect Interface
  - IA32_DEBUG_INTERFACE[DEBUGENABLE] > to enable/disable CPU debug features
  - IA32_DEBUG_INTERFACE[DEBUGLOCK] > to unlock/lock manually or automatically (with SMI#)
  - IA32_DEBUG_INTERFACE[DEBUGEOCCURED] > to indicate the status of bit DEBUGENABLE (RO)

Exploitation of misconfiguration

- Evil Maid Firmware Attacks Using USB Debug : https://eclypsium.com/2018/07/23/evil-maid-firmware-attacks-using-usb-debug/

Valid output

```
  [*] NOT IMPLEMENTED: CPU Debug features are not supported on this platform
    Skipping module chipsec.modules.debugenabled since it is not supported in this platform

OR

[x][ Module: CPU Debug features are not supported on this platform
[x][ ================================================================================
  [X] Checking IA32_DEBUG_INTERFACE msr status
  [+] CPU IA32_DEBUG_INTERFACE is enabled
  [X] Checking DCI register status
  [+] DCI Debug is disabled

  [+] All checks have successfully passed
```

Module chipsec.modules.common.sgx_check

Description

This module checks SGX related configuration into hardware register and protected memory. SGX is a CPU option to isolate and encrypt code and data between same process. This option consuming resources must be skippable.

Technical informations

The following hardware are checked :

- IA32_FEATURE_CONTROL[LOCK] > to prevent writing on this register
- IA32_FEATURE_CONTROL[ENABLE] > to enable/disable SGX

Valid output

```
[x][ Module: Check SGX feature support
[x][ ===========================================================================
[*] Test if CPU has support for SGX
[*] SGX BIOS enablement check
[*] Verifying IA32_FEATURE_CONTROL MSR is configured
[+] Intel SGX is Enabled in BIOS

[*] Verifying IA32_FEATURE_CONTROL MSR is locked
[+] IA32_Feature_Control locked

[*] Verifying if Protected Memory Range (PRMRR) is configured
[+] Protected Memory Range configuration is supported
[*] Verifying PRMR Configuration on each core.
[+] PRMRR config is uniform across all CPUs

[*] Verifying if SGX instructions are supported
...
[+] Intel SGX is available to use
...
[*] Check SGX debug feature settings
[+] SGX debug mode is disabled

[+] All SGX checks passed

OR

[!] Intel SGX instructions disabled by firmware
```

```
OR

[*] NOT IMPLEMENTED: CPU Debug features are not supported on this platform
Skipping module chipsec.modules.debugenabled since it is not supported in this platform
```