Utilisation de Chipsec pour valider la sécurité de plate-formes matérielles

Yves-Alexis Perez

Agence nationale de la sécurité des systèmes d'information

SSTIC 2020

Introduction & présentation

Qui sommes nous ?

Yves-Alexis Perez

- chef de laboratoire à l'ANSSI
- chercheur en sécurité

Arnaud Malard

Chercheur en sécurité

Thomas Letan

Ancien membre du laboratoire

Laboratoire Architectures matérielles et logicielles

- processeurs et extensions de sécurité
- plate-formes et périphériques
- systèmes embarqués
- micro-noyaux et hyperviseurs
- systèmes d'exploitations et distributions
- écosystèmes mobiles

Projets

CLIP OS, WooKey, LandLock, CamlCrush...

Déroulé

2. Contexte

Point de départ

2015

- détection de trafic réseau lié à Computrace dans une administration française
- identification d'un agent Computrace sur des postes récemment acquis et installés

Computrace

- module anti-vol reposant sur un composant BIOS[a] et un agent système
- Verrouillage de la plate-forme en cas de suspicion de vol
- Fonctionnalités de contrôle à distance avec privilèges élevés
- Kaspersky a publié deux rapports sur des vulnérabilités Computrace en 2009 et 2014[7])
- maintenant renommé *Persistence Technology, from Absolute*

[a] **sous forme d'Option ROM**

Investigation

Identification du problème

- Document Lenovo de décembre 2015 : *Unintended Computrace activation*[8]
- Certains BIOS Lenovo ont Computrace activé par défaut
- L'administration française a acquis des portables Lenovo en 2015
- Certains faisaient partie du lot problématique
- L'agent système a été déployé sur ces postes
- Ces postes ont ensuite servi à générer des master pour le reste

Réponse à incident

Court terme

- coordination avec Lenovo
- identification des portables problématiques et application du correctif

Long terme

- s'assurer que le matériel est aussi maintenu en conditions de sécurité
- améliorer l'application des mises à jours (souvent problématiques sur le matériel)
- identifier des fonctionnalités de sécurité ou de durcissement côté matériel
- s'assurer que les matériels acquis par l'administration suivent les bonnes pratiques
- augmenter le niveau de sécurité des constructeurs

Rédaction d'exigences de sécurité

Préparation

- identification d'exigences utiles, pertinentes et applicables
- coordination avec les constructeurs et les DSI afin de les valider
- mise en place de moyens de tests au moment de l'acquisition

Publication sur le site de l'ANSSI[2]

- Maîtrise de la plate-forme
- Caractéristiques matérielles
- Caractéristiques firmware
- Maintien en condition de sécurité

Détails

Maîtrise de la plate-forme

- s'assurer que le propriétaire maîtrise la plate-forme et non Intel, Microsoft ou le constructeur
- avoir une liste complète des divers composants en particulier les périphériques communicants
- avoir une liste complète des firmwares et leur version

Caractéristiques matérielles

- s'assurer de la présence de composants intéressants : I/OMMU, TPM etc.

Détails (2)

Caractéristiques firmware

- s'assurer que le propriétaire peut configurer les fonctions de sécurité habituelles
  changement de l'ordre de démarrage, mots de passe d'accès au BIOS etc.
- s'assurer que le firmware dispose de protections standards
  vérification d'intégrité, flash en lecture seule etc.
- éviter les solutions d'administration à distance activées par défaut
- s'assurer que le BIOS configure correctement la plate-forme

Maintien en conditions de sécurité

- s'assurer que les constructeurs fournissent des mises à jours pour tous les firmwares BIOS/UEFI mais aussi Intel CSME, TPM, firmwares des périphériques
- s'assurer qu'un constructeur n'impose pas un OS particulier pour les mises à jour

Validation

Comment recetter les exigences ?

Les constructeurs fournissent

- de la documentation (liste des composants, firmwares etc.)
- des machines de test

L'ANSSI réalise ensuite des tests sur les machines fournies

- Principalement pendant la consultation
  - chaque machine présélectionnée est testée
  - les offres non conformes sont rejetées
- Les changements de configuration sont de nouveau testés
- Possibilité de faire des tests échantillonnés plus tard

Moyens de tests

Configuration de la plate-forme

- Intel recommande fortement le paramétrage précis du CPU/Chipset par le BIOS
- Une mauvaise configuration peut entraîner des vulnérabilités

Procédure pratique

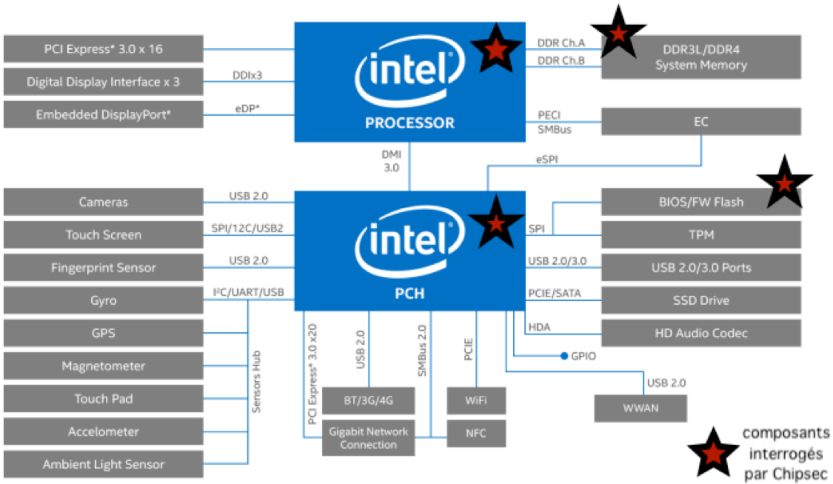
Sur la machine à tester

- vérification des fonctionnalités de sécurité BIOS (mots de passe..)
- démarrage d'une clé USB sur base Chipsec pour vérifier la configuration
- changement des clés SecureBoot et vérification du bon fonctionnement

Tous les outils et la procédure sont disponibles en ligne[1]

3. Composants matériels à contrôler

Configuration d'une plate-forme



CPU et modes de fonctionnement

CPL/Ring

3 pour le mode utilisateur, non privilégié

0 pour le mode noyau, privilégié

SMM : System Management Mode

- totalement isolé du reste du système
- déclenché via interruption matérielle (SMI)
- code fourni par le BIOS

PCH

Platform Controller Hub : chipset

- interface entre CPU et périphériques
- contient de multiples contrôleurs de périphériques
- relié à divers bus (USB, SPI, I^2C etc.)
- les contrôleurs sont eux même des périphériques PCIe
- exposent leur configuration via PCIe

Mémoires

Mémoire physique

- espace addressable par le CPU
- couvre la RAM mais aussi les périphériques
- routage des données dans le contrôleur

RAM

- héberge données utilisateurs, code exécuté par le CPU, configuration
- contrôleur mémoire arbitre les accès à la RAM depuis le CPU

Flash SPI

- code et données du BIOS
- code et données d'autres composants (Intel CSME, Ethernet..)
- accès via un contrôleur SPI hébergé dans le PCH

4. Accès à la configuration des composants

Registres de configuration

Qu'est-ce ?

- espace physique de petite taille (4-8o)
- exposé par un matériel (CPU, contrôleur, périphérique)
- stocke des données de configuration qui peuvent changer le comportement

Importance en terme de sécurité

directe passage en lecture seule de la flash SPI

indirecte verrouillage d'un autre registre

Provenance de la configuration

- valeur par défaut du matériel
- **configuration par le BIOS**
- configuration par l'OS

Type d'accès

Registres CPU

- retours de l'instruction CPUID
- *Model specific registers*

PCI Configuration space

- nombreux matériel exposés sur le bus PCIe
- PCIe prévoit un espace de configuration (256-4096o) exposé par les périphériques
- accès via PMIO (instructions in/out) ou MMIO (accès mémoire décodé par le chipset)

Autres accès MMIO

- mémoire exposée par des périphériques
- adresse via les PCIe BAR, ACPI, stockée en dur...

5. Vulnérabilités autour du matériel

Conséquences des vulnérabilités

Exécution de code bas niveau

Le « matériel » dispose souvent de privilèges élevés

Persistance

L'ajout d'implant dans un matériel est immune à une réinstallation standard

Furtivité

Le « matériel » n'est en général pas accessible directement/facilement

Exemple de vulnérabilité impactant la flash SPI

Flash ouverte en écriture

`BIOS_CNTL[BIOSWE]==1`

Mise à jour de BIOS

- le système (noyau) écrit dans la flash

Exploitation de vulnérabilité

- un code malveillant privilégié (ring0) écrit dans la flash

Exemple de vulnérabilité impactant la flash SPI

Flash en lecture seule

`BIOS_CNTL[BIOSWE]==0`

Mise à jour de BIOS

- `BIOS_CNTL[BIOSWE]=1`
- écriture dans la flash
- `BIOS_CNTL[BIOSWE]=0`

Exploitation de vulnérabilité

- `BIOS_CNTL[BIOSWE]=1`
- écriture dans la flash

Exemple de vulnérabilité impactant la flash SPI

Verrouillage de BIOSWE

`BIOS_CNTL[BLE]`

- 0 modification possible de `BIOSWE`
- 1 la modification de `BIOSWE` déclenche une SMI

Mise à jour de BIOS

ring0 `BIOS_CNTL[BIOSWE]=1`

SMM vérification de la mise à jour

SMM écriture dans la flash

SMM `BIOS_CNTL[BIOSWE]=0`

Attaque SpeedRacer[6]

Tentative de race condition

- écriture dans la flash depuis le ring 0 autorisée quand `BIOS_CNTL[BIOSWE]==1`
- remise à zéro effectuée par du code en mode SMM

En boucle sur plusieurs cœurs

ring0 `BIOS_CNTL[BIOSWE]=1`

ring0 écriture dans la flash

Plate-forme

SMM vérification de la mise à jour

SMM `BIOS_CNTL[BIOSWE]=0`

Attaque Sandman[5]

Désactivation des SMI

- écriture dans la flash depuis le ring 0 autorisée quand `BIOS_CNTL[BIOSWE]==1`
- remise à zéro effectuée par du code en mode SMM

Au préalable

ring0 désactivation des SMI

ring0 `BIOS_CNTL[BIOSWE]=1`

Plate-forme

SMM Jamais appelé

Attaque

ring0 écriture dans la flash

Parade à Sandman

SMM Bios Write Protect (`BIOS_CNTL[SMM_BWP]`)

- 0 écriture SPI possible depuis le `ring 0`
- 1 écriture SPI possible uniquement depuis SMM

Augmente le niveau nécessaire à l'attaquant

- écriture SPI uniquement possible par du code SMM
- code SMM fourni par le BIOS
- autre attaque nécessaire :
  - exécution de code arbitraire SMM (en plus du ring 0)
  - accès physique à la flash

Vulnérabilités SMRAM

System management RAM

- stocke le code et les données du SMM
- espace mémoire pris dans l'espace physique adressable
- accès contrôlé en plus de la MMU

`SMRAMC` *System Management Ram Control Register*

`D_OPEN` *SMM Space Open*

- 0 non-SMM access to SMRAM routed to system bus (VGA space)
- 1 non-SMM access to SMRAM routed to RAM

`D_LCK` *SMM Space Locked*

- 0 `SMRAMC` modifiable
- 1 `SMRAMC` verrouillé

Exploitation

2006[9] si SMRAMC[D_LCK] == 0

ring0 SMRAMC[D_OPEN]=1

ring0 modification d'un SMI handler

SMM exécution de code arbitraire

2009 Duflot[11] et ITL[12], contournement de D_LCK

ring0 modification de la stratégie de cache de la SMRAM en *write-back*

ring0 déclenchement d'une SMI

ring0 écrasement d'un SMI handler dans le cache (non protégé par SMRAMC)

ring0 déclenchement d'une SMI

SMM exécution du SMI handler modifié

Vulnérabilités concernant les CPU

Transient execution attacks[3]

2017 Meltdown, Spectre

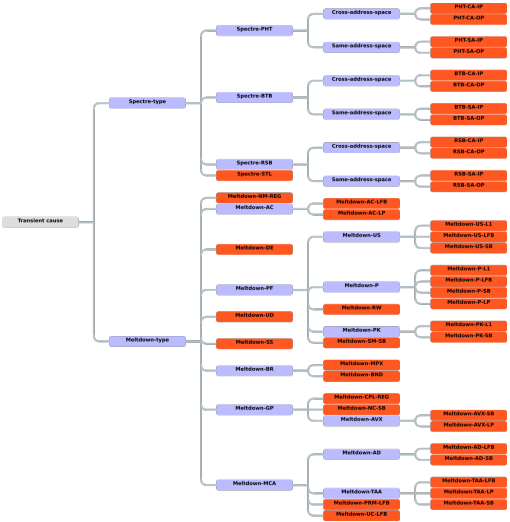
2018 Foreshadow

2019 Fallout, RIDL, Zombieload

2020 Cacheout

Contremesures

- IBRS, IBPB, IBRS_ALL, ...
- exposées par le processeur (cpuid, MSR)
- prise en compte par OS ou hyperviseur



6. Contrôle manuel

Comment s'assurer de la sécurité d'une plate-forme

Configuration exposée dans des registres

- à quel endroit ?
- avec quelle sémantique ?

Documentation ?

- pas toujours accessible
- quand elle existe elle peut être très dense
- information noyée
- nombreuses dépendances

Accès aux registres

Emplacements

- PCIe : PCI configuration space
- Projections mémoire (MMIO)
- Registre CPU

Privilèges

- au moins administrateur (root)
- plus probablement *ring0*
- parfois pas possible sur un système standard (*lockdown* sous Linux)

Identification de la plate-forme

- `/proc/cpuinfo` : Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz
- `https://ark.intel.com`, 8550U, datasheet, I/O datasheet vol. 2

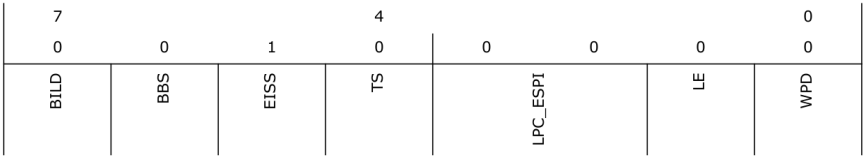
2.1.21 BIOS Control (BC)—Offset DCh

Access Method

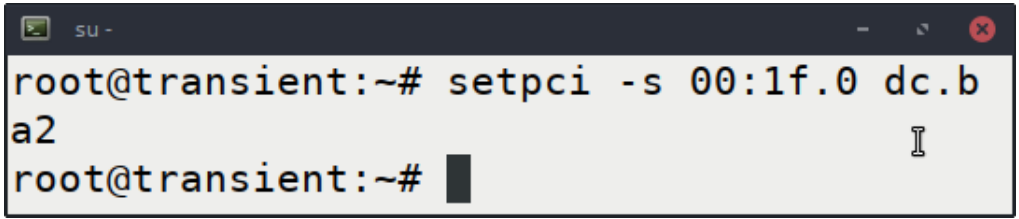
Type: CFG Register
(Size: 8 bits)

Device: 31
Function: 0

Default: 20h



Vérification

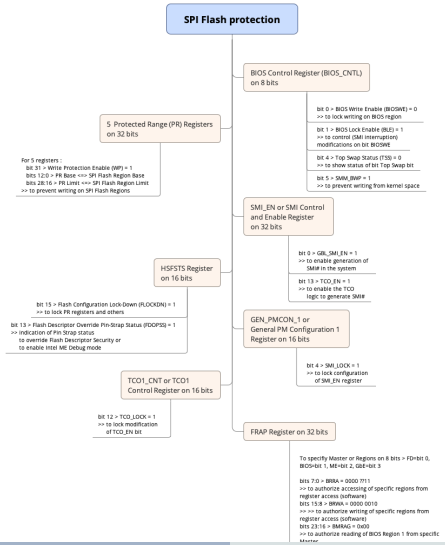


1	0h RW/1L	**Lock Enable (LE):** When set, setting the WP bit will cause SMI. When cleared, setting the WP bit will not cause SMI. Once set, this bit can only be cleared by a PLTRST#. When this bit is set, EISS - bit [5] of this register is locked down.
0	0h RW	**Write Protect Disable (WPD):** When set, access to the BIOS space is enabled for both read and write cycles to BIOS. When cleared, only read cycles are permitted to the FWH or SPI flash. When this bit is written from a 0 to a 1 and the LE bit is also set, an SMI# is generated. This ensures that only SMM code can update BIOS.

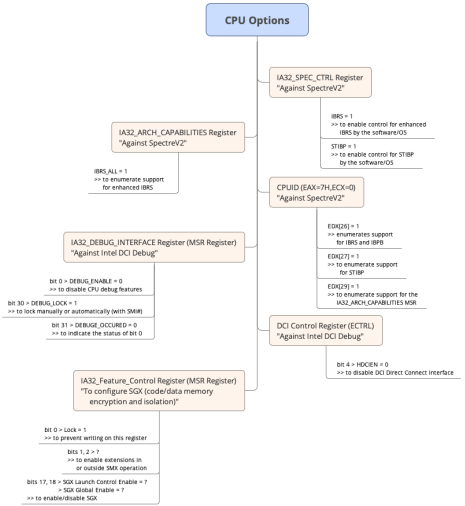
Interprétation des valeurs extraites

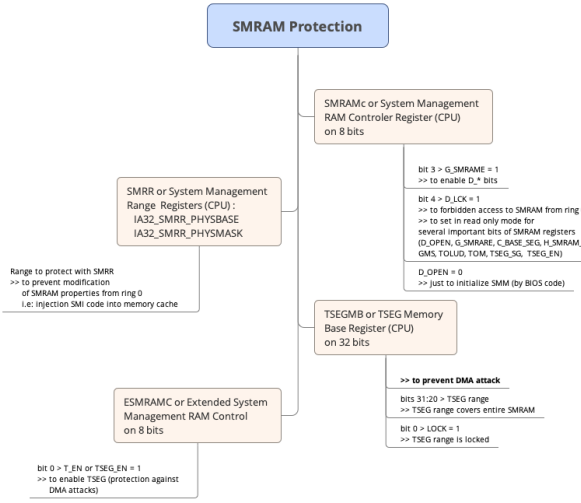
Long et fastidieux

- isoler chaque bit ou groupe de bits
- comparer les valeurs à la documentation
- passage à l'échelle compliqué
- surtout quand les plates-formes diffèrent



Mindmap


CPU Options
IA32_SPEC_CTRL Register
"Against SpectreV2"
IBRS = 1
>> to enable control for enhanced
IBRS by the software/OS
STIBP = 1
>> to enable control for STIBP
by the software/OS
IA32_ARCH_CAPABILITIES Register
"Against SpectreV2"
IBRS_ALL = 1
>> to enumerate support
for enhanced IBRS
CPUID (EAX=7H,ECX=0)
"Against SpectreV2"
EDX[26] = 1
>> enumerates support
for IBRS and IBPB
EDX[27] = 1
>> to enumerate support
for STIBP
EDX[29] = 1
>> to enumerate support for the
IA32_ARCH_CAPABILITIES MSR
IA32_DEBUG_INTERFACE Register (MSR Register)
"Against Intel DCI Debug"
bit 0 > DEBUG_ENABLE = 0
>> to disable CPU debug features
bit 30 > DEBUG_LOCK = 1
>> to lock manually or automatically (with SMM)
bit 31 > DEBUGE_OCCURED > 0
>> to indicate the status of bit 0
DCI Control Register (ECTRL)
"Against Intel DCI Debug"
bit 4 > HDCIEN = 0
>> to disable DCI Direct Connect Interface
IA32_Feature_Control Register (MSR Register)
"To configure SGX (code/data memory
encryption and isolation)"
bit 0 > Lock = 1
>> to prevent writing on this register
bits 1, 2 > ?
>> to enable extensions in
or outside SMX operation
bits 17, 18 > SGX Launch Control Enable = ?
> SGX Global Enable = ?
>> to enable/disable SGX


SMRAM Protection
SMRAMc or System Management
RAM Controler Register (CPU)
on 8 bits
bit 3 > G_SMRAME = 1
>> to enable D_* bits
bit 4 > D_LCK = 1
>> to forbidden access to SMRAM from ring
>> to set in read only mode for
several important bits of SMRAM registers
(D_OPEN, G_SMRARE, C_BASE_SEG, H_SMRAM,
GMS, TOLUD, TOM, TSEG_SG, TSEG_EN)
D_OPEN = 0
>> just to initialize SMM (by BIOS code)
SMRR or System Management
Range Registers (CPU) :
IA32_SMRR_PHYSBASE
IA32_SMRR_PHYSMASK
Range to protect with SMRR
>> to prevent modification
of SMRAM properties from ring 0
i.e: injection SMI code into memory cache
TSEGMB or TSEG Memory
Base Register (CPU)
on 32 bits
>> to prevent DMA attack
bits 31:20 > TSEG range
>> TSEG range covers entire SMRAM
bit 0 > LOCK = 1
>> TSEG range is locked
ESMRAMC or Extended System
Management RAM Control
on 8 bits
bit 0 > T_EN or TSEG_EN = 1
>> to enable TSEG (protection against
DMA attacks)

7. Contrôle assisté avec Chipsec

Chipsec

Platform Security Assessment Framework

- ensemble d'outils pour s'assurer de la sécurité d'une plate-forme
- publié à CanSecWest 2014[10], open-source[4], écrit en Python

Driver dédié (pour chaque plate-forme)

- tourne en *ring0*, proxifie les accès au matériel du code Python
- développé exprès pour ça : pas de contournement de mesures de sécurité

Fortement modulaire

- par vulnérabilités : SPI, SMRAM etc.
- par type d'accès : PMIO, MMIO etc.
- par technologie : PCIe, SPI etc.
- par famille de processeur/chipset

Configuration de Chipsec

Description XML de la plate-forme matérielle

- type d'accès aux registres
- adresses des registres

Facilement extensible

- support d'héritage
- sélection de la plate-forme via *Vendor ID* / *Product ID*
- possibilité de rajouter ses propres configurations

Support

Très bon pour Intel

- développé par des membres d'Intel
- soutenu par Intel

Plus restreint pour les autres (AMD)

- support générique des plate-formes x86
- pas de support spécifique

En pratique : pas idéal mais pas dramatique

- peu de plate-formes non Intel
- peu de vulnérabilités spécifiques AMD

Utilisation ANSSI

chipsec-check

- clé USB bootable sur base Debian
- intègre Chipsec
- scripts de collecte d'information sur la plate-forme
- outils de mise à jour de clé SecureBoot

Open-source[1]

- script de génération de la clé
- scripts de collecte d'information (lspci, lshw, TPM...)
- procédures de test
- documentation (mindmaps)

Procédure de test

Manuelle

- inspection machine et prise de photos
- analyse documentation constructeur
- vérification des options BIOS
- changement des clés SecureBoot

Assistée

- démarrage sur clé Chipsec
- scripts de collecte d'information
- `chipsec_main`
- collecte des logs

Formulaire

Vérificateur .....

Candidat .....

Référence Matériel .....

B.1 Exigences « documentaires »

Pour satisfaire aux exigences documentaires, le candidat doit fournir, sous forme de listes, les composants matériels de sa plate-forme ; pour l'aider dans sa démarche, il est conseillé de lui fournir le document décrit dans l'Annexe A.

En plus de cette liste, il convient de vérifier que sont bien fournis :

	Oui	Non
Le certificat Critères Communs du TPM	☐	☐
Les rapports d'évaluation suivant les nomes EN 55024 et EN 55022	☐	☐

B.2 Exigences « configuration »

La vérification de ces exigences est rapide dans la plupart des cas, mais peut être compliquée si le menu de configuration du BIOS n'est pas accessible facilement[1] ou si sa présentation n'est pas habituelle. Dans ces cas là, il ne faut pas hésiter à solliciter le candidat directement.

	Oui	Non
Présence d'une IO/MMU (VT-d, AMD-Vi) activée par défaut	☐	☐
Choix d'un mot de passe de démarrage	☐	☐
Choix d'un mot de passe pour le menu de configuration BIOS	☐	☐
Choix de l'ordre de sélection des périphériques au démarrage	☐	☐
Modules de gestion à distance (*AMT, Computrace, Anti-Theft*) désactivés par défaut	☐	☐
Possibilité de désactiver les interfaces de communication	☐	☐
Possibilité de désactiver les ports d'entrée/sortie (par exemple USB)	☐	☐

1. Par exemple, certains ordinateurs HP ne font pas la publicité de la séquence de touches nécessaires pour entrer dans le menu de configuration du BIOS.

Résultats

```
[CHIPSEC] **************************  SUMMARY  **************************
[CHIPSEC] Time elapsed            0.223
[CHIPSEC] Modules total           25
[CHIPSEC] Modules failed to run   0:
[CHIPSEC] Modules passed          20:
[+] PASSED: chipsec.modules.common.bios_kbrd_buffer
[+] PASSED: chipsec.modules.common.bios_smi
[+] PASSED: chipsec.modules.common.bios_ts
[+] PASSED: chipsec.modules.common.bios_wp
[+] PASSED: chipsec.modules.common.ia32cfg
[+] PASSED: chipsec.modules.common.me_mfg_mode
[+] PASSED: chipsec.modules.common.memlock
[+] PASSED: chipsec.modules.common.rtclock
[+] PASSED: chipsec.modules.common.secureboot.variables
[+] PASSED: chipsec.modules.common.smm
[+] PASSED: chipsec.modules.common.smrr
[+] PASSED: chipsec.modules.common.spd_wd
[+] PASSED: chipsec.modules.common.spi_desc
[+] PASSED: chipsec.modules.common.spi_fdopss
[+] PASSED: chipsec.modules.common.spi_lock
[+] PASSED: chipsec.modules.common.uefi.access_uefispec
[+] PASSED: chipsec.modules.debugenabled
[+] PASSED: chipsec.modules.memconfig
[+] PASSED: chipsec.modules.remap
[+] PASSED: chipsec.modules.smm_dma
[CHIPSEC] Modules information     1:
[#] INFORMATION: chipsec.modules.common.cpu.cpu_info
[CHIPSEC] Modules failed          0:
[CHIPSEC] Modules with warnings   4:
[!] WARNING: chipsec.modules.common.cpu.spectre_v2
[!] WARNING: chipsec.modules.common.sgx_check
[!] WARNING: chipsec.modules.common.spi_access
[!] WARNING: chipsec.modules.common.uefi.s3bootscript
[CHIPSEC] Modules not implemented 0:
[CHIPSEC] Modules not applicable  0:
[CHIPSEC] **************************************************************
root@transient:~#
```

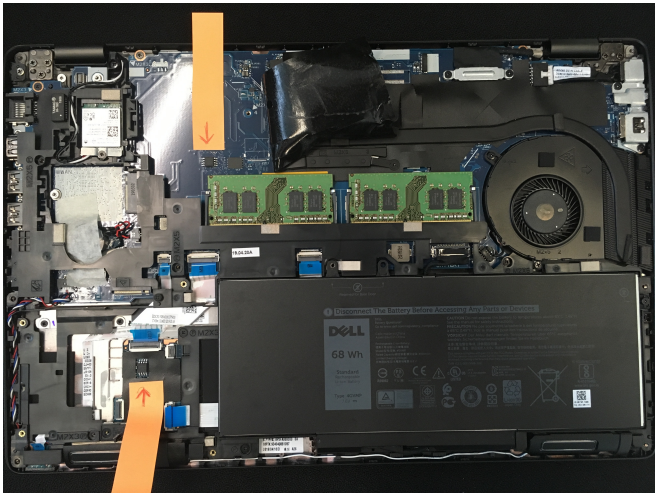
Synthèse des collectes

- vérification de chacune des exigences
- recensement des non-conformités
- dialogues avec les constructeurs

En cas de non-conformité

- investigation
- dialogues avec les constructeurs
- rejet de l'offre

En bonus



Collecte de firmware

- logiciel : `chipsec_util`
- matériel : SOIC-8

8. Possibilités futures

Nouvelles vérifications

Comportement correct du matériel

- TPM
- I/OMMU (pré-boot, post-boot)

Dialogue avec les constructeurs

Nouvelles exigences ?

- support des mises à jour via fwupd.org
- support de coreboot

Coopération

- prise de conscience en cours
- étape suivante : anticipation, R&D

Firmwares

Stockage et comparaison de firmwares

- dump logiciel
- dump matériel
- version fournie sur le site du constructeur
- versions qui évoluent dans le temps
- *hunting* lors de réponses à incidents

Collecte d'autres firmwares

- périphériques embarqués (cartes réseau etc.)
- périphériques connectés (imprimantes etc.)
- serveurs, SAN...

Conclusion

Vérification matérielle compliquée

- implants bien cachés
- vérifications compliquées (documentation, dépendances, accès matériel)

Chipsec aide beaucoup

- facilité d'utilisation
- facilité d'extension
- bon support Intel, très impliqué

Espoirs pour le futur

Nouvelles vulnérabilités découvertes régulierement, mais :

- constructeurs prèts à coopérer, soucieux de la sécurité
- coopération envisageable avec d'autres pays (échelle)

The end

Questions ?

References

[1] ANSSI.
Tools for testing requirements.
Available from : https://github.com/anssi-fr/chipsec-check.

[2] ANSSI.
Exigences de sécurité matérielle pour plate-formes x86.
https://www.ssi.gouv.fr/guide/exigences-de-securite-materielles/, 2019.

[3] Claudio Canella, Jo Van Bulck, Michael Schwarz, Moritz Lipp, Benjamin von Berg, Philipp Ortner, Frank Piessens, Dmitry Evtyushkin, and Daniel Gruss.
A Systematic Evaluation of Transient Execution Attacks and Defenses.
In *USENIX Security Symposium*, 2019.
extended classification tree at https ://transient.fail/.

References (cont.)

[4] Intel.
Platform security assessment framework.
https://github.com/chipsec/chipsec.

[5] Corey Kallenberg, Xeno Kovah, John Butterworth, and Sam Cornwell.
SENTER Sandman : Using Intel TXT to Attack BIOSes.
2014.

[6] Corey Kallenberg and Rafal Wojtczuk.
Speed racer : Exploiting an intel flash protection race condition.
2015.

References (cont.)

[7] Kaspersky.
Absolute computrace revisited.
Available from :
https://securelist.com/absolute-computrace-revisited/58278.

[8] Lenovo.
Unintended computrace activation.
Available from : https://support.lenovo.com/fr/fr/solutions/ht105220.

[9] Daniel Etiemble Loic Duflot and Olivier Grumelard.
Using cpu system management mode to circumvent operating system security functions.

References (cont.)

https://www.researchgate.net/publication/241643659_Using_CPU_System_Management_Mode_to_Circumvent_Operating_System_Security_Functions, 2006.

[10] John Loucaides and Yuriy Bulygin.
Platform security assessment with CHIPSEC.
2014.

[11] Benjamin Morin Loïc Duflot, Olivier Levillain and Olivier Grumelard.
Getting into the smram : Smm reloaded.
Available from : https://cansecwest.com/csw09/csw09-duflot.pdf.

References (cont.)

[12] Joanna Rutkowska Rafal Wojtczuk.
Attacking smm memory via intel® cpu cache poisoning.
Available from :
https://invisiblethingslab.com/resources/misc09/smm_cache_fun.pdf.