Semestre SILM : livre blanc

Guillaume Hiet

Semestre SILM : livre blanc

Guillaume Hiet

20 décembre 2020

Table des matières

Acronymes

ALU	Arithmetic–Logic Unit (unité arithmétique et logique). 34
AMT	Active Management Technology. 21
API	*Application Programming Interface* (interface de programmation). 63, 67
ASIC	*Application-Specific Integrated Circuit* (circuit intégré spécialisé). 65
ASLR	Address Space Layout Randomization (distribution aléatoire de l'espace d'adressage). 25, 38, 39, 42, 46, 56
BIOS	Basic Input Output System. 10
BMC	Baseboard Management Controller. 22
BTI	Branch Target Indicators. 26
CAT	Cache Allocation Technology. 38, 55
CET	Control-flow Enforcement Technology. 26
CFI	*Control Flow Integrity* (intégrité du flux de contrôle). 6, 17, 22, 23, 26, 28, 30, 60
CSME	Converged Security and Management Engine. 21–23, 52
DCA	Direct Cache Access. 50
DCI	Direct Connect Interface. 21
DFI	Data Flow Integrity. 29
DIFT	Dynamic Information Flow Tracking (suivi dynamique des flux d'information). 20, 22–24, 29
DMA	Direct Memory Access (accès direct à la mémoire). 14, 60
DRAM	Dynamic Random Access Memory (mémoire dynamique à accès aléatoire). 17, 18, 34, 44, 45, 50, 51, 60

DRM Digital Rights Management (gestion des droits numériques). 17, 21

ECC Error-Correcting Code (code correcteur d'erreurs). 50, 51

ETM Embedded Trace Macrocell. 24, 30, 31

FIDO Fast IDentity Online. 17

FPGA *Field-Programmable Gate Array* (réseau de portes programmables *in situ*). 8, 16, 32, 65

GPU Graphics Processing Unit. 22

HDL *Hardware Description Language* (langage de description du matériel). 33, 65

IBPB Indirect Branch Predictor Barrier. 59

IBRS Indirect Branch Restricted Speculation. 59

ISA *Instruction Set Architecture* (jeu d'instructions). 24–26, 29–32, 52, 63–65, 68

ISR Instruction Set Randomization. 30

LLC Last Level Cache. 36–38, 49, 50, 55

MAC Message Authentication Code. 28

MDS Microarchitectural Data Sampling. 42

MMU Memory Managment Unit. 38, 39

MPK Memory Protection Keys. 25–27, 42

MPX Memory Protection Extensions. 27, 28

MTE Memory Tagging Extension. 28

OS *Operating System* (système d'exploitation). 6, 10–13, 16–20, 22, 25–28, 30, 31, 39, 42, 43, 45–47, 51, 53, 55, 56, 59, 60, 64–68

PA Pointer Authentication. 28

PAC Pointer Authentication Code. 28, 29

PAN Privilege Access Never. 25

PCH Platform Controller Hub. 21

PCI Peripheral Component Interconnect. 22

PSP Platform Security Processor. 21

PTM Program Trace Macrocell. 24, 30, 31

PXN Privilege Execute Never. 25, 31

RAPL Running Average Power Limit. 39
REE Rich Execution Environment. 17, 18
ROP Return Oriented Programming. 16, 25–27, 29, 30, 41, 46, 56

SDK Software Development Kit (kit de développement logiciel). 19, 46
SE *Secure Element* (élément sécurisé). 66, 67
SEP Secure Enclave Processor. 20, 21
SGX Software Guard Extensions. 11, 16, 18–20, 39, 42–48, 51, 54–56
SMAP Supervisor Mode Access Prevention. 25
SMEP Supervisor Mode Execution Prevention. 25, 31
SMI System Management Interrupt. 11, 12
SMM System Management Mode. 9, 11, 12, 22, 23, 32, 68
SMRAM System Management RAM. 11
SMT Satisfiability Modulo Theories. 33
SoC System on a Chip (système sur une puce). 17, 18, 28, 30, 49, 50
STIBP Single Thread Indirect Branch Prediction. 59

TCB Trusted Computing Base (base de confiance). 14, 18–20, 45
TEE *Trusted Execution Environment* (environnement d'exécution de confiance). 12, 16–18, 20, 24, 45–48, 66–68
TLB Translation Lookaside Buffer. 38
TOCTOU Time-Of-Check to Time-Of-Use. 45
TPM Trusted Platform Module. 9–12, 21
TRR Target Row Refresh. 51
TSX Transactional Synchronization Extensions. 19, 37, 38, 56
TXT Trusted Execution Technology. 10, 11, 15, 16, 18

UEFI *Unified Extensible Firmware Interface* (interface extensible et unifiée du micrologiciel). 10, 21, 25, 44, 68
USB Universal Serial Bus. 21, 52

VM Virtual Machine (machine virtuelle). 12, 14–16
VMI *Virtual Machine Introspection* (introspection de machines virtuelles). 15, 62

Chapitre 1

Introduction

La sécurité des composants logiciels et celle des composants matériels ont souvent été considérées comme deux problèmes spécifiques, abordés par des communautés distinctes. Toutefois, il apparaît de plus en plus important de combiner les aspects logiciels et matériels afin de prendre en compte les nouvelles attaques logicielles. Par exemple, des vulnérabilités matérielles comme Spectre ou Meltdown peuvent être exploitées par des attaques purement logicielles. De telles attaques peuvent être exécutées à distance et ne nécessitent pas d'accès physique à la plateforme matérielle ciblée. Il est donc nécessaire d'étudier de manière approfondie la sécurité des interfaces logicielles/matérielles, tant du point de vue des attaques que des moyens de défense. Cette étude soulève un certain nombre de défis.

Tout d'abord, les plateformes matérielles sont de plus en plus complexes et les spécifications de leur différents composants matériels ne sont pas toujours publiquement accessibles. En outre, ces spécifications sont parfois incomplètes, incorrectes ou ambiguës. Des approches ont donc été proposées par différents travaux de recherche afin de déterminer le fonctionnement et l'état de ces composants matériels avant de pouvoir les utiliser ou les analyser.

Évaluer le niveau de sécurité qu'offrent ces plateformes matérielles face aux attaques logicielles constitue également un défi important. L'étude des vulnérabilités existantes ou nouvelles qui peuvent affecter les plateformes matérielles est cruciale. Il convient également d'analyser les attaques logicielles qui peuvent exploiter de telles vulnérabilités afin d'évaluer leur facilité de mise en œuvre et leur criticité.

Face à ces menaces, il convient enfin d'étudier les mécanismes de défense que l'on peut mettre en place. En effet, les plateformes récentes embarquent de plus en plus de mécanismes de sécurité au sein de leurs composants matériels. Cela soulève un certain nombre de questions intéressantes : Quels types de mécanismes de sécurité peuvent être implémentés dans les plateformes matérielles à un coût acceptable ? Quelles sont précisément les propriétés de sécurité qui sont garanties par de tels mécanismes ? En outre, des contremesures purement logicielles peuvent être déployées dans les systèmes d'exploitation et les applications exécutés sur les plateformes matérielles afin de les protéger contre des attaques logicielles qui exploiteraient des vulnérabilités matérielles.

Le semestre thématique SILM fut dédié à la sécurité des interfaces logiciel/matériel et aux différents défis listés précédemment. Nous nous sommes spécifiquement intéres-

sés aux problématiques de sécurité qui combinent des aspects logiciels et matériels, par exemple :

— L'exploitation de vulnérabilités matérielles par des attaques purement logicielles ;
— L'utilisation de protections logicielles pour lutter contre l'exploitation de vulnérabilités matérielles ;
— L'utilisation de protections matérielles pour lutter contre l'exploitation de vulnérabilités logicielles ;

Les travaux qui se focalisent essentiellement sur l'un des deux domaines, matériel ou logiciel, ne relèvent pas, *a priori* du périmètre de notre étude. Ainsi, les attaques logicielles exploitant uniquement des vulnérabilités logicielles, par exemple un logiciel malveillant exploitant une vulnérabilité de type *buffer overflow*, ne sont pas considérées. Il en est de même des techniques de protection purement logicielles contre ces attaques, par exemple, l'utilisation de canaris au niveau du compilateur ou le placement aléatoire des zones mémoires par l'OS (*Operating System*, système d'exploitation). Il en est de même des attaques ou des contre-mesures purement matérielles.

Toutefois, les techniques que peuvent mettre en œuvre les approches purement matérielles ou logicielles peuvent parfois être transposées dans un contexte hybride, combinant des aspects logiciels et matériels, tant du point de vue de l'attaque que de la défense. Des techniques de défense contre les vulnérabilités logicielles, comme le CFI (*Control Flow Integrity*, intégrité du flux de contrôle), sont classiquement implémentées de manière purement logicielle mais peuvent également être implémentées en partie de manière matérielle. De même, des attaques matérielles, comme l'injection de fautes ou les attaques par canaux auxiliaires, peuvent également être réalisées de manière logicielle. Ces différents cas de figure tombent naturellement dans le périmètre de notre étude.

Nous nous sommes plus particulièrement intéressés aux trois axes de recherche suivants :

1. Analyser le comportement et l'état des composants matériels en utilisant des mécanismes de trace, des techniques de *fuzzing*, de rétro-conception ou d'analyse des canaux auxiliaires ;
2. Étudier les vulnérabilités matérielles et les attaques logicielles qui peuvent les exploiter : par exemple les attaques par canaux auxiliaires, par injection de fautes ou celles exploitant des comportements non spécifiés ;
3. Détecter et prévenir les attaques logicielles en utilisant des composants matériels dédiés. Proposer des contre-mesures logicielles permettant de se protéger des vulnérabilités matérielles.

Les principaux événements organisés durant ce semestre thématique sont les suivants :

— L'école d'été SILM s'est déroulée du 8 au 12 juillet 2019 au centre Inria Rennes Bretagne Atlantique et sur le campus de Rennes de CentraleSupélec en coopération avec le GDR Sécurité Informatique du CNRS ;
— Le Workshop SILM 2019, a été organisé durant la European Cyber Week du 20 au 21 novembre 2019 au centre de congrès de Rennes ;
— Le séminaire SILM, est organisé depuis octobre 2019 au centre Inria Rennes Bretagne Atlantique ;

— Une Master Class, une présentation sur le stand du Ministère des armées et une démonstration sur le stand d'ALLISTENE ont été organisées au Forum International de la Cybercriminalité du 28 au 30 janvier 2020 à Lille ;
— Le Workshop SILM 2020, co-localisé avec la conférence IEEE Security and Privacy Europe, devait avoir lieu du 16 au 18 juin 2020 à Gênes. La conférence a été reportée au 7-11 septembre 2020 en raison de la pandémie de Covid-19. Notre workshop a été reporté au 12 septembre 2020 et s'est finalement tenu en visio-conférence, compte tenu de la situation sanitaire.

Les informations concernant les différents événements organisés durant ce semestre thématique sont accessibles à partir du site web du semestre : https://silm.inria.fr/. Le site permet notamment d'accéder aux supports de présentation et aux enregistrements vidéo des présentations, lorsque les auteurs nous ont donné leur accord pour la diffusion de ces documents.

Le présent document dresse un bilan de ce semestre. Le chapitre 2 présente un état de l'art du domaine au travers des trois axes identifiés précédemment. Le chapitre 3 liste les principales perspectives scientifiques et technologiques du domaine. Enfin, le chapitre **??** liste les principaux acteurs travaillant sur cette thématique, dans les domaines académiques et industriels.

Chapitre 2

État de l'art

Nous présentons dans ce chapitre un rapide état de l'art du domaine. Nous présentons dans un premier temps, en section 2.1 les approches qui utilisent des fonctionnalités matérielles pour détecter ou se prémunir de l'exploitation de vulnérabilités logicielles. Puis nous présentons en section 2.2 les différentes attaques logicielles qui exploitent des vulnérabilités matérielles, au niveau de la microarchitecture. Enfin, nous évoquons les contre-mesures logicielles qui peuvent être déployées pour détecter ou se prémunir de ces attaques, en section 2.3.

2.1 Support matériel pour la sécurité logicielle

Plusieurs travaux académiques se sont intéressés à utiliser un support matériel pour détecter ou prévenir les attaques logicielles. Certains utilisent des fonctionnalités existantes des processeurs disponibles sur le marché, d'autres proposent de nouvelles fonctionnalités matérielles en validant leurs approches par simulation ou en les implémentant sur FPGA (*Field-Programmable Gate Array*, réseau de portes programmables *in situ*). Cette tendance concerne également les produits disponibles sur le marché. Les différents fabricants de microprocesseurs et d'ordinateurs sont en effet de plus en plus enclins à ajouter des fonctionnalités de sécurité matérielles au sein de leurs produits. Les intérêts par rapport aux approches purement logicielles sont multiples [1] :

- les fonctionnalités matérielles ne peuvent être modifiées, ce qui les rend plus robustes face aux attaques logicielles ;
- une meilleure efficacité du temps d'exécution et de la consommation énergétique ;
- une interface réduite et contrôlée, qui permet de mieux protéger les secrets.

Toutefois, ces fonctionnalités ne sont pas toujours immunisées face aux attaques logicielles, notamment celles exploitant les canaux auxiliaires, présentées en section 2.2. En outre, l'immutabilité des fonctionnalités matérielles est également une faiblesse qui limite les possibilités de mise à jour des mécanismes de sécurité, en cas de découverte de vulnérabilités matérielles. Des travaux se sont récemment intéressés à cette limitation et définissent une architecture matérielle reconfigurable, permettant de s'adapter dynamiquement à la menace [2].

Différentes approches ont été proposées dans la littérature, implémentant différents types de mécanismes de sécurité, au sein même du processeur [3] où à l'aide d'un co-processeur

externe, isolé du processeur principal. Zhao *et al.* dressent un état de l'art de ces différents mécanismes, des propriétés qu'ils garantissent et des vulnérabilités qui les affectent, en les catégorisant suivant différents niveaux d'abstraction [1].

De nombreux travaux ont proposé des approches permettant de garantir une propriété d'isolation. Ces travaux partent du constat que les logiciels sont de plus en plus complexes et qu'il est difficile de s'assurer qu'ils ne contiennent pas de vulnérabilités exploitables par un attaquant. En cas de compromission, l'attaquant a potentiellement accès à l'ensemble des données de l'application. Pour limiter l'impact des intrusions et faciliter l'analyse des vulnérabilités, il convient d'appliquer le principe de séparation des tâches en découpant le logiciel en modules indépendants qui peuvent communiquer entre eux via une interface limitée et contrôlée.

L'isolation d'un module consiste à contrôler l'exécution des autres composants logiciels afin de s'assurer qu'ils ne pourront lire ou modifier ses données et son code. Zhang *et al.* ont réalisé un état de l'art des différentes approches permettant d'isoler matériellement des environnements d'exécution [4]. Ils identifient les différents cas d'usage légitimes de ces environnements isolés : la surveillance du système à l'exécution, l'analyse post-mortem de la mémoire, l'analyse dynamique des logiciels malveillants et l'exécution de tâches critiques pour la sécurité. Ils mentionnent également les cas d'usage illégitimes qui consistent à implanter des logiciels malveillants dans ces environnements isolés afin de les rendre difficilement détectables et persistants. Ils abordent enfin les vulnérabilités qui affectent ces mécanismes, notamment les mécanismes d'isolation.

En outre, un certain nombre de travaux se sont intéressés à l'informatique de confiance (*trusted computing*). Il s'agit de mettre en place une racine de confiance (*Root of Trust*) permettant de mesurer ou de vérifier l'intégrité des composants logiciels, de prouver cette intégrité à un tiers (attestation à distance) ou de conditionner le déchiffrement de données du système à la vérification d'intégrité. Ces approches sont notamment promues par le Trusted Computing Group, un consortium d'industriels, et ont donné lieu à la spécification et à la réalisation du TPM (Trusted Platform Module). Il s'agit originellement d'un composant matériel isolé réalisant des opérations cryptographiques pour implémenter les fonctions listées précédemment. Il permet notamment de vérifier et d'attester l'intégrité d'un système. Depuis la version 2 du standard, ces fonctionnalités peuvent également être réalisées par du logiciel exécuté sur le processeur principal, en s'appuyant sur des mécanismes d'isolation fournis par le processeur. Maene *et al.* ont publié un état de l'art des différentes architectures matérielles qui ont été proposées de manière plus large dans les milieux académiques et industriels pour mettre en place cette informatique de confiance et garantir notamment les propriétés d'isolation et d'attestation [5].

Nous détaillons par la suite les principaux travaux académiques et industriels visant à utiliser un support matériel pour protéger le logiciel. Nous évoquons dans un premier temps les travaux s'intéressant à l'informatique de confiance et s'appuyant sur le TPM (section 2.1.1). Nous abordons ensuite les mécanismes d'isolation implémentés au sein des microprocesseurs, comme le SMM (System Management Mode) (section 2.1.2), les mécanismes de virtualisation (section 2.1.3) ou les enclaves (section 2.1.4) ainsi que l'isolation par l'utilisation d'un co-processeur dédié (section 2.1.5). Des mécanismes de sécurité spécifiques ont également été proposés pour lutter contre l'exploitation de vulnérabilités logicielles, notamment celles issues de la gestion de la mémoire. La section 2.1.6 liste les approches qui implémentent ces mécanismes au niveau du jeu d'instructions et la sec-

tion 2.1.7 présente les approches utilisant les mécanismes de traces et les compteurs de performances. Enfin, plusieurs travaux s'intéressent à définir formellement les mécanismes de sécurité matériels et les propriétés de sécurité qu'ils sont censés garantir. L'objectif est de prouver *in fine* que les implémentations sont conformes à la spécification et garantissent effectivement les propriétés souhaitées.

2.1.1 *Trusted Platform Module* et *late launch*

2.1.1.1 Implémentation de services de sécurité à l'aide du TPM

Le TPM [6] est aujourd'hui présent dans la plupart des ordinateurs commercialisés. Il est notamment utilisé par Microsoft [7] pour stocker des clés cryptographiques, réaliser le scellement conditionnel des clés cryptographiques utilisées pour chiffrer le disque dur (*BitLocker Drive Encryption*), mesurer l'intégrité des composants logiciels (hyperviseur, noyau de l'OS, périphériques, éléments de configuration, etc.) au démarrage (*Measured Boot*) et fournir un service d'attestation à distance (*Health Attestation*).

Des chercheurs d'IBM ont également démontré qu'il était possible d'implémenter des services similaires sous Linux [8]. Aujourd'hui, le support du TPM a été intégré au chargeur de démarrage GRUB qui peut ainsi mesurer l'intégrité de la configuration du chargeur de démarrage, du noyau et de ses options [9]. IBM a développé *Integrity Measurement Architecture*, une solution complémentaire qui permet de mesurer, de vérifier et d'attester l'intégrité des fichiers d'un système Linux, après le démarrage du noyau [10].

Des travaux se sont intéressés à l'implémentation du TPM de manière logicielle, ce qui permet de le virtualiser et d'offrir ses services à différentes machines virtuelles [11]. Cette approche, qui est maintenant compatible avec les dernières spécifications du TPM, est notamment utilisée par Google dans son service *Shielded VMs* [12] au sein de son service Google Cloud.

Les précédentes solutions reposent sur une chaîne statique de vérification de l'intégrité (*Static Root of Trust for Measurement*). L'initialisation de cette chaîne est réalisée par un module du *boot firmware* [1] qui est exécuté en premier lors du démarrage et dont l'adresse est fixe. L'inconvénient de cette approche est qu'elle nécessite de mesurer tous les éléments de la chaîne de confiance depuis le démarrage de la machine (le *firmware* UEFI, les *firmware* des différents périphériques, le chargeur de démarrage, le noyau, les éléments de configuration de ces logiciels, etc.). Toute modification de l'un de ces éléments, notamment suite à une mise à jour, nécessite de mettre à jour les mesures de référence.

Pour contourner cette limitation, Intel et AMD ont proposé des mécanismes permettant d'implémenter le concept de *late launch* [13]. Ce concept consiste à initier la chaîne de confiance après la phase de démarrage de la machine, typiquement avant le chargement du noyau de l'OS. On parle alors de *Dynamic Root of Trust for Measurement*. Intel implémente ce mécanisme via l'extension TXT (Trusted Execution Technology) [14] qui propose de nouvelles instructions (`GETSEC[SENTER]`) permettant notamment d'initier dynamiquement la séquence de mesures. Cette instruction place le processeur dans un mode spécifique qui permet d'exécuter le code de mesure de manière isolée et exclusive (un seul cœur est actif). AMD propose une solution similaire intégrée à sa technologie de support

1. appelé également BIOS (Basic Input Output System) et qui implémente maintenant l'UEFI (*Unified Extensible Firmware Interface*, interface extensible et unifiée du micrologiciel)

à la virtualisation (AMD-V).

Intel fournit un code binaire spécifique permettant de mesurer la partie SMM du *firmware* (SINIT). L'utilisateur peut fournir un code réalisant la mesure du logiciel qui doit être exécuté par la suite (typiquement, le noyau de l'OS). TXT est supporté, entre autres, par Microsoft. Sous Linux, le projet tBoot [15] permet d'initier une séquence de mesures avant le démarrage du noyau. Cet outil est notamment utilisé par OpenDTeX Secure Boot [16], un projet développé par la société AMOSSYS et financé par la DGA, qui implémente des fonctionnalités de scellement conditionnel et de bannière de vérification d'intégrité sous Linux, en utilisant le TPM et les extensions Intel TXT.

Des travaux ont également proposé d'utiliser le *late launch* pour implémenter Flicker [17], un service d'enclave. Le mécanisme est alors quelque peu détourné de son but initial. Au lieu de démarrer le noyau d'un OS ou d'un hyperviseur, les chercheurs tirent profit de l'isolation fournie par les mécanismes de *late launch* pour protéger du code et des données sensibles vis-à-vis de l'application et de l'OS. Les mécanismes d'enclaves comme Intel SGX (Software Guard Extensions) sont aujourd'hui destinés à cet usage. Ces travaux ont été repris récemment pour implémenter un service original de prévention des *ransomware* disposant de privilèges administrateur. Les fichiers sont directement accessibles en lecture seule. Toute modification doit être effectuée par un composant de confiance. Ce composant implémente des modifications non-destructives qui permettent de restaurer l'état précédent d'un fichier. Les auteurs utilisent Flicker pour isoler le composant en charge de la modification des fichiers [18].

2.1.1.2 Attaques contre le TPM et Intel TXT

Intel TXT n'est pas exempt de vulnérabilités [19-24] en raison notamment de la complexité de ce mécanisme. Ces vulnérabilité sont notamment liées à l'interaction de ce mécanisme avec d'autres fonctionnalités des processeurs Intel comme le SMM ou à des défauts dans l'implémentation du code SINIT. D'autres vulnérabilités [25] sont liées à l'interaction avec le mécanisme de gestion de la mise en veille.

Les implémentations du TPM ont également fait l'objet de vulnérabilités. Récemment, des chercheurs ont démontré que certaines implémentations, dont l'implémentation logicielle de Intel (fPTM) mais également des implémentations physiques, qui sont censées être plus robustes que les implémentations logicielles et qui font l'objet d'évaluations selon les Critères Communs, sont vulnérables à des attaques par canaux auxiliaires [26]. Ces vulnérabilités peuvent être exploitées à distance et permettent de réaliser une atteinte à la confidentialité des clés privées stockées dans le TPM.

2.1.2 *System Management Mode*

Le SMM est le mode d'exécution le plus privilégié sur l'architecture x86, introduit par Intel dans le processeur Intel 386SL. Il permet d'exécuter des services privilégiés du logiciel de démarrage et d'initialisation de la plateforme (*boot firmware*), y compris après la phase de démarrage, lors de l'exécution de l'OS. Ce mode permet d'isoler ces services qui peuvent accéder à une partie dédiée de la mémoire (SMRAM (System Management RAM)), qui est uniquement accessible en SMM. Le processeur bascule dans ce mode lors d'une interruption matérielle spécifique (SMI (System Management Interrupt)) qui peut être déclenchée par le logiciel.

Ce mode n'a initialement pas été conçu pour la sécurité mais pour permettre à l'OS de déléguer au *firmware* certaines tâches très dépendantes de la configuration matérielle, comme la gestion de l'énergie. Le *firmware* est en effet développé en partie par le constructeur de la carte mère. Toutefois, ce mode a, au fil du temps, acquis une importance fondamentale dans la sécurité de la plateforme matérielle. Il permet en effet d'exécuter des services critiques comme la mise à jour du code du *firmware* ou la configuration du TPM. Seul un service exécuté en SMM peut permettre l'écriture dans la mémoire *flash* contenant le code du *firmware*.

Bien que ce mode soit destiné à l'exécution de services du *firmware*, plusieurs travaux de recherche ont proposé de tirer profit de l'isolation qu'il fournit pour isoler des fonctions de sécurité. Ainsi Reina *et al.* utilisent ce mode pour implémenter une solution d'analyse post-mortem de la mémoire [27]. Sun *et al.* proposent un système permettant à l'utilisateur de basculer entre différents systèmes exécutés sur la même machine, par exemple pour traiter des données de sensibilités différentes [28]. Ce mode a également été utilisé pour isoler des services de surveillance à l'exécution des autres composants logiciels de la plateforme (OS, application) [29, 30], des périphériques [31] ou pour réaliser des analyses dynamiques de logiciels malveillants [32].

SMM est une solution intéressante pour protéger des fonctions de sécurité. Toutefois, ce mode présente plusieurs limitations. Il est relativement complexe et n'a pas été conçu pour des besoins de sécurité. Il a d'ailleurs fait l'objet de nombreuses vulnérabilités qui ont été corrigées au fil du temps. En outre, il implique un changement de contexte qui a un impact non négligeable sur les performances. L'exécution des services en SMM préempte l'exécution de l'OS et de l'hyperviseur. Par conséquent, les services exécutés en mode SMM doivent avoir une durée limitée pour ne pas perturber le fonctionnement de l'OS et des applications. En outre, le code SMM ne peut s'exécuter que lors d'une SMI. Un attaquant peut donc empêcher l'exécution de ce code en bloquant les SMI. Il paraît aujourd'hui plus pertinent d'utiliser d'autres mécanismes d'isolation spécifiquement conçus pour l'isolation de fonctions de sécurité, notamment les fonctions de virtualisation ou les TEE (*Trusted Execution Environment*, environnement d'exécution de confiance).

2.1.3 Sécurité basée sur la virtualisation

Les techniques de virtualisation permettent de partager les ressources matérielles (processeur, mémoire, périphériques) entre plusieurs VM (Virtual Machine, machine virtuelle). Un hyperviseur (ou *Virtual Machine Monitor*) est un composant logiciel permettant de créer et d'exécuter plusieurs machines virtuelles (les systèmes invités) sur une plateforme matérielle (le système hôte). L'hyperviseur s'assure entre autre du partage des ressources matérielles du système hôte entre les différents systèmes invités. Il présente à chaque système invité une abstraction des ressources matérielles de l'hôte. Ces ressources virtuelles peuvent à leur tour être partagées entre différentes applications par un OS exécuté dans chaque système invité. On distingue classiquement deux types d'hyperviseurs :

— les hyperviseurs de type 1 (*bare-metal*) tels que Microsoft Hyper-V [2] ou VMware vSphere [3], etc. s'exécutent directement sur le matériel, le système hôte se réduit dans ce cas à l'hyperviseur et ne contient pas d'OS ;

2. https://docs.microsoft.com/fr-fr/virtualization/hyper-v-on-windows/
3. https://www.vmware.com/fr/products/vsphere-hypervisor.html

— les hyperviseurs de type 2 tels que VirtualBox [4], QEMU [5] ou VMware Workstation Player [6] sont des applications qui s'exécutent au dessus de l'OS du système hôte.

Dans un système virtualisé, l'abstraction du matériel présentée par l'hyperviseur aux systèmes invités est très similaire à l'interface avec le matériel physique. L'OS des systèmes invités peut donc s'exécuter sans nécessiter de modification de son code. Une alternative et/ou une approche complémentaire, appelée para-virtualisation, consiste à modifier le code des OS invités pour qu'ils réalisent directement des appels à l'hyperviseur (ou *hypercall*) plutôt que de tenter d'accéder aux ressources matérielles virtualisées. Cela simplifie la tâche de l'hyperviseur et limite la dégradation des performances liées à la virtualisation. Toutefois, cela nécessite de modifier en partie le code des OS invités. Cette solution, utilisée notamment par Xen, est aujourd'hui utilisée en partie par la plupart des hyperviseurs.

On remarque que la distinction entre hyperviseurs de type 1 et 2 tend à s'estomper. En effet, les hyperviseurs de type 2 comme QEMU s'appuient généralement sur des fonctionnalités qui ont été déportées dans le code du noyau de l'OS du système hôte (par exemple, QEMU utilise KVM [7], un module du noyau Linux implémentant une partie de l'hyperviseur). Toutefois, une distinction importante qui perdure est la quantité de code (et donc la surface d'attaque) du système hôte. Dans le cas d'un hyperviseur de type 2, le système hôte est important puisqu'il contient un OS complet, l'hyperviseur et potentiellement différentes applications utilisateurs alors que dans le cas d'un hyperviseur de type 1, il se réduit au seul hyperviseur.

Les hyperviseurs de type 1 sont plutôt utilisés dans les serveurs hébergeant les environnements de type *cloud* et les hyperviseurs de type 2 sont plutôt utilisés sur des ordinateurs personnels, permettant à l'utilisateur d'exécuter différents environnements virtuels en plus des applications exécutées nativement sur le système hôte. Mais ces domaines d'utilisation ne sont pas exclusifs. Par exemple, Microsoft utilise Hyper-V (son hyperviseur de type 1) pour virtualiser des fonctions de sécurité sur les ordinateurs personnels [33].

Pour pouvoir partager les ressources matérielles, les hyperviseurs doivent pouvoir intercepter et contrôler les accès au matériel réalisés par les OS invités. Afin de faciliter ce contrôle et limiter l'impact sur les performances, les fabricants de processeurs ont proposé des extensions matérielles (Intel VT, AMD-V , etc.) que l'hyperviseur peut utiliser pour contrôler les VM. Cela se traduit par l'ajout d'un niveau de privilège supplémentaire, dans lequel s'exécute le système hôte (et donc l'hyperviseur). Ce mode permet à l'hyperviseur d'exécuter des VM dans un mode moins privilégié et d'intercepter les accès au matériel réalisés par les VM. Le processeur appelle le code de l'hyperviseur lorsqu'un tel accès est réalisé par l'OS d'un système invité. Il peut ainsi partager les ressources matérielles entre les VM.

L'hyperviseur assure également une propriété d'isolation entre les VM : un programme exécuté dans une VM (y compris un OS invité) ne peut interférer avec une autre VM (en particulier il ne peut pas lire ou modifier les zones mémoires allouées à d'autres VM). Cela fait de l'hyperviseur un des composants logiciels les plus privilégiés au sein d'une

4. https://www.virtualbox.org/
5. https://www.qemu.org/
6. https://www.vmware.com/fr/products/workstation-player/workstation-player-evaluation.html
7. https://www.linux-kvm.org/page/Main_Page

plateforme. Il s'agit donc d'un composant critique pour la sécurité [34]. Une vulnérabilité dans l'hyperviseur permet potentiellement à un attaquant de briser l'isolation et d'accéder aux ressources de toutes les VM.

On remarque par ailleurs que les hyperviseurs sont régulièrement utilisés spécifiquement pour assurer des tâches en lien avec la sécurité. Par exemple, l'analyse dynamique des *malware* se fait dans des *sandbox*, des environnements contrôlés et isolés. La plupart de ces environnements reposent sur l'utilisation d'hyperviseurs (par exemple, Cuckoo [8] supporte la plupart des hyperviseurs du marché, DRAKVUF [35] utilise Xen, etc.). L'utilisation d'un hyperviseur permet dans ce cas d'isoler le système infecté, d'analyser et de contrôler les comportements malveillants du *malware* analysé (notamment sa propagation) mais également de restaurer facilement et rapidement le système infecté dans un état sain, en tirant parti des fonctions de capture d'instantanés (*snapshoting*) fournies par la plupart des hyperviseurs.

Nous évoquons par la suite plus en détail les approches qui visent à renforcer la sécurité des systèmes virtualisés en chiffrant la mémoire des VM (section 2.1.3.1) et en réduisant la surface d'attaque de l'hyperviseur (section 2.1.3.2). Nous présentons ensuite en section 2.1.3.3 les travaux de recherche qui utilisent la virtualisation pour isoler un moniteur de sécurité afin de surveiller et protéger les systèmes invités.

2.1.3.1 Chiffrement de la mémoire des machines virtuelles

Dans un cas d'usage de type *cloud*, les VM sont déployées sur une infrastructure fournie par un tiers, qui n'est pas nécessairement de confiance. Afin de protéger la mémoire des machines virtuelles face à un hyperviseur potentiellement malveillant ou corrompu, AMD a proposé un mécanisme permettant de chiffrer la mémoire des VM [36]. L'approche s'appuie sur le co-processeur de sécurité intégré aux processeur d'AMD, qui réalise les opérations de chiffrement/déchiffrement à la volée. Il est possible de chiffrer l'intégralité de la mémoire avec une clé unique (ce qui simplifie le support logiciel) ou en associant une clé différente à chaque VM et à l'hyperviseur. Intel dispose d'une technologie similaire mais plus souple [37]. Il est en effet possible d'associer des clés à des pages et d'utiliser différentes clés au sein d'une même VM, par exemple pour isoler différentes applications ou différentes fonctionnalités au sein d'une même application.

Des travaux récents [38] ont montré que les opérations d'entré/sortie réalisant des DMA (Direct Memory Access, accès direct à la mémoire) ne pouvaient être chiffrées avec une clé de VM, dans la solution d'AMD. Si l'hyperviseur est corrompu ou malveillant, cela signifie que les données échangées avec les périphériques peuvent être lues et modifiées par l'hyperviseur. Pire, cela permet à l'hyperviseur de créer un oracle de chiffrement et de déchiffrement de la mémoire d'une VM en manipulant les données recopiées entre la zone mémoire de la VM et les pages en clair. L'implémentation d'Intel ne semble pas souffrir de cette limitation.

2.1.3.2 Réduction de la surface d'attaque de l'hyperviseur

L'hyperviseur a un rôle clé dans la mise en place de l'isolation entre les différentes VM. Il fait de fait partie de la TCB (Trusted Computing Base, base de confiance). Idéalement, sa surface d'attaque devrait être réduite au minimum et un effort important devrait être

8. https://cuckoosandbox.org/

déployé pour minimiser les vulnérabilités. Cependant, même les hyperviseurs de type 1, qui offrent une surface d'attaque plus réduite que ceux de type 2, peuvent présenter des vulnérabilités exploitables par un attaquant [9]. Ces vulnérabilités viennent en partie du fait que les hyperviseurs sont développés dans des langages bas niveau offrant peu de garanties de sécurité, comme le C ou C++ et qu'ils comprennent une quantité de code importante, réalisant des fonctionnalités complexes. Par exemple, le code source d'Hyper-V comporte environ 100 000 lignes de code écrit en C [39].

Des travaux de recherche se sont intéressés à réduire cette surface d'attaque. DeHypen [40] propose ainsi de transférer une part importante du code d'un hyperviseur intégré au noyau du système hôte (comme KVM) dans l'espace utilisateur. Ainsi, l'approche permet de réduire les privilèges pour une part importante des fonctionnalités, ce qui limite l'impact en cas de compromission. De manière similaire, CloudVisor-D [41] utilise la virtualisation imbriquée pour protéger les VM face à l'hyperviseur hôte. Afin de réduire le coût de la virtualisation imbriquée, il décompose l'hyperviseur imbriqué en plusieurs VM de filtrage qui s'exécutent en parallèle des VM utilisateurs. L'hyperviseur imbriqué est alors fortement réduit et se concentre sur la protection des VM de filtrage.

2.1.3.3 Surveillance et protection des machines virtuelles

Plusieurs travaux de la littérature utilisent la virtualisation pour protéger les systèmes invités face aux attaques. Ainsi TrustVisor [42] propose un hyperviseur minimal dans le but d'implémenter des enclaves. Ces enclaves permettant d'isoler les fonctions sensibles d'une application. L'approche est similaire à celle suivie par Flicker [17] mais les auteurs utilisent ici la virtualisation au lieu de Intel TXT pour isoler les enclaves, ce qui permet de limiter le surcoût à l'exécution lors des changements de contexte. Gateway [43] vise à protéger le code du noyau de logiciel malveillant exploitant des vulnérabilités dans les pilotes de périphériques. Pour cela, il isole les pilotes dans un espace mémoire dédié et il contrôle les interactions entre ces pilotes et le reste du noyau, via une interface spécifique. Les auteurs utilisent la virtualisation pour s'assurer que les pilotes ne puissent interagir que via cette interface.

La surveillance du comportement des systèmes invités via des techniques de VMI (*Virtual Machine Introspection*, introspection de machines virtuelles) à fait l'objet de nombreux travaux [44]. Ces techniques permettent par exemple d'analyser dynamiquement des logiciels malveillants [35, 45] ou de vérifier l'intégrité du système surveillé [46].

En plaçant le moniteur dans l'hyperviseur ou dans une VM isolée du système surveillé, on le protège des attaques visant le système surveillé. Toutefois, cette isolation crée un fossé sémantique [47] : le moniteur ne dispose pas de toutes les informations de contexte qui lui permettent d'inférer précisément l'état du système surveillé. Par exemple, il ne dispose pas de la valeur du registre pointant sur la table des pages. En outre, il doit inférer les différentes structures de données du noyau invité pour retrouver, par exemple, la liste des processus ou des fichiers ouverts.

Différentes approches ont été proposées pour réduire ce fossé sémantique [44, 47]. Par exemple SYRINGE [48] injecte des appels de fonctions dans le système surveillé pour obtenir les informations souhaitées. L'outil contrôle l'exécution de ces appels de fonctions

9. Par exemple, Hyper-V fait l'objet de plusieurs CVE : https://www.cybersecurity-help.cz/vdb/SB2019111214

depuis une VM isolée afin de s'assurer qu'un attaquant ne puisse les perturber. ShadowMonitor [46] utilise une approche similaire. Une partie du moniteur est déportée dans le système surveillé et le moniteur isolé dans la VM de contrôle s'assure que la partie déportée du moniteur n'est pas perturbé par l'attaquant. PrivWatcher [49] s'intéresse plus particulièrement à la détection des *non-control-data attacks* qui modifient les données d'un programme.

2.1.4 Enclaves sécurisées

Historiquement, les processeurs ont implémenté des mécanismes permettant d'isoler les différentes applications utilisateurs ainsi que le noyau de l'OS. Des extensions sont également présentes dans bon nombre de processeurs permettant d'isoler différentes machines virtuelles et l'hyperviseur qui les contrôle. Ces mécanismes reposent sur une hiérarchie de confiance verticale : l'isolation entre composants de niveau n est mise en place et contrôlée par un composant de niveau $n-1$, qui doit être de confiance. Cette approche nécessite de faire confiance aux composants logiciels de bas-niveau (OS, hyperviseur).

Des mécanismes d'enclaves ont également été proposés pour isoler certaines fonctionnalités d'une application, sans nécessiter de faire confiance à ces composants de bas niveau, notamment dans un cas d'usage de type *cloud computing*. Des travaux ont proposé d'utiliser des mécanismes existants des processeurs de l'époque, comme Intel TXT [17] ou la virtualisation [50].

D'autres projets ont proposé de modifier l'architecture des processeurs afin d'implémenter des mécanismes dédiés [51]. Toutefois ces travaux n'ont pu être évalués qu'en simulation. De même, Noorman *et al.* ont proposé Sancus, un mécanisme d'enclave dédié aux systèmes embarqués [52] qui ne disposent pas de mécanisme d'isolation telle que la mémoire virtuelle. Sancus vise principalement à isoler différents composants logiciels sur un même processeur en assurant l'intégrité de ces composants (la confidentialité n'est pas un objectif de ce projet). Sancus a donné lieu à la réalisation d'un prototype sur FPGA en modifiant un *softcore* MSP430. Ces approches étendent le jeu d'instruction des processeurs pour permettre la création et la gestion des enclaves. L'exécution du code d'une enclave passe par un saut à des adresses spécifiques, appelées points d'entrée. Cela permet de limiter les attaques de type ROP (Return Oriented Programming). Ces projets assurent l'isolation de la mémoire de l'enclave et fournissent des services de vérification d'intégrité et d'attestation à distance.

Ces différents concepts se retrouvent également dans les mécanismes d'enclaves disponibles maintenant dans les processeurs du marché. Nous évoquons par la suite plus en détail les projets qui reposent sur les mécanismes ARM TrustZone (section 2.1.4.1), Intel SGX (section 2.1.4.2) et les différents mécanismes d'enclaves pour les processeurs RISC-V (section 2.1.4.3).

2.1.4.1 ARM TrustZone

TrustZone [53] est un mécanisme d'isolation développé par ARM et disponible sur certains de ses processeurs (Cortex-A, utilisés typiquement dans les *smartphone* et Cortex-M, destinés aux systèmes embarqués). Les processeurs ARM étant omniprésents sur les *smartphone*, ce dispositif est essentiellement utilisé pour implémenter le concept de TEE défini par le consortium GlobalPlatform [54].

Le terme de TEE est couramment employé dans différents articles de recherche et publications marketing. Toutefois, suivant les ouvrages, les environnements ainsi nommés ne fournissent pas toujours les mêmes services. Sabt *et al.* en donnent une définition précise et dressent un état de l'art des différents projets existants [55]. Cependant, TrustZone fournit un mécanisme générique d'isolation qui peut être utilisé à d'autres fins. Par exemple, Nyman *et al.* l'utilisent pour mettre en œuvre du CFI et protéger une *shadow stack* [56].

TrustZone définit deux modes de fonctionnement pour un cœur ARM [57] :

— le *non Secure World* correspond au mode d'exécution classique, dans lequel s'exécute l'OS applicatif (par exemple Android) et les applications utilisateurs ;
— le *Secure World* correspond à un mode d'exécution isolé.

Le passage d'un mode à l'autre se fait via une instruction spécifique (`SMC`). L'exécution de cette instruction provoque une transition vers le *Secure World* et l'exécution du *monitor*, un composant logiciel qui gère les changements de contexte entre les deux modes d'exécution. Suivant les implémentations logicielles, une ou plusieurs applications de sécurité peuvent être exécutées dans le *Secure World*. Un OS sécurisé (*TEE OS*) peut également être utilisé pour faciliter l'accès et le partage des ressources matérielles entre ces différentes applications de sécurité. L'ensemble des applications et l'OS qui s'exécutent dans le *non Secure World* sont parfois désignés sous le terme de REE (Rich Execution Environment).

Le *Secure World* permet d'accéder à l'ensemble des ressources de la plateforme (y compris la mémoire des logiciels exécutés dans le *non Secure World*) mais l'inverse n'est pas vrai. Le mode courant du processeur peut-être exporté vers les périphériques via le bus, ce qui permet de restreindre l'accès à certains périphériques en fonction du mode d'exécution. Cela permet notamment de restreindre l'accès à une plage d'adresses physiques de la DRAM (Dynamic Random Access Memory, mémoire dynamique à accès aléatoire) au *Secure World*. Toutefois, cela doit être implémenté par le fabricant du SoC (System on a Chip, système sur une puce).

Sur le marché des *smartphones* Android, on retrouve principalement trois implémentations de TEE TrustZone :

— QSEE/QTEE pour les téléphone utilisant un SoC Qualcomm ;
— TrustedCore, développé par Huawei ;
— Kinibi, développé par Trustonic (et utilisé notamment sur certains téléphones Samsung).

Toutefois, d'autres implémentations existent [58], notamment Linaro OP-TEE [59], qui est distribuée en *open-source*. Samsung utilisait principalement Kinibi mais a maintenant développé sa propre implémentation : TEEGRIS [60].

Les principales applications exécutées dans les TEE fournissent des services aux applications Android : gestion des DRM (Digital Rights Management, gestion des droits numériques), Android KeyMaster (protection des clefs cryptographiques), paiement sécurisé ou l'authentification FIDO (Fast IDentity Online).

Samsung a également développé Knox, une solution qui vérifie et renforce l'intégrité des applications et du système Android [61]. Cette solution implémente différents services dont la vérification de l'intégrité du noyau [62, 63]. Ces solutions sont issues de travaux

réalisés par des chercheurs de l'université d'État de Caroline du Nord (NC State) [64]. Des chercheurs ont également proposé des mécanismes permettant de sécuriser les échanges entre le REE et le TEE [65].

Une des problématiques des TEE TrustZone est que le déploiement d'applications sécurisées nécessite l'accord du fabricant du périphérique qui a intégré le TEE, ce qui limite en pratique le déploiement de ces applications. En effet, contrairement aux autres mécanismes d'enclaves comme SGX, TrustZone ne fournit qu'un seul environnement sécurisé. Toutes les applications sécurisées s'exécutent dans le même environnement. L'ajout d'une application sécurisée augmente donc la taille de la TCB puisque ces applications sont faiblement cloisonnées entre elles. Le *TEE OS* peut tout de même les isoler avec les mécanismes classiques de pagination. Des travaux récents ont également proposé d'utiliser le langage Rust pour limiter l'exploitation de vulnérabilités liées à la gestion de la mémoire dans le TEE [66].

Afin de faciliter le déploiement d'applications sécurisées, Sun *et al.* ont proposé TrustICE [67], un mécanisme qui permet d'isoler des enclaves dans le REE. La partie installée dans le *Secure World* est minimale et n'a pas besoin d'être modifiée lors du déploiement d'une enclave. L'approche est similaire à Flicker [17] mais utilise TrustZone au lieu de Intel TXT.

Toutefois, ces approches implémentent une isolation temporelle qui n'est pas compatible avec les processeurs multi-cœurs modernes. Des travaux récents ont étendu cette approche en renforçant l'isolation et en permettant l'exécution en parallèle de l'enclave avec un ou plusieurs TEE [68]. SecTEE [69] propose une approche similaire mais vise également à protéger les applications des attaques physiques et des attaques par canaux auxiliaires exploitant les accès à la mémoire. Ce projet est particulièrement intéressant car TrustZone ne fournit pas nativement de protection face aux attaques contre la microarchitecture. En outre, TrustZone ne fournit pas non plus de mécanisme de chiffrement de la mémoire principale. Il est donc recommandé que les TEE utilisent seulement la mémoire interne du SoC, afin d'éviter les attaques physiques d'écoute sur le bus mémoire. Toutefois, cette mémoire est très limitée en quantité, ce qui impose des contraintes sur le TEE. Zhao *et al.* ont proposé d'utiliser un TEE minimal qui réside dans la mémoire interne du SoC et permet de stocker des applications sécurisées dans la DRAM, en utilisant du chiffrement [70].

TrustZone est une technologie développée initialement pour le domaine des *smartphones*. Les processeurs ARM sont également de plus en plus utilisés au niveau des serveurs, dans le domaine du *cloud computing*. Néanmoins, TrustZone n'est pas nativement adapté à la virtualisation. Des chercheurs ont donc proposé une approche pour supporter plusieurs TEE virtualisés [71].

2.1.4.2 Intel SGX

Intel a développé la technologie SGX permettant d'exécuter des enclaves sécurisées [72]. L'objectif est de pouvoir isoler des fonctions sensibles sans faire confiance à l'OS ni à l'hyperviseur. Le cas d'usage typique est le *cloud computing* où des utilisateurs souhaitent déployer des applications sur des infrastructures qu'ils ne maîtrisent pas. On peut donc redouter dans ce cas que l'OS et l'hyperviseur soient malveillants ou compromis. Le code et les données exécutés dans une enclave SGX sont donc également isolés de ces composants.

Contrairement à TrustZone, l'objectif est de se prémunir également des attaques physiques par observation des communications entre le processeur et la mémoire. Pour cette raison, la mémoire des enclaves est chiffrée par le processeur avec une clé unique. En revanche, les attaques par canaux auxiliaires ne font pas partie du modèle de menace, ce qui est problématique, étant donné les cas d'usage envisagés. Le code des enclaves doit donc utiliser des contre-mesures pour lutter contre ces attaques. Par exemple Weiser *et al.* proposent d'utiliser une mémoire ORAM implémentée en FPGA pour protéger le code des enclaves contre ce type d'attaques [73].

Contrairement à TrustZone, le code des enclaves s'exécute dans l'espace utilisateur et ne peut réaliser d'appel système. Pour interagir avec le matériel, il doit donc appeler du code situé en dehors de l'enclave. De même, SGX s'appuie sur l'OS pour gérer les pages mémoires de l'enclave. Des mécanismes de protection s'assurent que l'OS ne peut compromettre l'intégrité des enclaves. Toutefois, les enclaves SGX peuvent être vulnérables aux *controlled-channel attacks*. L'OS peut provoquer intentionnellement des erreurs de page afin d'observer la séquence des adresses accédées par l'enclave. Afin de lutter contre ce canal auxiliaire, Shih *et al.* ont proposé T-SGX [74] qui s'appuie sur le mécanisme de mémoire transactionnelle Intel TSX (Transactional Synchronization Extensions).

Ce mécanisme a été utilisé par de nombreux travaux de recherche [75]. Des chercheurs du MIT ont notamment publié un document qui explique le fonctionnement de ce mécanisme complexe qui a des interactions avec les nombreux mécanismes déployés dans les architecture x86 [76].

Afin de faciliter le développement d'enclaves, Intel fournit un SDK (Software Development Kit, kit de développement logiciel) [77]. Il est également possible d'utiliser des SDK fournis par d'autres éditeurs comme Microsoft [78], Google [79] ou Fortanix [80]. Ce dernier permet de développer des enclaves en Rust, ce qui limite leur surface d'attaques. C'est également l'approche proposée par des chercheurs de Baidu [81].

Ces SDK sont destinés à isoler certaines fonctions d'applications qui doivent être développées spécifiquement pour cette technologie. D'autres outils [82-84] ont été développés pour permettre d'isoler des applications entières au sein d'une enclave, en minimisant l'effort d'adaptation. Comme il n'est pas possible de réaliser directement des appels système depuis les enclaves, ces outils utilisent une approche de *library OS*. Cette approche consiste à ajouter dans l'enclave une bibliothèque qui fournit les fonctions de l'OS au code de l'enclave et réalise des appels externes. Graphene [85] suit la même approche mais ne nécessite pas de recompiler les applications.

Ces approches permettent d'isoler des applications en minimisant l'effort d'adaptation. Toutefois, comme l'intégralité du code de l'application est exécutée dans l'enclave, cela augmente la surface d'attaque et la TCB de l'enclave. Des approches ont donc proposé de réduire cette surface d'attaque [86]. De même, certaines approches renforcent la sécurité en cloisonnant le code exécuté au sein de l'enclave à l'aide de Intel MPK [87], en mettant en place de l'ASLR au niveau de l'enclave [88] ou en utilisant le langage Rust [80, 81].

2.1.4.3 Enclaves RISC-V

RISC-V est une architecture de processeur ouverte issue des travaux de l'université de Berkeley et promue par la fondation RISC-V, qui publie notamment les spécifications. Plusieurs projets se sont intéressés à définir un mécanisme d'enclave pour cette architec-

ture. Dessouky *et al.* dressent un état de l'art de ces différents projets en comparant leur architecture, les fonctionnalités apportées, le modèle d'attaquant considéré, la TCB et les fonctionnalités matérielles utilisées [89].

Sanctum [90], développé par le MIT, est un des premiers projets d'enclaves RISC-V. Il prend en compte, dans son modèle de menaces, les attaques par canaux auxiliaires mais pas les attaques physiques. Le mécanisme est similaire à Intel SGX. Les enclaves s'exécutent en espace utilisateur et s'appuient sur l'OS. Toutefois, les auteurs ont inclus des mécanismes pour se prémunir contre les attaques par canaux auxiliaires et notamment les *controlled-channel attacks*.

Keystone [91], un projet de l'université de Berkeley, réalisé notamment par l'équipe qui a développé l'architecture RISC-V, considère un modèle de menaces similaire mais inclut également les attaques physiques par espionnage du bus mémoire. Toutefois, à l'inverse de Sanctum et de Intel SGX, les enclaves Keystone comportent également du code qui s'exécute en mode noyau.

TIMBER-V [92] propose un mécanisme d'enclave pour les systèmes embarqués en s'appuyant sur une approche d'étiquetage de la mémoire (*tagged memory*) qui permet d'associer une étiquette à des blocs mémoire. Cette approche a été utilisée dans d'autres travaux de la littérature pour implémenter du DIFT (Dynamic Information Flow Tracking, suivi dynamique des flux d'information) [93] ou des systèmes de capacités [94]. Les auteurs l'utilisent ici pour implémenter un mécanisme d'isolation à grain fin afin d'éviter la fragmentation de la mémoire et d'utiliser au mieux la quantité limitée de mémoire disponible dans les systèmes embarqués.

Ces projets offrent différents modèles d'enclaves qui sont plus ou moins adaptés selon les besoins. En outre, ils offrent une protection limitée contre les attaques par canaux auxiliaires et ne proposent pas de canaux de communications sécurisés vers les périphériques. Le projet CURE [95] tente de résoudre ces différents problèmes en proposant différents types de modèles d'enclaves.

Des entreprises ont également développé et commercialisent des mécanismes d'enclaves RISC-V, notamment HEX-Five [96].

2.1.5 Co-processeur de sécurité

Les enclaves présentées dans les sections précédentes s'exécutent au sein du processeur principal. Elles utilisent des mécanismes d'isolation fournis par le processeur mais partagent les ressources matérielles (cœur CPU, cache, RAM, périphériques) avec le reste des applications. Cela les rend, en général, vulnérables aux attaques par canaux auxiliaires. En outre, elles peuvent être vulnérables à un défaut dans le mécanisme d'isolation.

Afin de renforcer l'isolation, des approches ont proposé d'utiliser un processeur ou un cœur d'exécution, séparé physiquement du processeur principal, pour isoler des fonctions de sécurité. Historiquement, IBM a été un des premiers constructeurs à suivre cette approche en développant le processeur IBM 4758 [97]. L'objectif était de fournir un TEE pour exécuter des applications sécurisées. En outre, des chercheurs d'IBM ont proposé d'utiliser ce coprocesseur pour implémenter un service de surveillance des applications exécutées sur le processeur principal, afin de détecter d'éventuelles intrusions [98].

De nos jours, Apple déploie un coprocesseur de sécurité appelé SEP (Secure Enclave

Processor) dans ses téléphones iPhone depuis la version 5s [99]. Il s'agit d'un processeur séparé du processeur principal. Il utilise une zone dédiée de la DRAM, qui est chiffrée. Le processeur principal peut uniquement communiquer avec le SEP via un mécanisme de boîte aux lettres. Ce processeur permet entre autres de stocker des clés cryptographiques, de chiffrer les données biométriques et de gérer le démarrage sécurisé du *smartphone*.

AMD utilise également un coprocesseur appelé *Secure Processsor* ou PSP (Platform Security Processor) [100] qui fournit des services similaires. Il s'agit d'un processeur ARM qui gère notamment le démarrage sécurisé des plateformes AMD car le PSP est démarré avant le processeur x86. Il comprend également un TPM logiciel. Le *firmware* de ce processeur est stocké de manière sécurisée dans une mémoire flash dédiée. Toutefois, des chercheurs ont pu récemment exploiter une faille leur permettant de récupérer ce *firmware* en écoutant sur le bus SPI reliant le PSP à la mémoire *flash* [100]. Des employés d'une société israélienne ont également démontré plusieurs failles permettant d'injecter du code dans le *firmware* du PSP, en raison d'une vulnérabilité dans le code de vérification d'intégrité du *firmware* [101]. Comme ce processeur a accès, sans restriction, à l'intégralité de la mémoire physique, l'exécution de code malveillant au sein du PSP permet de contourner les autres mécanismes de sécurité de la plateforme. Ces vulnérabilités ont depuis été corrigées.

HP utilise également un microcontrôleur dédié pour implémenter sa solution *HP Sure Start* [102, 103]. Ce mécanisme permet notamment de vérifier l'intégrité du *firmware* UEFI des ordinateurs haut de gamme de la marque, et de les restaurer en cas de compromission.

Intel utilise aussi un processeur isolé, le CSME (Converged Security and Management Engine), qui est maintenant situé au sein du *chipset* intégré des processeurs Intel, le PCH (Platform Controller Hub) [104]. Il s'agit, dans les dernières versions des processeurs Intel, d'un processeur Intel 486 isolé, qui est relié au bus PCIe. Ce composant a un accès privilégié à la mémoire et aux différents périphériques de l'ordinateur. Son *firmware* comprend un OS dérivé de Minix [105] et plusieurs applications. Il est utilisé pour la gestion des DRM et pour implémenter la technologie Intel AMT (Active Management Technology). Cette dernière permet d'administrer l'ordinateur à distance, même lorsqu'il est éteint, dès lors qu'il est alimenté électriquement et relié à un réseau informatique. Le CSME exécute également un TPM logiciel.

Toutefois, de nombreuses vulnérabilités ont été découvertes concernant les logiciels exécutés sur le CSME et en particulier ceux gérant AMT [106]. Comme il s'agit d'un composant très privilégié, l'exploitation d'une vulnérabilité permet à un attaquant d'obtenir des privilèges importants sur la plateforme. En outre, les différents travaux de rétroconception ont démontré que les mécanismes de protection du logiciel (protection contre les corruptions mémoire, isolation interne, etc.) sont assez limités.

Le firmware du CSME est situé dans une mémoire *flash*. Il est signé et chiffré. Cependant, des employés de la société russe Positive Technology ont pu exploiter une vulnérabilité sur certaines cartes mères permettant d'activer la mise au point via JTAG du processeur x86 et de son *chipset* [107]. Cette attaque leur a permis d'avoir accès au code du *firmware* du CSME. En effet, sur ses processeurs récents, Intel a intégré la possibilité de transporter le protocole JTAG via USB (Universal Serial Bus) en utilisant le DCI (Direct Connect Interface) [108].

Ces différentes technologies sont propriétaires et peu d'informations publiques sont dis-

ponibles. Les détails d'implémentation ont souvent été révélés suite à l'exploitation de vulnérabilités ayant permis d'effectuer un travail de rétroconception. En outre, elles permettent essentiellement d'implémenter des services de sécurité du concepteur de la plateforme.

Des travaux de recherche se sont également intéressés à l'utilisation d'un coprocesseur isolé afin de surveiller les applications exécutées sur le processeur principal, notamment l'OS et l'hyperviseur. Au delà de la vérification d'intégrité, des approches ont également été proposées pour vérifier différents types de politique de sécurité, comme le CFI ou le DIFT.

2.1.5.1 Vérification de l'intégrité de l'OS et de l'hyperviseur

Plusieurs travaux ont proposé des approches de vérification de l'intégrité du noyau ou de l'hyperviseur en utilisant un coprocesseur isolé. Ainsi, Copilot [109] utilise un coprocesseur connecté au bus PCI (Peripheral Component Interconnect) afin de vérifier l'intégrité du code du noyau ainsi que certaines tables de pointeurs. L'objectif est de détecter des *rootkit* au niveau du noyau. L'approche de détection est simple : le détecteur s'assure que les structures n'ont pas été modifiées en vérifiant leurs résumés cryptographiques. Les auteurs de cette approche l'ont ensuite étendue afin de détecter un plus large spectre d'attaques [110]. En effet, l'approche précédente n'est pas adaptée pour surveiller des structures qui sont censées être modifiées régulièrement durant l'exécution du noyau, comme par exemple la liste des processus. Ils ont donc proposé une approche permettant d'exprimer des invariants qui doivent être respectés sur la manipulation de ces structures par le code du noyau. Des chercheurs d'Intel ont proposé une approche similaire pour détecter des *rootkit* dans un hyperviseur. Ils ont proposé pour cela d'utiliser l'ancêtre du CSME [111].

Wang *et al.* ont développé une approche hybride pour surveiller l'intégrité de l'OS et de l'hyperviseur [112]. Ils utilisent le SMM pour réaliser l'acquisition de la mémoire et déportent la vérification vers un serveur externe en utilisant une carte réseau spécifique. De même Hypersentry utilise un contrôleur BMC (Baseboard Management Controller) afin de pouvoir déclencher à distance la vérification de l'intégrité de la mémoire, qui est réalisée par un code exécuté en SMM [113]. Bien qu'elles utilisent un composant externe, ces approches reposent en grande partie sur l'isolation fournie par le SMM.

Les travaux présentés jusqu'ici reposent sur l'acquisition régulière de la mémoire. Toutefois, cette approche est vulnérable aux *transient attacks* qui consistent à réaliser une attaque sur le système surveillé entre deux acquisitions puis à nettoyer les traces de l'attaque. Pour lutter contre ce type d'attaques, les auteurs de Vigilare [114] proposent de surveiller l'ensemble des transactions réalisées sur le bus mémoire. Ils utilisent pour cela un composant externe connecté sur le bus en question. Ils ont par la suite étendu leur approche pour surveiller les structures dynamiques du noyau [115]. L'approche de détection est similaire à celle proposée par Petroni *et al.* [110].

Toutefois, les approches surveillant le bus mémoire peuvent être contournées par un attaquant utilisant une politique de type *write-back* sur les caches. Lee *et al.* ont proposé une approche complémentaire utilisant l'interface de mise au point du processeur (*Core Debugging Interface*) afin de prendre en compte cette limitation [116]. En outre, ces approches nécessitent des modifications importantes au niveau du matériel. Koromilas *et al.* ont proposé d'utiliser le GPU (Graphics Processing Unit) pour isoler le moniteur et réaliser

régulièrement des acquisitions de la mémoire à une fréquence importante, ce qui permet d'éviter les *transient attacks* sans requérir de modification au niveau matériel [117]. De même Nighthawk [118] implémente le moniteur dans le CSME, ce qui permet de surveiller l'ensemble du code s'exécutant sur le processeur principal, y compris en SMM et ce sans nécessiter de modification au niveau matériel.

Ces différentes approches se focalisent sur la vérification d'intégrité. Toutefois d'autres politiques de sécurité peuvent être implémentées via un coprocesseur externe. Ces approches sont une alternative à la vérification logicielle ou intégrée au processeur. L'utilisation d'un support matériel permet de protéger le mécanisme et d'accélérer les traitements. Le déport dans un processeur externe a l'avantage de ne pas impliquer de modification de la microarchitecture du processeur principal. Toutefois, en isolant la fonction de sécurité de la sorte, se pose le problème du fossé sémantique. Le processeur externe possède une visibilité réduite sur l'état du processeur principal.

Pour franchir ce fossé, il est nécessaire d'extraire les informations nécessaires du processeur principal et de les envoyer vers le coprocesseur de sécurité. Ces informations dépendent de la politique de sécurité implémentée. Des travaux ont proposé de réutiliser des mécanismes de traces existants sur les processeurs du marché pour envoyer ces informations au coprocesseur. Ces mécanismes, dont l'objectif premier est la mise au point des programmes, peuvent être détournés pour mettre en place des mécanismes de sécurité, comme présenté en section 2.1.7. Nous évoquons ici leur utilisation dans le cadre des échanges avec un coprocesseur de sécurité.

2.1.5.2 Vérification de politiques de sécurité

Des travaux se sont intéressés à implémenter une politique de CFI à l'aide d'un coprocesseur externe. Cette politique permet de lutter contre les attaques de corruption de la mémoire qui visent à détourner le flot d'exécution de l'application. Ainsi, l'entreprise Secure-IC a développé HCODE [119], un projet permettant de surveiller les instructions de branchement depuis un moniteur externe afin de vérifier la politique de CFI. Ils modifient pour cela un cœur de processeur SPARC LEON afin de générer les traces et de s'interfacer avec le moniteur externe.

Lee *et al.* ont également proposé d'utiliser le mécanisme de traces ARM PTM afin de vérifier le CFI sur un moniteur de sécurité externe [120]. Cette approche ne nécessite aucune modification au niveau du matériel.

Des chercheurs de l'équipe projet Inria/CentraleSupélec CIDRE ont proposé, dans le cadre d'une collaboration avec les laboratoires d'HP Inc, une approche permettant de vérifier une politique de CFI au niveau du code du firmware s'exécutant en SMM [121]. Comme indiqué en section 2.1.2, ce type de code s'exécute avec des privilèges élevés. Une vulnérabilité peut alors permettre à l'attaquant de se camoufler et de devenir persistant sur la machine. Ce type de code étant développé en C, il est potentiellement vulnérable aux attaques de corruption mémoire, d'où l'intérêt de le protéger en utilisant le CFI.

Le DIFT est un autre type de politique de sécurité permettant de détecter un large spectre d'attaques. Il s'agit d'associer des métadonnées, appelées *tag* ou *label* dans la littérature, à des conteneurs d'information (typiquement, des pages mémoires et des registres). Il s'agit ensuite de propager ces étiquettes en fonction des instructions exécutées afin de refléter les flux d'information qui résultent de l'exécution de chaque instruction.

Afin d'implémenter la propagation et la vérification des étiquettes dans un coprocesseur externe, plusieurs approches ont été proposées. Raksha [122] est un des premiers travaux à proposer de découpler le mécanisme de DIFT de l'implémentation du processeur principal. Pour combler le fossé sémantique lié à l'isolation, il est nécessaire d'envoyer la suite des instructions générant des flux d'information vers le coprocesseur. L'approche repose sur la mise en place d'un mécanisme de traces qui permet d'envoyer ces informations au moniteur DIFT. Rasksha nécessite quelques modifications de la microarchitecture du processeur principal pour mettre en place ce mécanisme de traces, ce qui limite son adoption.

Des travaux ont proposé d'utiliser les mécanismes de traces fournis par les processeurs disponibles sur le marché. Ainsi Lee *et al.* utilisent le mécanisme ARM ETM (Embedded Trace Macrocell) disponible sur la plupart des processeurs ARM destinés aux systèmes embarqués [123]. Ce mécanisme permet de tracer toutes les instructions exécutées.

Toutefois, sur les processeurs plus puissants, utilisés typiquement sur les *smartphone*, ce mécanisme n'est pas implémenté car le débit des traces serait trop important pour pouvoir être traité en temps réel sans impacter les performances. ARM implémente à la place le mécanisme PTM (Program Trace Macrocell) qui ne trace que les exceptions et les sauts. Il est donc nécessaire de retrouver les séquences d'instructions exécutées linéairement dans un bloc de base. HardBlare implémente le DIFT en utilisant le mécanisme de traces ARM PTM. Il complète les informations manquantes, nécessaires au DIFT, en utilisant une préanalyse statique et une instrumentation des applications lors de la compilation. Ces informations manquantes sont stockées sous forme d'annotations et envoyées au coprocesseur lors du chargement du programme exécutable [93].

Ces différentes approches se focalisent sur la vérification d'une politique de sécurité. PHMon [124] propose un mécanisme générique qui permet de surveiller différents types de politiques de sécurité et ainsi de rentabiliser les mécanismes matériels ajoutés à l'architecture.

2.1.5.3 Limites des moniteurs externes

Les approches qui reposent sur un moniteur externe et déclenchent des vérifications lors de certains événements sont potentiellement vulnérables à des attaques en évasion. L'attaquant peut inférer le type d'événements capturés ainsi que la fréquence des vérifications afin de conduire son attaque entre deux vérifications et effacer ses traces [125].

En outre, les moniteurs ne peuvent observer que la mémoire physique. Des attaques en évasion sont donc possibles en modifiant la correspondance entre mémoire virtuelle et physique, au dépend du moniteur [126]. Enfin, les données qui résident dans le cache peuvent échapper à la vérification, si le moniteur se contente de surveiller la mémoire [127]. Ces limitations peuvent également concerner les moniteurs isolés à l'aide de TrustZone [128].

2.1.6 Mécanismes de sécurité au niveau du jeu d'instructions

Outre l'ajout de mécanismes d'isolation permettant d'implémenter des TEE ou des enclaves, différents projets ont proposé d'implémenter des mécanismes de sécurité au niveau de l'ISA (*Instruction Set Architecture*, jeu d'instructions) des processeurs.

2.1.6.1 Protection du noyau du système d'exploitation

Le noyau des OS est une cible de choix pour les attaquants puisqu'il s'exécute dans un contexte privilégié. Entre outre, les OS tels que Microsoft Windows, Linux ou Apple Mac OSX sont développés dans des langages peu robustes aux attaques liées à la gestion de la mémoire, tel que le C. Ils comportent en outre de nombreuses fonctionnalités et représentent une quantité de code importante. De fait, ils font l'objet régulièrement de vulnérabilités qui sont exploitées par des attaquants pour élever leurs privilèges et mettre en place des *rootkit*.

Les fabricants de microprocesseurs ont donc proposé des extensions de leur ISA afin de renforcer la protection du noyau face à ces attaques. Ainsi, Intel a intégré l'extension SMEP (Supervisor Mode Execution Prevention) à ses microprocesseurs qui empêche le noyau d'exécuter du code situé dans une page de l'espace utilisateur. Cette extension est également disponible sur les processeurs AMD. Elle permet d'éviter les attaques qui détournent le flot d'exécution du noyau vers du code situé en espace utilisateur, sous la maîtrise de l'attaquant (attaques appelées *return-to-user* ou *ret2usr*). ARM propose un mécanisme similaire : PXN (Privilege Execute Never).

Ces mécanismes peuvent être déployés facilement car, en général, le noyau ne doit pas légitimement exécuter du code utilisateur. SMAP (Supervisor Mode Access Prevention) est un mécanisme complémentaire, disponible sur les processeurs Intel et AMD, qui interdit de manière générale au noyau d'accéder aux pages mémoire de l'espace utilisateur. Toutefois, ce mécanisme ne peut être activé en permanence car le noyau doit parfois accéder légitimement à l'espace des processus utilisateurs, notamment dans le cadre de certains appels système (par exemple, pour un appel système `read`, le noyau doit recopier les données lues dans le fichier vers un tampon situé dans la mémoire du processus). Le code du noyau doit donc être modifié pour activer et désactiver ce mécanisme suivant les besoins. ARM dispose d'un mécanisme similaire : PAN (Privilege Access Never).

Toutefois, des chercheurs ont montré que ces mécanismes ne sont pas suffisants car le noyau partage implicitement des espaces mémoire avec les processus [129].

Des chercheurs ont également utilisé SMAP pour mettre en œuvre de l'isolation interne aux processus [130], avec de meilleures performances que les mécanismes dédiés à cet usage comme Intel MPK (Memory Protection Keys), qui sont présentés par la suite.

2.1.6.2 Vérification de l'intégrité du flot de contrôle

Une part importante des applications et du code privilégié (OS, hyperviseur, *firmware* UEFI, etc.) est toujours développée à l'aide de langage offrant peu de protection contre les attaques de corruption de la mémoire. Différents mécanismes ont été peu à peu adoptés pour lutter contre ces attaques. Ces mécanismes sont implémentés au niveau du matériel (pages non exécutables pour éviter l'injection de code), de l'OS (ASLR (Address Space Layout Randomization, distribution aléatoire de l'espace d'adressage) pour complexifier l'utilisation du ROP) et du compilateur (utilisation de canaris pour détecter les dépassements de pile).

De nos jours, ces contre-mesures permettent de se prémunir efficacement contre les attaques de corruption de la mémoire utilisant de l'injection de code. Toutefois, l'utilisation du ROP reste possible, notamment lorsque l'ASLR est mise en défaut (suite à des fuites

mémoire ou à des attaques par canaux auxiliaires). Dans ce contexte, l'utilisation du CFI paraît prometteuse. Afin de limiter l'impact à l'exécution de cette protection, des travaux se sont intéressés à étendre l'ISA afin d'implémenter matériellement la vérification d'une politique CFI.

Ainsi, Davi *et al.* ont proposé d'ajouter deux nouvelles instructions permettant d'étiqueter les branchements et les retours de fonction afin d'implémenter une politique CFI [131]. Plus précisément, ils cherchent à se prémunir contre les attaques détournant le flot de contrôle au niveau de l'adresse de retour des fonctions (*backward edge* CFI). Ils ont ensuite étendu ces travaux en proposant et en évaluant l'approche sur des processeurs *softcore* Intel Siskiyou Peak et LEON3 [132]. Des travaux se sont intéressés plus globalement à protéger à la fois l'adresse de retour et les appels indirects de fonction (qui utilise des pointeurs de fonction) [133].

Ces approches nécessitent de modifier la microarchitecture et ne s'appliquent pas aux processeurs existants. Toutefois, les fabricants de processeurs ont intégré récemment un support du CFI au sein de leurs produits. Ainsi, Intel a développé CET (Control-flow Enforcement Technology) [134, 135], une extension permettant de mettre en place du CFI à gros grain. L'approche permet de protéger les retours de fonction à l'aide d'une *shadow stack* et les appels indirects de fonction à l'aide d'une instruction spécifique. Plus précisément, tout branchement doit sauter au début d'une fonction valide, ce qui limite le ROP. Une instruction spécifique sert à identifier le début valide d'une fonction (la cible d'un branchement doit toujours correspondre à cette instruction). Toutefois, l'attaquant peut toujours contourner le flot de contrôle en sautant à une adresse marquant le début d'une fonction légitime.

ARM a également proposé une approche similaire avec sa solution BTI (Branch Target Indicators) [136]. Toutefois, cette technologie n'implémente pas de *shadow stack* et ne protège pas les adresses de retour.

2.1.6.3 Isolation interne aux processus

Les processeurs modernes proposent de nombreux mécanismes d'isolation (niveau de privilège d'exécution, extension de virtualisation, mécanismes d'enclaves, etc.). Cependant, ces mécanismes ne permettent pas aisément de cloisonner le code appartenant à un même processus, sans nécessiter des changements de contexte coûteux en termes d'impacts sur les performances. Un tel besoin existe car certaines applications implémentent de nombreuses fonctionnalités et comprennent de nombreuses lignes de code. Une vulnérabilité dans le code de l'application peut donc permettre à un attaquant d'accéder à l'ensemble des données de l'application.

Des chercheurs ont proposé d'utiliser le niveau de privilèges intermédiaire des processeurs x86 (*ring* 2), qui n'est en général pas utilisé dans les OS modernes, afin d'implémenter un mécanisme permettant de séparer les applications en deux espaces isolés [137]. Cette approche a l'avantage d'être compatible avec tous les processeurs de l'architectures x86 et ne nécessite pas de mécanisme additionnel. Toutefois, il ne fournit que deux environnements isolés pour chaque processus.

Intel a également développé MPK, un mécanisme permettant de restreindre l'accès d'un *thread* à un groupe de pages. Plusieurs travaux de recherche ont proposé d'utiliser ce mécanisme pour isoler le code à l'intérieur d'un même processus. Park *et al.* ont notamment

proposé `libmpk`, une bibliothèque qui abstrait le mécanisme matériel, renforce la sécurité et comble certaines lacunes du mécanisme, notamment le nombre limité de clés [138]. Les auteurs ont appliqué leur approche à différents programmes dont OpenSSL et différents compilateurs JavaScript.

ERIM [139] combine l'utilisation de MPK avec une analyse statique du code binaire des applications afin de s'assurer qu'il ne comprend pas de séquence d'instructions, exécutables par une attaque ROP, permettant de contourner l'isolation.

D'autres cas d'usages ont été proposés. Ainsi Wang *et al.* proposent d'utiliser MPX (Memory Protection Extensions) pour isoler un moniteur de référence au sein des applications [140]. A titre d'exemple, ils ont implémenté une approche de *Multi-Va-riant eXecution (MVX)*, qui consiste à comparer les résultats de l'exécution de différentes implémentations d'une même fonctionnalité (diversité fonctionnelle).

Ces approches sont intéressantes mais ne peuvent être implémentées que sur les processeur x86 disposant de MPK (les processeurs AMD récents supportent maintenant également cette fonctionnalité). En outre, Connor *et al.* ont évalué le niveau de sécurité apporté par ces mécanismes d'isolation [141]. Ils ont démontré que cette isolation pouvait être contournée par des attaques de type *confused deputy* utilisant l'OS.

Mogosanu *et al.* ont également présenté un mécanisme d'isolation à grain fin interne au processus, complémentaire de la pagination [142]. Ils ont proposé une implémentation de leur approche sur un *softcore* RISC-V. Leur approche vise notamment à minimiser le surcoût lié à l'utilisation du cloisonnement. Schrammel *et al.* ont proposé un mécanisme d'isolation intra-processus pour les processeurs RISC-V qui est à la fois robuste en ce qui concerne la sécurité (qui ne nécessite pas, par exemple, d'analyser le code binaire de l'application) et performant [143]. Les auteurs ont implémenté leur approche sur un *softcore* RISC-V et fournissent un mode d'émulation pour x86 utilisant MPK.

2.1.6.4 Vérification de la validité des pointeurs

Les langages bas niveau comme le C offrent peu de garanties sur la gestion de la mémoire. En particulier, l'utilisation de pointeurs peut conduire à des vulnérabilités si le développeur n'inclut pas explicitement des vérifications dans le code de son application. L'absence de vérification peut typiquement conduire à des vulnérabilités de type *buffer overflow*. Le pointeur est alors utilisé pour accéder à la mémoire en dehors des zones réservées par le développeur.

Pour lutter contre les attaques exploitant ces vulnérabilités, des travaux ont proposé des extensions matérielles permettant de vérifier la validité des pointeurs avant leur utilisation. L'objectif est de réduire le surcoût à l'exécution lié à la vérification, par rapport à des solutions purement logicielles. Il s'agit typiquement d'associer aux pointeurs des métadonnées permettant de déterminer leur domaine de validité (par exemple, les bornes d'un tableau).

Kwon *et al.* ont par exemple proposé d'encoder ces métadonnés de manière adjacente aux bits d'adresses pour former un *fat pointer* [144]. Ils ont pour cela conçu un processeur avec une architecture et une microarchitecture spécifique. Leur approche permet en outre d'associer un type (valeur entière, pointeur valide, pointeur invalide, etc.) à chaque mot manipulé par le processeur. Cette approche, développée dans le cadre du projet CRASH/SAFE, leur a permis de mettre au point une architecture générique [145] permettant

de mettre en œuvre différents types de politiques de sécurité [146]. Cela permet, entre autres, de protéger les données sur la pile [147].

Cette approche est intéressante car elle fournit un support matériel générique, permettant d'implémenter différents mécanismes et donc de contrer un vaste spectre d'attaques. Toutefois, il s'agit d'une architecture très différente des architectures disponibles sur le marché. Tous les composants logiciels (OS, compilateur, applications, etc.) doivent être redéveloppés à cet effet, ce qui limite l'adoption d'une telle approche.

Les constructeurs de microprocesseurs ont également proposé des extensions matérielles permettant de vérifier la validité des pointeurs. Ainsi, Intel a proposé MPX, une extension permettant de spécifier et vérifier les bornes de validité des pointeurs. Cette extension est notamment utilisée dans le projet LMP [148] pour protéger la *shadow stack* pour la vérification du CFI. Les auteurs notent cependant que certaines fonctionnalités de MPX ont un surcoût important à l'exécution. Ce résultat a été confirmé par Oleksenko *et al.* qui ont réalisé une analyse expérimentale détaillée de MPX et de ses performances [149]. Les auteurs sont arrivés à la conclusion surprenante que cette solution matérielle n'offre pas de meilleurs performances que les solutions purement logicielles. Depuis, le support de MPX dans le compilateur GCC et le noyau Linux a été arrêté et Intel n'intègre plus cette extension dans ses nouveaux processeurs.

ARM propose également les extensions PA (Pointer Authentication) et MTE (Memory Tagging Extension) dans son architecture ARMv8-A. Ces deux fonctionnalités utilisent les bits de poids fort des adresses virtuelles qui ne sont pas utilisés pour encoder l'adresse. En effet, les adresses virtuelles sont, par exemple, encodées au maximum sur 39 bits pour l'architecture ARM 64 bits sous Linux. Ces deux fonctionnalités partagent l'espace restant, quand elles sont utilisées de concert.

MTE, introduite dans l'architecture ARMv8.5-A, permet d'associer une étiquette (*tag*) aux données stockées en mémoire ainsi qu'aux pointeurs permettant de manipuler ces données. Si le programme manipule des données à l'aide d'un pointeur possédant une étiquette différente de celle des données, une exception est levée. La solution stocke les étiquettes de la mémoire en dédiant 4 bits pour chaque mot de 16 octets en mémoire. Les étiquettes des pointeurs sont stockées dans les bits de poids fort de l'adresse. Des instructions ont été ajoutées pour contrôler les étiquettes.

Ce type de mécanisme permet par exemple de s'assurer que les pointeurs ne peuvent être utilisés pour accéder à une zone non allouée ou, de manière générale, à une zone possédant une étiquette différente. Cela réduit la faisabilité des attaques exploitant des corruptions mémoire. Toutefois, le nombre d'étiquettes étant réduit, l'attaquant peut toujours utiliser un pointeur pour manipuler des données en dehors des bornes mais vers une zone disposant de la même étiquette. MTE est maintenant supportée par le noyau Linux et le compilateur LLVM mais aucun processeur commercialisé à la date de rédaction de ce rapport ne dispose de cette fonctionnalité.

PA, introduite dans l'architecture ARMv8.3-A, permet d'authentifier les pointeurs en leur associant un MAC (Message Authentication Code) appelé PAC (Pointer Authentication Code), situé lui aussi dans les bits de poids fort des adresses. Cette fonctionnalité est notamment implémentée sur le SoC Apple A12 utilisé dans les iPhone XS. Des instructions ont été ajoutées afin de créer et d'authentifier des PAC. Un pointeur créé de la sorte doit être authentifié avant de pouvoir être utilisé, sinon la traduction d'adresse génère

une faute de page. Cette protection permet d'éviter qu'un attaquant puisse modifier ou forger un pointeur à partir d'un débordement de tampon, par exemple. Cette extension a notamment été utilisée par Liljestrand *et al.* [150]. Les auteurs proposent une solution permettant de résister aux attaques visant à forger ou réutiliser des PAC.

Le projet CHERI [94], développé principalement par des chercheurs de l'université de Cambridge, étend les principes d'étiquetage de la mémoire et de vérification des bornes des pointeurs présentés précédemment. Il utilise pour cela un mécanisme plus générique de capacités, qui permet également d'associer des permissions aux pointeurs. Cette approche permet de contrer un large spectre d'attaques exploitant des corruptions mémoire [151, 152]. Les chercheurs ont implémenté dans un premier temps cette approche sur un *softcore* spécifique. Ils ont également établi une collaboration avec ARM dans le cadre du projet Morello [153]. ARM a proposé une modification de son architecture supportant les extensions de CHERI et va développer un processeur expérimental implémentant cette architecture. Toutefois, il s'agit seulement pour l'instant d'un projet de recherche et ARM ne s'engage pas à inclure ce support dans ses futurs produits.

2.1.6.5 Autres approches

Des travaux se sont intéressés à implémenter d'autres types de mécanismes de sécurité au niveau de l'ISA.

Ainsi, Zagieboylo *et al.* ont proposé une ISA permettant de contrôler les flux d'informations, y compris ceux résultants de canaux auxiliaires temporels [154]. Pour cela, l'architecture associe des labels aux données manipulées par le processeur. Ce dernier doit propager et vérifier les étiquettes lors de l'exécution des instructions. L'objectif est de définir une ISA que doivent respecter des processeurs implémentant du DIFT. Ce type de processeur peut ainsi être utilisé par des logiciels souhaitant contrôler leur flux d'informations. L'approche n'a toutefois pas donné lieu à une implémentation.

Le projet HDFI [155] fournit un support matériel pour implémenter du DFI (Data Flow Integrity). L'approche, proposée initialement par des chercheurs de Microsoft Research, consiste à vérifier que les accès mémoire respectent le graphe de flot de données de l'application. Cette approche nécessite de déterminer un tel graphe et d'annoter le programme en associant, à chaque instruction de lecture, l'ensemble des instructions du programme qui ont pu légitimement écrire cette donnée. La vérification consiste à associer une étiquette à chaque donnée lors de son écriture puis à vérifier lors, de chaque lecture, que l'étiquette provient bien d'une écriture légitime. HDFI permet d'accélérer matériellement cette vérification. Les auteurs proposent d'étendre l'ISA RISC-V avec des instructions permettant d'étiqueter les données et de vérifier les étiquettes. Ils ont implémenté leur approche sur un *softcore* RISC-V. Toutefois, l'approche ne permet de stoker des étiquettes que sur un bit, ce qui permet tout de même d'implémenter différents types de protection mais diverge significativement de l'approche originale.

Des approches ont également proposé d'implémenter diverses formes de diversifications au niveau du jeu d'instructions. Il s'agit de diversifier la forme du code ou des données à l'exécution afin que l'attaquant ne puisse prédire cette forme lors d'une attaque de type ROP ou *data-only attacks*, qui vise à modifier ou lire les valeurs des variables de façon illégitime.

HARD [156] implémente une approche similaire au DFI. Il s'agit de chiffrer les variables

avec des clés choisies aléatoirement mais en s'assurant que des variables appartenant à une même classe d'équivalence soient chiffrées avec la même clé. Cette approche permet de protéger le code contre les *data-only attacks*.

Wang *et al.* ont proposé une architecture composée de deux ordinateurs, utilisant différentes ISA, reliés par un bus InfiniBand [157]. Cette architecture possède un jeu d'instructions hétérogènes et permet d'exécuter des applications qui peuvent utiliser l'une ou l'autre des ISA. L'application peut ainsi s'exécuter sur les deux ordinateurs et comparer les résultats (diversification) ou alterner aléatoirement entre les deux ISA.

De Clercq *et al.* proposent une approche d'ISR (Instruction Set Randomization) dont l'objectif est de contrer les attaques par injection de code et de type ROP [158]. Pour cela, le compilateur chiffre les instructions avec des clés dépendant du flot de contrôle. A l'exécution, le processeur déchiffre les instructions à la volée. Cette approche permet de mettre en œuvre un forme de CFI. Les auteurs visent le domaine des systèmes embarqués et ont implémenté leur approche sur un *softcore* LEON3.

Yu *et al.* ont proposé une architecture facilitant le développement d'applications *constant-time*, dont le temps d'exécution ne varie pas en fonction de la valeur des secrets [159]. Cette approche permet de lutter contre les attaques par canaux auxiliaires temporels. Les auteurs évaluent également leur approche en fournissant une microarchitecture implémentant cette ISA.

2.1.7 Mécanismes de traces et compteurs de performance

Les approches présentées dans la section précédente nécessitent de modifier l'ISA et la microarchitecture. D'autres approches ont à l'inverse proposé d'utiliser les fonctionnalités existantes des processeurs afin d'implémenter des protections contre les attaques logicielles. Des travaux se sont notamment intéressés à l'utilisation des mécanismes de traces et aux compteurs de performances. Ces mécanismes ont initialement été introduits pour des besoins de mise au point et de profilage des applications. Toutefois, leur usage peut être détourné pour implémenter des mécanismes de sécurité.

Sur l'architecture x86, des travaux utilisent ainsi le mécanisme de trace Intel PT [160] qui permet de tracer les branchements et les changements de contexte du processeur. Le mécanisme peut être configuré par le noyau de l'OS. Le processeur écrit les traces, qui sont fortement compressées, directement dans la mémoire physique. Ce mécanisme a notamment été utilisé pour implémenter une politique de CFI [161, 162] puisqu'il fournit des informations sur les branchements. En outre, Schumilo *et al.* ont proposé de l'utiliser pour réaliser du *fuzzing* au niveau du noyau de l'OS [163].

Les processeurs ARM disposent également de la technologie CoreSight [164, 165]. Suivant le type de processeur, le mécanisme de trace diffère. Pour les microcontrôleurs destinés aux systèmes embarqués, ARM intègre généralement un ETM qui permet de tracer toutes les instructions exécutées. En revanche, les processeurs applicatifs de type Cortex-A disposent d'un PTM qui, à l'instar d'Intel PT, ne permet de tracer que les branchements et les exceptions. Suivant le SoC et son implémentation, les traces peuvent être envoyées dans un *buffer* interne au SoC, ce qui permet d'y accéder depuis un programme exécuté sur le SoC, ou vers une interface de mise au point (interface JTAG). Certains SoC permettent également d'envoyer les traces vers un autre composant, par exemple un FPGA intégré comme dans le *SoC* ZYNQ de Xilinx.

Ninja utilise ainsi l'ETM pour analyser des logiciels malveillants depuis un environnement isolé à l'aide de TrustZone [166]. Kargos utilise le PTM pour détecter les injections de code dans le noyau de l'OS. Cette approche apporte les mêmes garanties que Intel SMEP ou PXN mais elle n'induit pas de pénalité sur les performances [167]. HART utilise l'ETM pour protéger les modules du noyau de l'OS sans nécessiter de les recompiler [168]. Les auteurs utilisent ce mécanisme pour implémenter AddressSanitizer [169], une protection logicielle qui permet de vérifier les accès mémoire pour prévenir les attaques de dépassement de tampon et de *use-after-free*. Contrairement à l'approche logicielle, l'approche utilisant HART ne nécessite pas d'avoir accès au code source du module et peut donc être appliquée sur les modules propriétaires fournis par un tiers.

Ces mécanismes présentent également une menace potentielle pour la sécurité du code exécuté dans les environnements cloisonnés comme TrustZone. Le mécanisme de trace des processeurs ARM permet en effet de tracer le code exécuté en mode protégé, depuis l'espace non protégé. Cette fonctionnalité peut être verrouillée mais Ning *et al.* ont montré que ce verrouillage n'était souvent pas mis en place dans les produits du marché [170].

Des approches ont également exploré l'utilisation des compteurs de performances pour implémenter des mécanismes de sécurité, notamment de la détection d'intrusions ou de la détection de codes malveillants [171]. Das *et al.* dressent un état de l'art de ces approches [172]. Suite à une analyse détaillée de ces mécanismes et des garanties qu'ils peuvent fournir, les auteurs démontrent qu'ils doivent être utilisés avec prudence dans le contexte de la sécurité informatique. En effet, ces mécanismes n'ont pas été conçus pour cet usage et les auteurs démontrent qu'il est possible de réaliser des attaques en évasion sur les mécanismes de détection qui les utilisent.

2.1.8 Spécification et preuve formelles des architectures matérielles

Les sections précédentes présentent les différents mécanismes de sécurité qui ont été implémentés à l'aide d'un support matériel. De fait, les microprocesseurs disponibles sur le marché embarquent de plus en plus de mécanismes de sécurité. Toutefois, se pose la question de la confiance que l'on peut avoir dans ces mécanismes. En effet, certains d'entre eux ont déjà fait l'objet de vulnérabilités exploitables par des attaquants.

Afin de renforcer la confiance dans ces mécanismes de sécurité, des travaux se sont intéressés à la spécification formelle de ces mécanismes. En outre, ce travail de spécification permet par la suite de vérifier formellement que ces mécanismes garantissent certaines propriétés de sécurité, définies elles aussi formellement. L'étape ultime consiste à s'assurer que l'implémentation du mécanisme est bien conforme à la spécification et permet donc d'assurer effectivement les propriétés de sécurité attendues.

Les mécanismes de sécurité matériel présentés dans les sections précédentes reposent en grande partie sur l'état de l'architecture matérielle. En outre, certains apportent des modifications à l'ISA. Il est donc nécessaire, dans un premier temps, de spécifier formellement la sémantique de l'ISA.

2.1.8.1 Formalisation de la sémantique du jeu d'instruction

Les spécifications publiques des ISA des principales architectures disponibles sur le marché (x86, ARM) sont publiées sous forme d'une description textuelle informelle. Cette description est volumineuse, parfois ambiguë. La description précise de la sémantique de certains mécanismes peut être dispersée dans plusieurs ouvrages. Cet état de fait complexifie la mise au point et l'adoption de méthodes formelles pour la vérification des mécanismes de sécurité matériels.

Toutefois, des travaux récents ont proposé de formaliser la sémantique de l'ISA des processeurs. Ainsi, ARM a établi une spécification formelle exécutable de ses processeurs dans le but de réaliser des vérifications formelles par *model-checking* et du test [173]. ARM a appliqué cette approche à plusieurs de ces microprocesseurs [174]. Cette spécification formelle est disponible sur le site du constructeur [175].

Des travaux se sont également intéressés à la formalisation de l'ISA des processeurs x86 [176] mais seule la partie utilisateur est modélisée, ce qui représente tout de même un travail conséquent, étant donné la complexité de ce jeu d'instructions. Toutefois, cette limitation ne permet pas de vérifier les mécanismes de sécurité qui s'appuient sur les instructions privilégiées.

Le projet SAIL de l'université de Cambridge a développé un *framework* et un langage de spécification permettant de spécifier formellement différentes architectures et d'utiliser cette spécification formelle dans différents outils de test et de validation formelle [177]. Il permet également de générer automatiquement une description textuelle de l'architecture. Les auteurs ont pu appliquer ce *framework* pour modéliser formellement l'ISA des processeurs ARM (en collaboration avec le constructeur), RISC-V et CHERI. Le langage a été adopté officiellement par la fondation RISC-V pour la spécification formelle de l'architecture RISC-V.

Un des défis consiste à s'assurer que l'implémentation matérielle, qu'il s'agisse d'un circuit ASIC ou d'un *softcore* implémenté sur FPGA, respecte cette spécification formelle. Chlipala *et al.* [178] ont proposé le langage de description de matériel Kami, qui permet de concevoir des composants matériels (comme les langages VHDL ou Verilog), mais aussi de prouver (grâce au langage Coq) l'équivalence entre de tels composants. Chlipala *et al.* ont proposé, comme application de leur langage, une implémentation d'une partie d'un processeur RISC-V. Plus récemment, les auteurs de Kami ont proposé Kôika [179], un compilateur vérifié d'un HDL proche de Bluespec vers des circuits RTL.

L'entreprise SiFive[10], qui propose notamment des implémentations FPGA de RISC-V, a repris la formalisation de Kami et une implémentation plus complète d'un processeur RISC-V, en Kami, est en cours de développement [180]. Toutefois, le projet ne semble plus actif.

2.1.8.2 Modélisation et preuve formelle de mécanismes de sécurité matériels

Letan *et al.* ont travaillé sur la formalisation de mécanismes de sécurité matériels, en étudiant notamment le SMM des processeurs x86 [181]. Une spécification formelle de ce mode d'exécution, ainsi qu'un modèle minimal de l'architecture x86, ont été écrits en Coq.

10. https://www.sifive.com/

Une preuve a ensuite été établie que cette spécification respecte une politique de sécurité formellement spécifiée, sous réserve que le BIOS utilise correctement ce mécanisme. Ce dernier point fait l'objet d'hypothèses qui expriment explicitement les contraintes d'utilisation du mécanisme que les développeurs du BIOS doivent respecter. Toutefois, le modèle du processeur utilisé n'est pas complet et la preuve porte uniquement sur la spécification.

Ferraiuolo *et al.* ont proposé un HDL (*Hardware Description Language*, langage de description du matériel) fortement typé permettant de spécifier et de prouver des mécanismes d'isolation en utilisant des techniques de vérification des flux d'information, à l'aide d'une analyse statique de vérification du typage [182]. Ils ont appliqué leur approche en implémentant un mécanisme similaire à TrustZone sur un *softcore* MIPS. Des chercheurs ont proposé une approche similaire en s'appuyant sur Chisel, un HDL de haut niveau utilisé notamment pour implémenter des processeurs RISC-V [183]. Ils vérifient que la politique de flux d'information est respectée à l'aide d'un solveur SMT (Satisfiability Modulo Theories).

Récemment, les chercheurs impliqués dans les projet CHERI et SAIL ont utilisé le langage SAIL pour spécifier l'architecture CHERI et prouver qu'elle garantit certaines propriétés de sécurité, notamment l'évolution monotone des capacités (un attaquant ne peut faire évoluer une capacité pour la rendre plus permissive) [184]. SAIL peut également être utilisé pour tester une implémentation afin de s'assurer qu'elle est conforme à la spécification. Le projet ne permet cependant pas, à l'heure actuelle, de vérifier formellement qu'une implémentation respecte la sémantique de la spécification formelle.

2.2 Attaques logicielles contre la microarchitecture

Le matériel est souvent réputé plus robuste aux attaques logicielles, car il ne peut être modifié. Toutefois, les architectures des processeurs sont de plus en plus complexes et implémentent de nombreux mécanismes qui peuvent interférer entre eux. Des erreurs dans la spécification de ces mécanismes peuvent conduire à des vulnérabilités permettant de contourner les mécanismes de sécurité.

L'implémentation de ces architectures est également de plus en plus complexe, pour des raisons d'optimisation de la performance et de la consommation énergétique, qui sont les deux défis principaux auxquels doivent faire face les constructeurs de processeurs. En effet, en raison du *power wall*, il n'est plus possible d'augmenter facilement les performances des processeurs en augmentant la fréquence de leur horloge, comme cela a été le cas pendant des décennies, sans impliquer une consommation et une dissipation énergétique rédhibitoires. Pour ces raisons, les constructeurs ont concentré leurs efforts sur l'optimisation de la microarchitecture des processeurs. Ces différentes optimisations consistent, d'une manière ou d'une autre, à exécuter un maximum d'instructions en parallèle.

Ainsi, les processeurs modernes comportent généralement plusieurs cœurs indépendants qui permettent d'exécuter plusieurs fils d'instruction en parallèle. Au sein d'un cœur, les opérations de récupération (*fetch*), de décodage (*decode*), d'exécution (*execute*), d'accès mémoire (*memory access*) et de modification de l'état des registres (*writeback*) de ces instructions sont réalisées en parallèle au sein d'un *pipeline*. Ainsi, par exemple, la récupération de la prochaine instruction peut être initiée avant la fin de l'exécution de l'instruction qui la précède.

En outre, les architectures modernes superscalaires comportent plusieurs unités d'exécutions (notamment plusieurs ALU (Arithmetic–Logic Unit, unité arithmétique et logique)) et peuvent donc récupérer et exécuter plusieurs instructions en parallèle, pour un même fil d'exécution ou pour plusieurs fils d'exécution (cœur virtuel ou *hyperthreading*).

Sur les processeurs x86, les instructions sont traduites par le microprocesseur sous la forme d'une ou plusieurs micro-instructions, qui sont ensuite exécutées. Cette traduction est réalisée par un *microcode* qui peut être chargé dynamiquement, ce qui permet une forme limitée de mise-à-jour de la microarchitecture. Toutefois, le format de ce microcode n'est pas public.

Afin d'optimiser ce traitement en parallèle, le processeur peut être amené à exécuter des instructions dans le désordre (*out-of-order execution*) et parfois de manière spéculative. En effet, l'exécution de certaines instructions, notamment celles qui réalisent des accès mémoire, peuvent nécessiter un nombre de cycles plus importants que les autres. En outre, certaines instructions peuvent nécessiter des unités d'exécution qui ne sont pas disponibles, à la différence des instructions qui les suivent dans le programme.

Toutefois, les processeurs exposent les résultats de l'exécution de ces instructions au niveau de l'architecture (*Retirement Unit*), en respectant l'ordre des instructions. Si une exception s'est produite, le processeur peut décider de ne pas exposer le résultat de certaines instructions au niveau de l'architecture. L'état des registres exposé par l'architecture est alors restauré à celui précédent l'instruction fautive.

En outre, le processeur peut inférer les prochaines instructions à exécuter. Cela est notamment nécessaire lors des branchements conditionnels ou des sauts indirects. L'adresse de la prochaine instruction dépend alors du résultat des instructions précédentes, qui peut ne pas être disponible. Le processeur vérifie *a posteriori* que la prédiction était correcte et, en cas d'erreur, annule l'exécution des instructions exécutées spéculativement, en s'assurant que l'état de l'architecture est identique à celui précédant l'exécution de ces instructions.

L'exécution spéculative et l'exécution dans le désordre peuvent donc conduire à annuler l'exécution de certaines instructions. La littérature emploie le terme d'exécution transitoire (*transient execution*) pour désigner l'exécution de ces instructions avant qu'elle ne soit validée et exposée au niveau de l'architecture.

En outre, la microarchitecture repose généralement sur un ou plusieurs niveaux de mémoire cache. Les caches ont un temps d'accès beaucoup plus faible que la mémoire principale (DRAM). Toutefois, leur taille est limitée en raison de leur coût. Ils permettent d'accélérer les accès mémoire tout en donnant l'illusion au logiciel de disposer d'une quantité importante de mémoire (correspondant à la taille de la DRAM) avec un faible temps d'accès. L'accès au cache est transparent pour le développeur de l'application.

A la différence de l'architecture, les détails de la microarchitecture sont en général peu documentés et évoluent entre les différentes versions des processeurs. Les différentes optimisations de la microarchitecture ont donné lieu à des vulnérabilités exploitables par le logiciel [185]. Ces attaques contre la microarchitecture s'expliquent notamment par le fait que plusieurs fils d'exécution partagent des ressources (cache, unité d'exécution, etc.) et que l'état de la microarchitecture peut être en partie visible par différents programmes s'exécutant en parallèle ou en temps partagé (l'état de la microarchitecture n'étant pas réinitialisé entre chaque changement de contexte). Ces attaques peuvent donc contourner

les mécanismes d'isolation garantis au niveau de l'architecture.

Nous détaillons par la suite les travaux relatifs aux attaques logicielles par canaux auxiliaires (*side channel attacks*), dans la section 2.2.1. Ces attaques consistent à exploiter des fuites d'information résultant de l'observation d'une grandeur physique (temps d'exécution, consommation électrique, dissipation thermique, etc.). Contrairement aux attaques physiques, ces observations sont réalisées directement par du logiciel exécuté sur le processeur. Puis nous évoquons en section 2.2.2 les attaques qui exploitent l'exécution transitoire et qui s'appuient sur les attaques par canaux auxiliaires pour faire fuir des données confidentielles. Nous présentons en section 2.2.3 les attaques par injection de fautes, qui permettent de modifier l'état du processeur en contournant les mécanismes d'isolation. Nous nous attardons en section 2.2.4 sur les attaques qui visent plus particulièrement à contourner les mécanismes d'enclave. Enfin, la réalisation d'attaques et l'évaluation de la sécurité des processeurs nécessitent d'obtenir des informations précises sur la microarchitecture. Nous présentons donc en section 2.2.5 les travaux liés à la rétroconception de la microarchitecture.

2.2.1 Attaques logicielles par canaux auxiliaires

Les attaques logicielles par canaux auxiliaires visent à mesurer, par l'exécution d'un programme, des effets physiques (temps d'exécution, consommation électrique ou température) qui permettent d'inférer des informations confidentielles appartenant à un autre programme. Ces attaques permettent donc de contourner les mécanismes d'isolation au niveau de l'architecture qui ne prennent pas en compte ce type de menace. Pour ce type d'attaque, l'attaquant maîtrise seulement le code permettant d'effectuer des mesures et cible un programme qui va influencer la grandeur mesurée à son insu. Si cette fluctuation dépend d'une valeur confidentielle, une fuite d'information est possible.

Ces attaques sont à distinguer des attaques par canal caché. Dans ce dernier cas, l'attaquant contrôle les deux extrémités du canal de communication. Il s'agit donc d'un modèle d'attaquant plus puissant et qui est souvent moins réaliste. Toutefois, la mise aux point d'une attaque par canal caché peut être une première étape dans l'analyse des attaques par canaux auxiliaires.

Des chercheurs ont dressé un état de l'art et une classification de ces attaques [186, 187]. La plupart des travaux de la littérature s'intéressent aux canaux temporels pour lesquels l'attaquant peut mesurer une différence de temps d'exécution. Ces différences peuvent résulter de plusieurs facteurs :

— selon les chemins parcourus dans le graphe de contrôle du programme, le temps d'exécution peut varier en fonction du nombre et du type d'instructions exécutées ;
— le temps d'exécution d'une instruction d'accès à la mémoire varie en fonction de l'état des caches (l'accès est plus rapide si la donnée ou l'instruction est présente dans le cache) ;
— dans les architectures superscalaires, le temps d'exécution d'une instruction dépend de l'état de disponibilité des unités d'exécution nécessaires à l'exécution de l'instruction.

La plupart des travaux sur l'exploitation des canaux auxiliaires temporels se sont intéressés à ceux résultant de l'utilisation des caches. Toutefois, des travaux ont également démontré que la variation de temps d'exécution liée à la disponibilité des unités d'exécution (*port*

contention) pouvait également être exploitée [188] lorsque plusieurs files d'exécution sont exécutées simultanément sur un même cœur physique (*hyper-threading*).

Ces attaques nécessitent d'identifier une ressource partagée et de réaliser un travail de rétroconception, bien souvent spécifique à chaque microarchitecture, afin de pouvoir éventuellement l'exploiter. Gras *et al.* ont proposé ABSynthe [189], une approche en boîte noire permettant d'identifier automatiquement des canaux cachés de type *port contention* à partir d'un programme cible et d'une microarchitecture, sans nécessiter d'effort de rétroconception explicite de la microarchitecture.

2.2.1.1 Exploitation des temps d'accès aux caches

Plusieurs types d'attaques contre les caches ont été mis en évidence dans la littérature. Ainsi, Osvik *et al.* sont parmi les premiers à démontrer la faisabilité de ce type d'attaques [190]. Les auteurs cherchent à retrouver la clé secrète AES utilisée par un processus ou par le noyau Linux, à partir d'un second processus. Ils démontrent ainsi que ce type d'attaque permet de contourner l'isolation fournie par le mécanisme de pagination de la mémoire. Ils utilisent pour cela une attaque de type EVICT+TIME. Cette attaque consiste à évincer des données d'une ligne de cache puis de mesurer le temps d'exécution du programme victime afin de déterminer s'il a réalisé un accès mémoire à une adresse correspondant à cette ligne de cache.

Les auteurs utilisent également une attaque de type PRIME+PROBE qui consiste à précharger toutes les lignes d'un même ensemble puis à lire les adresses correspondant à ces lignes. Un temps d'accès plus long signifie que le programme victime a réalisé un accès mémoire correspondant à cet ensemble. Ces attaques visent le cache de premier niveau (L1) et doivent donc être exécutées sur le même cœur physique que la victime.

Zhang *et al.* ont également démontré que ce type d'attaque était possible entre différentes machines virtuelles s'exécutant sur différents cœurs d'un processeur [191]. Toutefois, l'attaque nécessite un filtrage complexe du bruit généré notamment par l'exécution de l'hyperviseur et le débit du canal auxiliaire est relativement lent (l'attaque nécessite six heures d'observation pour retrouver une clé cryptographique).

Plus récemment, Yarom *et al.* ont démontré que le partage de pages mémoire entre processus, par exemple les pages de code des bibliothèques partagées, permet de réaliser une attaque de type FLUSH + RELOAD [192]. Ce type d'attaque présente un bon rapport signal sur bruit et permet des débits plus importants. L'attaque consiste à vider une ligne de cache avant d'exécuter le processus victime puis de réaliser une lecture à une adresse correspondant à cette ligne. Un faible temps d'accès permet alors d'inférer que la victime a réalisé une lecture à une adresse correspondant à cette ligne.

Contrairement aux attaques précédentes, les auteurs exploitent le LLC (Last Level Cache), qui est partagé entre les différents cœurs d'un microprocesseur. Le processus de l'attaquant ne doit donc plus nécessairement être exécuté sur le même cœur que le processus victime. Les auteurs démontrent qu'il est possible de retrouver la clé privée utilisée par un processus exécutant le logiciel de chiffrement GnuPG. Apecechea *et al.* ont confirmé que cette attaque pouvait être utilisée entre différentes machines virtuelles d'un même serveur [193]. Enfin, Zhang *et al.* ont démontré que cette attaque est également possible sur les processeurs ARM [194].

Les attaques FLUSH+RELOAD sont très performantes et permettent de contourner

l'isolation, notamment celle implémentée par les instructions de virtualisation. Dans le contexte du *cloud computing*, le partage de ressources doit faire l'objet d'une attention particulière et les fournisseurs configurent l'hyperviseur pour éviter le partage de page mémoire entre différentes machines virtuelles.

Toutefois, Liu *et al.* ont démontré que la propriété d'inclusion des caches de l'architecture x86 pouvait être exploitée pour réaliser une variante de l'attaque PRIME+PROBE ciblant le LLC, sans nécessiter de partage d'une page mémoire [195]. De même Apecechea *et al.* sont arrivés à un résultat similaire en utilisant des pages de grandes tailles [196]. Maurice *et al.* ont également démontré qu'il était possible d'établir un canal caché à l'aide d'une attaque de type PRIME+PROBE entre deux machines virtuelles isolées [197]. Les auteurs ont réussi à déployer des techniques permettant d'établir un canal de communication suffisamment robuste pour supporter une communication SSH entre les deux VM.

Ces attaques génèrent des anomalies dans le nombre de *cache hit* et *cache miss* que des approches ont proposé de détecter, notamment à l'aide des compteurs de performances. Gruss *et al.* ont proposé une nouvelle attaque de type FLUSH+FLUSH [198] qui exploite le fait que l'instruction permettant de vider une ligne de cache s'exécute plus rapidement si la ligne correspondant à l'adresse indiquée n'est pas contenue dans le cache. Cette attaque ne nécessite pas de réaliser d'accès supplémentaire pour tester si les données sont présentes dans le cache. Toutefois, des techniques de détection ont été proposées permettant de détecter cette attaque. Briongos *et al.* ont proposé récemment un nouveau type d'attaque, RELOAD+REFRESH [199], dans l'objectif de contourner les mécanismes de détection proposés jusqu'à présent. L'attaque, qui vise le LLC, ne nécessite pas d'évincer une ligne de cache.

Moritz *et al.* ont démontré que des attaques de type EVICT+RELOAD pouvaient également être réalisées sur les processeurs ARM, couramment utilisés dans les *smartphone* [200], sans nécessiter de permissions spécifiques.

Disselkoen *et al.* ont démontré que le mécanisme de mémoire transactionnelle implémenté par Intel TSX, permettait de réaliser une attaque de type PRIME+ABORT [201], qui est une variante de l'attaque PRIME+PROBE. Toutefois, contrairement aux autres attaques, il ne s'agit pas d'un canal temporel qui nécessite que l'attaquant puisse réaliser une mesure précise du temps d'exécution. Elle est donc particulièrement intéressante pour contourner les mécanismes qui visent à restreindre les possibilités de mesure du temps d'exécution. Elle repose sur les propriétés de la mémoire transactionnelle qui utilise le cache pour stocker temporairement les résultats des accès mémoire avant de les valider. L'attaquant débute une transaction et précharge toutes les lignes d'un même ensemble, comme dans l'attaque PRIME+PROBE. Toutefois, dans le cadre de l'attaque PRIME+ABORT, si la victime réalise un accès mémoire à une adresse correspondant au même ensemble, la transaction échoue et l'attaquant est notifié sans avoir besoin d'effectuer de mesure temporelle. L'attaque ne nécessite pas que l'attaquant s'exécute sur le même cœur que la victime ni qu'il ne partage une page mémoire.

Les attaques précédentes permettent de contourner les mécanismes d'isolation des processeurs pour réaliser des atteintes à la confidentialité des données. Toutefois, elles nécessitent que le code ciblé réalise des accès mémoire dont l'adresse dépend de la valeur confidentielle. La réalisation d'une attaque demande donc une étude approfondie du code de l'application ciblée. Gruss *et al.* ont proposé une approche de type *Cache Template Attacks* permettant d'automatiser les attaques par canal auxiliaire utilisant le LLC [202].

Leur approche ne nécessite pas d'analyser le code de l'application. Elle repose sur une première phase permettant d'analyser les accès mémoire de l'application pour différents événements ciblés, en mesurant le *cache hit ratio* sur différentes adresses mémoires. Les événements peuvent être de diverses natures telles que la frappe d'une touche du clavier, la réception d'un email ou le chiffrement à l'aide d'un élément spécifique d'une clé. La phase d'attaque consiste à exploiter les profils ainsi obtenus. Les deux phases reposent sur des attaques FLUSH+RELOAD.

Les attaques précédentes supposent que l'attaquant et la victime s'exécutent sur le même processeur, même s'ils peuvent potentiellement s'exécuter sur des cœurs différents. Irazoqui *et al.* ont démontré la faisabilité d'attaques par canaux auxiliaire entre différents processeurs d'un ordinateur utilisant une configuration à plusieurs processeurs reliés par un bus d'interconnexion comme AMD HyperTransport ou Intel Quickpath [203].

Allan *et al.* ont montré que les attaques par dégradation des performances permettaient d'amplifier les attaques par canaux auxiliaires [204]. Pour ce type d'attaques, l'attaquant cherche délibérément à utiliser les ressources partagées avec la victime, ce qui a pour effet de ralentir l'exécution du code de la victime. Ce ralentissement facilite la réalisation d'une attaque par canal auxiliaire.

Les attaques sur le LLC sont les moins contraignantes pour l'attaquant car elles ne nécessitent pas que le processus attaquant s'exécute sur le même cœur que sa cible. Elles sont donc particulièrement redoutées dans un contexte de *cloud computing*. Toutefois, comme évoqué précédemment, les attaques nécessitant le partage d'une page mémoire sont maintenant prises en compte par les fournisseurs de *cloud* qui évitent le partage de pages mémoire entre machines virtuelles. En outre, les autres types d'attaques reposent sur la propriété d'inclusion des caches. Toutefois, ces attaques sont remises en cause par les nouvelles architectures de processeurs qui ne présentent plus cette propriété d'inclusion. Yan *et al.* ont cependant proposé une variante de l'attaque PRIME+PROBE qui fonctionne également pour ce type de configuration [205].

Les attaques précédentes visent les caches de données et d'instructions. Des contre-mesures logicielles, s'appuyant sur des fonctionnalités matérielles comme TSX ou CAT (Cache Allocation Technology), ont été proposées pour lutter contre ces attaques. Toutefois, Gras *et al.* ont démontré que d'autres caches partagés pouvaient être exploités [206], notamment le TLB (Translation Lookaside Buffer) utilisé dans la traduction des adresses virtuelles.

En outre, des contre-mesures purement logicielles ont été proposées afin de séparer les ensembles du LLC entre différents domaines de sécurité. Pour cela, ces approches exploitent la manière dont les adresses physiques sont associées aux ensembles du LLC. L'objectif est de restreindre les adresses utilisables par le code sous le contrôle potentiel d'un attaquant afin qu'il ne puisse accéder aux ensembles du cache correspondant au code sensible, manipulant des données confidentielles. Toutefois, van Schaik *et al.* ont montré que ces approches pouvaient être contournées en exploitant un composant matériel de confiance, comme la MMU (Memory Managment Unit) [207]. En effet, ce composant peut également utiliser le cache pour stocker des structures de données qu'il utilise, comme les tables de pages. Comme ces accès ne sont pas réalisés directement par l'exécution d'instructions d'accès à la mémoire, ils ne sont pas protégés.

La même équipe de recherche a également démontré qu'il était possible de casser le mécanisme d'ASLR en réalisant une attaque de type EVICT+TIME contre le processus

de traduction d'adresses réalisé par la MMU, en observant les données mises en cache par cette dernière [208]. Les chercheurs ont démontré que cette attaque fonctionnait sur les processeurs ARM, Intel et AMD. Des travaux ont également démontré que des attaques par canaux auxiliaires temporels pouvaient compromettre l'ASLR au niveau du code noyau de l'OS [209, 210].

Les attaques présentées dans cette section nécessitent que l'attaquant puisse exécuter du code générant des accès mémoire à des adresses spécifiques et puisse mesurer précisément des temps d'exécution. Ces contraintes laissent à penser que ce type d'attaque ne peut être réalisé que par un attaquant disposant déjà d'un accès à la machine. Toutefois, des travaux ont montré que ces attaques pouvait également être réalisées à partir de programmes JavaScript exécutés dans un navigateur. L'attaque peut donc être réalisée à distance par un attaquant qui réussit à convaincre un utilisateur de naviguer sur une page Web contenant le script malveillant. Il est alors possible de réaliser des attaques contournant les restrictions de la *sandbox* JavaScript et d'espionner la mémoire d'autres processus [211]. Shusterman *et al.* ont également démontré que de telles attaques pouvaient permettre de réaliser des attaques de type *website fingerprinting* permettant à l'attaquant d'inférer les sites Web visités par la victime dans d'autres onglets de son navigateur [212].

2.2.1.2 Exploitation de la température et de la consommation électrique

Des travaux se sont intéressés à d'autres types de canaux auxiliaires exploitables par des attaques logicielles, notamment ceux liés à la température et la consommation électrique. Toutefois, les attaques purement logicielles exploitant ce type de canal auxiliaire ont pour l'instant fait l'objet de moins de travaux de recherche.

Masti *et al.* ont ainsi démontré qu'il était possible d'exploiter l'information sur la température, fournie par les capteurs intégrés aux différents cœurs d'un microprocesseur pour la gestion dynamique de la consommation énergétique et de la température du processeur [213]. Cette température est potentiellement influencée par la charge des coeurs adjacents, ce qui permet de créer un canal caché de 12.5 bps.

Lipp *et al.* ont récemment exploité l'interface RAPL (Running Average Power Limit) des processeurs Intel qui expose de manière non privilégiée des données corrélées à la consommation électrique des cœurs du microprocesseur [214]. A l'aide de ce canal auxiliaire, les auteurs ont pu faire fuir une clé AES-NI d'un code exécuté au sein d'une enclave SGX et casser l'ASLR utilisée au sein du noyau.

2.2.2 Exploitation de l'exécution transitoire

Les attaques logicielles par canaux auxiliaires, notamment celles utilisant les caches, exploitent des vulnérabilités matérielles qui résultent du partage de ressources entre des domaines qui devraient être normalement isolés (par exemple, différents processus ou différentes machines virtuelles). Toutefois, elle s'appuient également sur le fait que le code de la victime réalise des accès mémoire qui dépendent d'une information confidentielle. Ces contre-mesures logicielles existent pour éviter ce type de motifs dans les algorithmes cryptographiques, mais sont difficilement généralisables à tout type de logiciel.

Plus récemment, un nouveau type d'attaques contre la microarchitecture est apparu : les attaques exploitant l'exécution transitoire. Ces attaques utilisent des canaux auxiliaires pour faire fuir des données mais elles exploitent à la base des vulnérabilités différentes,

liées à l'exécution spéculative et *out-of-order*. Ces attaques ont été révélées avec les failles Meltdown [215] et Spectre [216]. D'autres types de vulnérabilités ont depuis été mises à jour.

Des travaux ont récemment dressé un état de l'art de ces différentes attaques et des vulnérabilités qu'elles exploitent [217, 218]. Google maintient également un répertoire contenant des exemples illustrant ces différentes attaques [219].

Plusieurs chercheurs impliqués dans la découverte des vulnérabilités Spectre et Meltdown ainsi que leurs variantes, proposent d'adopter une classification que nous reprenons dans ce rapport [217]. Cette classification est également mise à jour sur le site web mis en place par les auteurs [220]. Ainsi, les auteurs distinguent tout d'abord les attaques de type Spectre et celles de type Meltdown.

Les attaques de type Meltdown exploitent l'exécution *out-of-order* des instructions et plus précisément le fait que le traitement des exceptions est retardé. Lorsque l'exécution d'une instruction provoque une exception, par exemple un défaut de page, plusieurs instructions consécutives à l'instruction fautive sont exécutées de manière transitoire avant que l'exception ne soit traitée. Lors du traitement de l'exception, le processeur annule les effets architecturaux de ces instructions. Ainsi, les registres sont restaurés à leur état précédant l'exécution de ces instructions. Toutefois, pour des raison de performance, l'état de la microarchitecture n'est pas modifié. Cela inclut notamment l'état des caches. Comme les instructions ne permettent pas directement d'accéder à l'état de la microarchitecture, le mécanisme était réputé sûr.

Les travaux présentés dans la section précédente ont cependant démontré qu'il était possible d'utiliser des attaques par canal auxiliaire pour faire fuir l'état de la microarchitecture, notamment en mesurant des temps d'accès. Les attaques de type Meltdown consistent donc à provoquer sciemment une exception qui va permettre à l'attaquant de forcer le processeur à exécuter son code de manière transitoire, dans un domaine de protection auquel il n'a normalement pas accès (par exemple, le mode noyau). Ces instructions auront alors accès à l'espace mémoire du domaine de protection et vont faire fuir des données situées dans cet espace mémoire, en utilisant un canal auxiliaire. Le motif typique consiste à réaliser un accès mémoire à une adresse correspondant à la valeur que l'on souhaite faire fuir. Les attaques de types Meltown sont les plus critiques car elles permettent à l'attaquant de faire exécuter son propre code de manière transitoire, ce qui lui donne un contrôle important.

Les attaques de type Spectre exploitent l'exécution spéculative liée aux branchements conditionnels, aux sauts indirects ou aux retours de fonction. L'attaquant cherche à influencer les différents mécanismes de prédiction du processeur pour forcer l'exécution spéculative, dans le processus victime, d'une séquence d'instructions (*gadget*). Cette séquence lui permet de faire fuir des informations confidentielles du processus victime, à l'aide d'un canal auxiliaire. Il s'agit donc d'une forme de *code reuse attack*. Toutefois, les *gadget* sont exécutés de manière spéculative et ne peuvent faire fuir de l'information qu'à l'aide d'un canal auxiliaire, car l'état de l'architecture est restauré par le processeur lorsqu'il détecte l'erreur de spéculation. Ces attaques sont plus complexes à mettre en œuvre car elles reposent sur la présence de *gadget* dans le code de la victime et elles nécessitent d'influencer les mécanismes de prédiction utilisés pour le code de la victime. Elles sont aussi plus difficiles à contrer que l'attaque Meltdown.

Nous détaillons par la suite les différentes variantes de ces attaques qui ont été publiées à ce jour.

2.2.2.1 Attaques de type Spectre

Canella *et al.* . classifient les différentes variantes de Spectre selon le mécanisme de prédiction empoisonné par l'attaquant. Ils montrent ensuite que, pour la plupart de ces variantes, l'empoisonnement peut être réalisé dans le même espace d'adressage que la victime ou dans un espace distinct. En outre, l'empoisonnement peut être réalisé par le propre code du programme victime ou par le code d'un processus exécuté par l'attaquant.

Dans l'article originel décrivant la vulnérabilité Spectre [216], deux variantes sont décrites, nommées Spectre-PHT et Spectre-BTB par Canella *et al.* . La première variante consiste à empoisonner le *Pattern History Table* qui est utilisé par le processeur pour inférer si un branchement conditionnel sera pris. L'exploitation consiste à identifier, dans le code de la victime, des motifs de code qui réalisent des vérifications de borne ou de typage avant de réaliser un accès en lecture à un tableau ou une structure. L'empoisonnement permettra d'exécuter du code de manière spéculative en dehors des bornes du tableau et d'inspecter la mémoire du processus victime. La seconde variante consiste à empoisonner le *Branch Target Buffer* qui est utilisé pour déterminer l'adresse d'un branchement indirect.

Maisuradze *et al.* ont mis à jour une troisième variante, Spectre-RSB, où l'attaquant empoisonne le *Return Stack Buffer* utilisé pour prédire la valeur de l'adresse de retour [221].

Kiriansky [222] présente une variante de Spectre-PHT où le code spéculé effectue une écriture en mémoire. Cette écriture en mémoire, réalisée de manière spéculative, peut elle-même réaliser un *buffer-overflow* et modifier des données dans la mémoire. Ces modifications seront annulées lorsque le processeur détectera l'erreur de spéculation mais elles peuvent permettre de contourner des protections contre Spectre-PHT ou de poursuivre l'exécution spéculative en contrôlant son flot d'exécution.

Les travaux précédents utilisent les caches comme canal auxiliaire. Bhattacharyya *et al.* ont démontré qu'il est également possible d'utiliser les canaux de type *port contention* [223]. De même Schwarz *et al.* utilisent un canal auxiliaire reposant sur les différences de temps d'exécution des instructions Intel AVX2 pour réaliser une exploitation à distance d'une faille de type Spectre [224].

Des travaux se sont également intéressés à l'utilisation de ces attaques de type Spectre pour réaliser des attaques plus générales. Par exemple, Wampler *et al.* utilisent ces attaques pour camoufler des logiciels malveillants [225].

Une des difficultés liée à l'exploitation des failles de type spectre est qu'elle nécessite des *gadgets* assez longs et complexes. Toutefois, Bhattacharyya *et al.* ont démontré qu'il était possible de chaîner plusieurs *gadgets*, comme dans une attaque classique de type ROP [226]. Toutefois, ces gadgets sont exécutés de manière spéculative et la longueur de la chaîne est limitée par le nombre de cycles nécessaires pour que le processeur détecte l'erreur de spéculation.

Göktaş *et al.* ont proposé de combiner une attaque de type Spectre avec l'exploitation d'une corruption mémoire à l'aide d'une attaque de type *code-reuse attack* permettant de

contourner les protections contre Spectre [227]. L'attaque permet *in fine* de casser l'ASLR sans provoquer le crash de l'application.

2.2.2.2 Attaques de type Meltdown

Canella *et al.* . classifient les différentes variantes de l'attaque Meltdown selon le type d'exception utilisé.

L'article original présentant la vulnérabilité Meltdown [215] détaille la variante Meltdown-US-L1 qui exploite le mécanisme de défaut de pages et plus précisément l'attribut *user/supervisor* des pages. L'attaque consiste à tenter d'accéder à l'espace mémoire du noyau qui est présent dans l'espace virtuel du processus mais qui n'est normalement pas accessible au code de l'application (les pages possèdent l'attribut *supervisor*). Une contre-mesure a consisté à supprimer les pages du noyau de l'espace d'adressage de l'application lorsque le processeur n'est plus en mode noyau. La vulnérabilité permettait de lire arbitrairement dans l'espace mémoire du noyau et potentiellement dans l'ensemble de la mémoire physique lorsque l'OS réalisait un *mapping* complet de cette mémoire dans son espace virtuel. Canella *et al.* ont également identifié une variante de cette attaque qui exploite le cache comme canal caché mais permet de contourner la protection mise en place par Intel MPK.

LazyFP est une attaque originellement associée aux attaques de type Spectre [228]. Canella *et al.* . l'ont classifiée comme une variante de type Meltdown (Meltdown-NM-REG). Elle repose sur l'utilisation des registres de l'unité de calcul flottant. Pour éviter de sauvegarder ces registres à chaque changement de contexte, il est possible de les marquer comme indisponibles. Si un autre processus tente de les utiliser, une exception *device-not-available* est levée. Dans ce contexte, l'exécution transitoire des instructions permet de faire fuir l'état de ces registres liés à des données du processus victime.

Différentes variantes de Meldown [229-231] ont été identifiées qui font fuir des données situées dans des *buffer internes* de la microarchitecture Intel. Ces variantes, appelées MDS (Microarchitectural Data Sampling) par Intel exploitent différents types d'exceptions selon le *buffer* utilisé. Ces *buffers* sont internes à chaque cœur physique et peuvent contenir des données provenant d'un autre domaine de protection (par exemple d'une enclave SGX). Ragab *et al.* ont montré qu'il est également possible d'exploiter le *staging buffer*, qui est partagé entre plusieurs cœurs d'un processeur [232].

Moghimi *et al.* ont proposé une approche de type *fuzzing* permettant d'identifier différentes variantes d'attaques de type Meltdown [233]. Leur outil leur a permis d'identifier de nouvelles sous-variantes de cette famille d'attaques.

2.2.2.3 Attaques de type *Load Value Injection*

LVI est un troisième type d'attaque exploitant l'exécution transitoire [234]. Tout comme les attaques MDS, elle expoite les *buffer* internes des processeurs Intel. Toutefois, l'attaque fonctionne dans le sens opposé. L'attaquant va empoisonner ces *buffer* avec des valeurs puis provoquer une exception dans le code de la victime. Cette exception aura pour effet l'exécution transitoire d'instructions dans le code de la victime, qui vont lire les valeurs injectées dans le *buffer* et exécuter des *gadgets* similaires à ceux des attaques Spectre. Ces *gadgets* vont faire fuir de l'information de la mémoire de la victime via des canaux auxiliaires.

2.2.3 Attaques logicielles par injection de fautes

Des attaques logicielles par injection de fautes matérielles ont également été mises en évidence dans la littérature. Il s'agit de provoquer une altération d'une grandeur physique qui va provoquer une erreur dans le matériel (par exemple le changement d'une valeur dans la mémoire) qui *in fine* conduira à une défaillance des mécanismes de protection du processeur. Contrairement aux injections de fautes par attaques physiques, la faute est provoquée uniquement par l'exécution d'un programme, ce qui permet de les réaliser potentiellement à distance.

Nous détaillons ici les travaux relatifs à deux types d'injections de fautes par attaques logicielles, qui ont été mis en évidence récemment : les attaques exploitant la gestion de l'énergie (section 2.2.3.1) et les attaques par martellement de la mémoire (*rowhammer*), présentées en section 2.2.3.2.

2.2.3.1 Exploitation logicielle de la gestion de l'énergie

Afin d'optimiser la gestion de la consommation d'électricité, les processeurs disposent de fonctionnalités de gestion dynamique de l'énergie. l'objectif est d'adapter au mieux la consommation énergétique en fonction de la demande, c'est-à-dire de la charge du processeur. Cette fonctionnalité est notamment cruciale pour les systèmes embarqués, les ordinateurs portables ou les *smartphone* alimentés sur batterie. Elle est également liée à la gestion de la dissipation énergétique et de la température du processeur.

Afin de moduler cette consommation énergétique, le processeur peut moduler certains paramètres : désactivation et mise en veille de certaines fonctionnalités, réduction de la fréquence ou de la tension d'alimentation. Afin d'optimiser cette modulation, les fabriquants exposent ces fonctionnalités au niveau de l'interface avec le logiciel. Ainsi, les programmes (généralement les noyaux des OS) peuvent adapter le fonctionnement du processeur en fonction de leurs besoins. Certains travaux se sont intéressés à la menace que pouvaient représenter ces interfaces de configuration.

Tang *et al.* ont ainsi étudié les mécanismes de gestion d'énergie des processeurs ARM utilisés dans les *smartphones* [235]. Ils ont démontré qu'un module malveillant du noyau de l'OS pouvait manipuler la gestion dynamique de la fréquence afin de provoquer des fautes. En fait, la fonctionnalité matérielle expose des paramètres de configuration et ne vérifie pas que ces paramètres appartiennent à la zone de fonctionnement nominale du processeur. Ainsi, les auteurs ont réussi à extraire des clés stockées dans la partie isolée par TrustZone. Ils ont également pu charger un programme auto-signé dans le *secure world*. Cette approche a été reprise dans VoltJockey [236] qui se focalise sur la manipulation de la valeur de la tension au lieu de faire varier la fréquence. L'attaque est ainsi plus furtive et difficile à prévenir. Les auteurs ont aussi démontré que cette attaque pouvait être réalisée sur les processeurs Intel contre SGX [237]

Murdock *et al.* ont démontré qu'une attaque similaire pouvait être réalisée sur les plateformes Intel pour contourner l'isolation fournie par SGX et faire fuir des données confidentielles de l'enclave [238]. Contrairement aux attaques par martellement de la mémoire, présentées dans la section suivante, cette injection de faute cible le processeur et non la mémoire. Ainsi le chiffrement de la mémoire réalisé par SGX ne permet pas de lutter contre cette attaque.

Kenjar *et al.* ont réalisé des travaux similaires. Ils s'attaquent toutefois plus spécifiquement aux instructions de calcul vectoriel et ont également mis en évidence des atteintes à l'intégrité permettant de contrôler le flot d'exécution [239] au sein d'une enclave SGX. Face à ces attaques, Intel a notamment publié une mise à jour du *firmware* UEFI permettant de désactiver certaines fonctionnalités de gestion de l'énergie.

2.2.3.2 Attaques par martellement de la mémoire

Contrairement aux attaques présentées dans les sections précédentes, Rowhammer est une attaque par injection de fautes contre la DRAM. Cette attaque exploite une vulnérabilité liée à la conception des circuits intégrés de DRAM. Ce type de mémoire repose essentiellement sur des cellules permettant de stocker un bit d'information à l'aide d'un condensateur et d'un transistor. L'état du condensateur (chargé ou déchargé) permet de représenter l'état de la cellule. Toutefois, le condensateur se décharge en raison de courants de fuite. Il est donc nécessaire de rafraîchir régulièrement l'état des cellules afin de recharger les condensateurs. Ces cellules sont organisées sous la forme d'une matrice. Les différents bits d'une même adresse mémoire sont en fait répartis sur plusieurs lignes. La lecture de ces lignes est destructive et nécessite de recharger l'état des cellules de la ligne.

L'attaque résulte de l'optimisation de cette technologie. Afin de proposer des quantités de mémoire plus importantes, les constructeurs ont augmenté la densité de cellules dans leurs circuits intégrés. Cela conduit à des cellules de plus en plus petites et de plus faible capacité. Ces circuits sont donc de plus en plus sujets aux perturbations électromagnétiques qui peuvent modifier l'état d'une cellule. L'attaque consiste à autogénérer ces perturbations en effectuant des lectures répétées de la mémoire. Ces lectures génèrent des signaux électriques pour l'activation des lignes de mémoire correspondant à l'adresse lue. Des lectures fréquentes provoquent une perturbation électromagnétique qui peut influencer l'état de cellules adjacentes. Plus précisément, ces signaux accélèrent la décharge des condensateurs. Si cette décharge est plus rapide que la période de rafraîchissement de l'état de la cellule, une erreur se produit et l'état de la cellule est modifié.

Ce phénomène était suspecté par les fabricants mais des chercheurs ont démontré qu'il pouvait effectivement être provoqué par l'exécution d'un programme [240], sans pour autant fournir d'attaque exploitant ce phénomène. Par la suite, des chercheurs de Google ont démontré des attaques permettant de contourner les mécanismes d'isolation du processeur [241].

Gruss *et al.* ont démontré que ce type d'attaque pouvait être réalisé depuis un programme JavaScript exécuté dans le navigateur, ce qui permet une exploitation à distance, si l'attaquant parvient à convaincre la victime de naviguer sur un site malveillant [242]. Bosman *et al.* combinent une attaque Rowhammer avec une exploitation du mécanisme de déduplication des pages. Ces attaques leur donnent un accès arbitraire à la mémoire du processus du navigateur, en lecture et en écriture, depuis du code JavaScript [243]. Razavi *et al.* montrent qu'une évolution de cette attaque permet en outre d'accéder à l'intégralité de la mémoire physique [244].

Xiao *et al.* ont démontré qu'une attaque de type Rawhammer pouvait être réalisée entre différentes machines virtuelles afin de contourner l'isolation assurée par l'hyperviseur Xen [245]. Van der Veen *et al.* ont confirmé que cette attaque est également possible sur les *smartphones* utilisant une architecture ARM [246].

Différents mécanismes de protections ont été proposés pour contrer les précédentes attaques. Gruss *et al.* ont toutefois démontré que ces contre-mesures pouvaient être contournées et qu'il était possible de réaliser des attaques Rawhammer furtives depuis une enclave SGX, permettant de réaliser des escalades de privilèges [247].

Cette attaque représente une vulnérabilité sérieuse. Cojocar *et al.* ont proposé une méthodologie permettant à des fournisseurs de *cloud computing* d'évaluer si leur infrastructure est vulnérable à ce type d'attaque [248].

Plus récemment, Kwong *et al.* ont proposé une variante de l'attaque Rawhammer permettant de réaliser une attaque par canal auxiliaire. Cette attaque permet à un processus attaquant de faire fuir des données confidentielles d'un processus à partir du martellement de la mémoire du processus attaquant [249]. En effet, les auteurs démontrent qu'en observant les modifications de l'état des bits de sa propre mémoire, un attaquant peut en déduire l'état des lignes adjacentes de la DRAM.

2.2.4 Attaques contre les enclaves sécurisées

De nombreux travaux se sont intéressés à évaluer la sécurité des mécanismes d'enclaves présentés en section 2.1.4. Ces enclaves doivent faire face à deux types de vulnérabilités :

— le code exécuté dans l'enclave peut comprendre des vulnérabilités logicielles, notamment liées à la gestion de la mémoire ;
— des vulnérabilités matérielles peuvent mettre en défaut l'isolation censée être assurée par le mécanisme d'enclave et permettre ainsi des attaques logicielles par canaux auxiliaires.

2.2.4.1 Exploitation de vulnérabilités logicielles

Les enclaves et les TEE permettent d'isoler des fonctionnalités. Toutefois, le code exécuté dans ces environnements peut comporter des vulnérabilités logicielles. Initialement, l'objectif était d'exécuter des composants logiciels de taille restreinte et ayant un spectre fonctionnel réduit. Toutefois, en pratique, le code exécuté en enclave peut s'avérer relativement volumineux et s'appuyer sur des bibliothèques ou des systèmes d'exploitations (dans le cas des enclaves TrustZone, par exemple). En outre des environnements ont été proposés permettant d'exécuter l'intégralité d'une application dans une enclave (notamment pour SGX). La surface d'attaque peut devenir importante, de même que la TCB. L'exploitation de certaines de ces vulnérabilités peut remettre en cause l'isolation censée être assurée ou les fonctions de sécurité assurées par le code de l'enclave.

En outre, les enclaves telles que SGX ou Sanctum reposent en grande partie sur des services fournis par l'OS, comme l'ordonnancement des fils d'exécution ou la gestion de la pagination. Le modèle d'attaquant est censé garantir que l'OS, qui n'est pas de confiance, ne peut corrompre le code exécuté en enclave. Toutefois, le contrôle qu'il exerce sur les applications exécutées dans l'enclave permet de réaliser ou de faciliter certaines attaques.

Ainsi, Weichbrodt *et al.* montrent que l'OS peut influencer l'ordonnancement des fils d'exécution et amplifier ainsi des vulnérabilités liées à la synchronisation des fils d'exécution de l'enclave [250]. Par exemple, ce type d'attaque permet de mettre en évidence ou d'amplifier des vulnérabilités de type *use-after-free* ou TOCTOU (Time-Of-Check

to Time-Of-Use). L'exploitation de ces vulnérabilités permet ensuite potentiellement de contourner des vérifications réalisées par le code de l'enclave.

Lee *et al.* s'intéressent à la réalisation d'attaques de type ROP, adaptées aux contraintes imposées par SGX. Ces attaques exploitent des vulnérabilités de corruption mémoire dans le code de l'enclave et forcent ce code à réveler des secrets, brisant ainsi les garanties d'isolation de l'enclave [251]. Biondo *et al.* étendent la portée de ce type d'attaques [252]. Leur approche permet de réaliser des attaques depuis l'espace utilisateur et malgré l'utilisation de l'ASLR dans l'enclave SGX.

Cloosters *et al.* ont développé l'outil TeeRex qui permet d'automatiser l'analyse de vulnérabilités du code binaire des applications exécutées dans les enclaves SGX [253]. Les auteurs ont appliqué leur outil avec succès sur plusieurs codes d'enclaves, notamment ceux développés par Intel et Baidu, ce qui leur a permis d'identifier des vulnérabilités dans ces logiciels.

Van Bulck *et al.* s'intéressent à la sécurité des environnements permettant d'exécuter des composants logiciels au sein d'une enclave [254]. Leur étude porte sur différents types d'environnements, qu'il s'agisse de SDK comme ceux fournis par Intel [77], Microsoft [78] ou Google [79] pour exécuter des fonctions au sein d'une enclave SGX ou d'environnements plus complexes de type *library OS* comme Graphene [85] ou SGX-LKL [83], qui permettent d'exécuter une application complète dans l'enclave SGX. Ils s'intéressent également à Keystone [91] (enclave RISC-V) ou Sancus [52] (TEE pour système embarqué). Ils ont révélé la présence de vulnérabilités au niveau des interfaces de ces systèmes avec le code non isolé de l'application. Ils soulignent le fait que ces environnements représentent une surface d'attaque importante. En outre, le filtrage que ces environnements sont censés mettre en œuvre est imparfait. Le filtrage des points d'entrées, situé dans l'interface avec le code non isolé, doit faire l'objet d'une attention particulière.

Machiry *et al.* s'intéressent aux vulnérabilités des TEE TrustZone et de leurs implications pour le système exécuté dans le *Normal World* [255]. Ils démontrent qu'un attaquant peut exploiter l'accès privilégié des applications exécutées dans le TEE. En effet, ces applications ont également accès à l'intégralité de la mémoire des applications et de l'OS exécuté dans le *Normal World*, sachant que l'inverse n'est pas vrai. Ainsi, un attaquant peut utiliser une application du TEE pour accéder à la mémoire d'une autre application exécutée dans le *Normal World*, en exploitant une forme de *confused deputy attack*. Ils ont pu vérifier la faisabilité de ce type d'attaques sur plusieurs TEE disponibles sur le marché dont OP-TEE, QSEE et le TEE développé par Huawei.

Une approche similaire [256] a été développée par Suciu *et al.* . Toutefois, ces travaux ne supposent pas la présence de vulnérabilités dans une application sécurisée. Les auteurs ont en outre développé un outil leur permettant d'automatiser la recherche de vulnérabilités dans les applications sécurisées. Ils ont ainsi pu identifier 19 vulnérabilités parmi les applications sécurisées de trois TEE (Kinibi, QSEE, et Teegris). L'exploitation de ces vulnérabilités permet à des applications du *Normal World* d'obtenir des données appartenant à d'autres applications.

Di Shen présente également comment Knox, la solution de Samsung permettant de vérifier l'intégrité du noyau Linux, peut être contournée afin d'exploiter une vulnérabilité dans le noyau et *in fine* obtenir les droits root sur le téléphone [257]. Les ingénieurs de la société Quarkslab ont également identifié plusieurs vulnérabilités dans Kinibi, le TEE développé

par Trustonic et utilisé sur certains téléphones Samsung [258].

Wang *et al.* s'intéressent plus particulièrement à la sécurité des *Isolated Execution Environments* [259]. Ces environnements sont des enclaves qui s'exécutent dans le *Normal World* mais qui sont isolées des autres applications et de l'OS, grâce à un moniteur de sécurité qui s'exécute dans le *Secure World*. Cette approche est notamment mise en œuvre par TrustICE [67] et SANCTUARY [68], présentés en section 2.1.4.1. Elle vise à répondre aux limitations de déploiement des applications dans TrustZone. Toutefois, les auteurs montrent que ces solutions sont vulnérables à des attaques visant le cache, qui n'est pas protégé, contrairement aux applications s'exécutant au sein de TrustZone.

Ces attaques sont différentes des attaques par canaux auxiliaires utilisant le cache. Elles exploitent le fait que l'attaquant maîtrise l'OS et notamment le mécanisme de pagination. Un module noyau malveillant peut donc configurer une page mémoire pour qu'elle corresponde à des adresses physiques utilisées par une enclave. Même si le moniteur empêche l'accès à la mémoire, l'attaquant peut lire les données présentes dans le cache.

2.2.4.2 Attaques par canaux auxiliaires contre les enclaves

De nombreux travaux ont démontré que l'isolation fournie par les mécanismes d'enclaves n'était en général pas robuste aux attaques par canaux auxiliaires. Ce résultat est, pour certaines enclaves (notamment SGX), attendu car elles ne prennent pas en compte ce type de menaces dans leur modèle d'attaquant.

Xu *et al.* présentent un nouveau type d'attaques par canal auxiliaire, les *controlled-channel attacks*, qui s'appliquent aux mécanismes de TEE (notamment SGX) qui s'appuient sur les services de l'OS (gestion des interruptions, de la pagination, de l'ordonnancement, etc,) [260]. Habituellement, dans les attaques par canaux auxiliaires, on ne suppose pas que l'OS soit sous le contrôle de l'attaquant. Il est donc source de bruit qu'il faut filtrer. Dans le contexte des enclaves, l'OS est supposé potentiellement malveillant. Dès lors, il est possible de construire des attaques par canaux auxiliaires très précises, sans bruit lié à l'OS. Les auteurs utilisent le mécanisme de défaut de page pour mettre en place leur attaque. Ils ont notamment appliqué cette attaque à Haven [82], qui est un environnement d'exécution de type *library OS* pour SGX.

Toutefois, des contre-mesures logicielles sont possibles pour se prémunir des attaques précédentes qui exploitent les défauts de page. Moghimi *et al.* utilisent une attaque de type PRIME+PROBE, qui exploite les accès au cache, et démontrent que la maîtrise de l'OS permet à l'attaquant de réaliser des attaques par canaux auxiliaires [261]. Ces attaques permettent de suivre tous les accès mémoire réalisés par le code de l'enclave avec une grande précision. De même, Van Bulck *et al.* utilisent d'autres formes de canaux auxiliaires issus de la traduction d'adresse, qui ne génèrent pas de défaut de page et ne sont pas protégés par les contre-mesures logicielles [262]. Wang *et al.* dressent un bilan de ces différents travaux et réalisent une analyse détaillée des vulnérabilités potentielles de SGX par rapport à ce type d'attaque et de la robustesse des contre-mesures proposées jusqu'à présent [263]. Ils proposent de nouvelles attaques et montrent que les contre-mesures proposées ne permettent pas de lutter efficacement contre ces attaques.

Plus récemment, Moghimi *et al.* ont démontré qu'il était possible de diminuer la granularité des *controlled-channel attacks* utilisant le défaut de page en exécutant les applications en pas-à-pas [264].

Lee *et al.* ont proposé un autre type de canal caché visant les enclaves SGX [265]. Ils utilisent le *Last Branch Record*, un mécanisme fourni par les processeurs Intel afin de tracer les derniers branchements pris par un programme. Cette fonctionnalité a été introduite pour des besoins de profilage des applications. Malheureusement, ce *buffer* n'est pas nettoyé lors de la sortie d'une enclave SGX. Les auteurs ont démontré que ce mécanisme pouvait être utilisé pour réaliser une attaque par canaux auxiliaires qui révèle des informations sur le flot de contrôle du code de l'enclave. Les auteurs ont ainsi pu récupérer une clé RSA manipulée par le code de l'enclave.

Van Bulck *et al.* ont proposé NEMESIS [266], un nouveau type d'attaque par canal auxiliaire utilisant le temps nécessaire au traitement des interruptions dans le processeur. Le mécanisme utilise le même phénomène qui est exploité dans les attaques de type Meltdown. Toutefois, il ne s'agit pas ici de tirer profit de l'exécution transitoire des instructions qui en découle mais de mesurer le temps pris par le traitement des interruptions, afin d'inférer l'état de la microarchitecture dans l'enclave. Les auteurs ont notamment appliqué leurs travaux à SGX et Sancus [52].

Toutes les attaques par canaux auxiliaires contre les enclaves SGX, présentées précédemment, nécessitent la présence d'un code vulnérable qui réalise des accès mémoire en fonction de la valeur d'un secret. Wang *et al.* ont proposé une approche permettant d'identifier automatiquement les attaques par canaux auxiliaires qu'il est possible d'utiliser à partir de l'analyse dynamique du code binaire exécuté dans l'enclave [267].

Des attaques exploitant l'exécution transitoire ont également été démontrées contre SGX.

Les auteurs de NEMESIS ont par la suite proposé Foreshadow [268], une attaque exploitant l'exécution transitoire et permettant de faire fuir les données d'une enclave se trouvant dans le cache L1 du processeur vers un processus utilisateur. Il s'agit d'une variante des vulnérabilités de type Meltdown [217]. Contrairement aux attaques précédentes, cette attaque ne suppose pas la présence d'un code vulnérable dans l'enclave. Les auteurs ont notamment pu la mettre en œuvre pour récupérer la clé privée utilisée par le mécanisme d'attestation à distance de SGX.

Chen *et al.* ont également proposé des attaques de type Spectre qui exploitent des gadgets présents dans la plupart des environnements d'exécution pour SGX [269].

Des attaques par canaux auxiliaires ont également été mis en évidence contre les TEE protégés par TrustZone.

Lapid *et al.* ont ainsi démontré que l'implémentation AES utilisée dans l'application Keymaster du TEE Kinibi, utilisée sur certains téléphones Samsung, était vulnérable à une attaque par canaux auxiliaires utilisant le cache [270].

Cho *et al.* ont proposé Prime+Count [271], un nouveau type de canal caché entre une application sécurisée et le *Normal World*, qui utilise le cache partagé entre les deux mondes TrustZone. L'objectif est de pouvoir contourner les mécanismes de sécurité qui ont été ajoutés afin de filtrer les messages provenant du *Normal World* et d'authentifier les applications qui peuvent légitimement envoyer ces messages et communiquer avec le *Secure World* [65].

2.2.5 Rétroconception de la microarchitecture

La mise au point des attaques contre la microarchitecture nécessite bien souvent de connaître précisément les détails d'implémentation. Toutefois, ces détails sont souvent peu documentés car ils font partie des secrets industriels des constructeurs. Il faut donc dans ce cas réaliser une étape de rétroconception ou de modélisation de cette microarchitecture.

2.2.5.1 Modélisation et évaluation des attaques contre la microarchitecture

Un certain nombre de travaux se sont intéressés à développer des outils permettant d'analyser les attaques contre la microarchitecture afin de mieux les comprendre et d'obtenir des éléments quantitatifs sur ces attaques, par exemple sur le nombre d'instructions qu'il est possible d'exécuter de manière transitoire.

Zhang *et al.* ont notamment proposé un outil permettant d'analyser les attaques par canaux auxiliaires contre les caches [272]. L'outil permet d'analyser différentes techniques (Prime-Probe, Evict-Time, Flush-Reload, Flush-Flush, Prime-Abort) et d'évaluer les performances des contre-mesures. Pour cela, les auteurs utilisent des techniques d'apprentissage à base de réseau de neurones.

Abel *et al.* ont développé uops.info [273], un outil permettant de construire des modèles fiables de la latence, du débit et de l'utilisation des ports des unités d'exécution pour les instructions de l'architecture x86. Cet outil a été développé à des fins d'optimisation de code mais il peut fournir des informations intéressantes pour le développement d'attaques contre la microarchitecture.

Xiao *et al.* ont proposé SPEECHMINER, un outil permettant d'analyser les attaques exploitant l'exécution transitoire [274]. Leur approche regroupe dans un même outil différents tests permettant d'évaluer différents types d'attaques et de les mesurer.

2.2.5.2 Rétroconception et modélisation des caches

Les attaques par canaux auxiliaires utilisant le cache nécessitent de connaître précisément l'organisation de la hiérarchie des différents niveaux de caches, sur une microarchitecture donnée.

Ainsi Abel *et al.* ont réalisé une étude sur les politiques de remplacement des caches à l'aide d'un *microbenchmarck* développé spécifiquement [275]. Cette étude a été réalisée dans l'objectif de modéliser les performances des systèmes mais ce type d'approche fournit des informations intéressantes pour la sécurité.

Maurice *et al.* ont réalisé la rétroconception de l'adressage du LLC des processeurs à l'aide des compteurs de performances [276]. Cette technique est nécessaire car l'adressage des LLC fait maintenant appel à des fonctions complexes et non documentées.

Green *et al.* ont révélé la présence d'AutoLock [277], un mécanisme présent sur certains SoC ARM qui améliore les performances des caches mais peut gêner la réalisation des attaques par canaux auxiliaires. Cette fonctionnalité n'était pas documentée publiquement et les auteurs ont du rétroconcevoir cette fonctionnalité. Ils documentent la fonctionnalité

et fournissent des méthodes permettant de tester si la fonctionnalité est présente sur un SoC donné.

Youngjoo *et al.* proposent une méthodologie permettant d'identifier des attaques par canal caché pour une implémentation donnée d'un algorithme [278]. L'approche développée a permis aux auteurs d'identifier un nouveau type d'attaque par canal caché exploitant les accès réalisés par le *Instruction Pointer-based stride prefetcher*. Cet élément de la microarchitecture des processeurs Intel va inférer les adresses des prochaines instructions afin de précharger le cache avec les instructions en question.

Les attaques par canal caché utilisant le LLC reposent généralement sur la capacité de l'attaquant à pouvoir évincer les différentes lignes d'un ensemble de ce cache. Cela nécessite pour l'attaquant de pouvoir déterminer des ensembles minimaux d'éviction. Il s'agit d'ensembles d'adresses virtuelles qui, lorsqu'on réalise des accès en lecture à chacune des adresses qui les composent, permettent d'évincer complètement un ensemble donné du cache. Calculer explicitement cet ensemble est difficile, d'autant que l'adressage du LLC utilise les adresses physiques et que l'attaquant ne dispose pas toujours des privilèges du noyau lui permettant de retrouver la correspondance entre adresses virtuelles et physiques. Song *et al.* proposent donc des heuristiques pour déterminer ces ensembles [279]. Ils montrent que leur approche permet de construire ces ensembles plus rapidement que prévu.

Kurth *et al.* ont réalisé la rétroconception du DCA (Direct Cache Access) pour les processeurs Intel [280]. Cette fonctionnalité permet aux périphériques de réaliser des accès directement dans le LLC, ce qui améliore sensiblement les performances. Toutefois, les auteurs montrent que ce mécanisme peut être utilisé pour réaliser une attaque par canaux auxiliaires utilisant un périphérique. Ils montrent notamment qu'une carte réseau peut réaliser des attaques de type PRIME+PROBE.

2.2.5.3 Rétroconception et modélisation de la mémoire

Des travaux de rétroconception ont également été réalisés sur l'organisation de la mémoire à différents niveaux.

Pessl *et al.* ont notamment réalisé un travail de rétroconception pour modéliser l'organisation de la mémoire physique sur un ordinateur disposant de plusieurs processeurs [281]. Sur ce type de configuration, l'exploitation d'un canal caché utilisant le cache n'est pas possible car la victime est souvent localisée sur un processeur différent de celui utilisé par l'attaquant. Les auteurs montrent qu'un canal auxiliaire utilisant le *DRAM row buffer*, qui est partagé entre les différents processeurs, est possible. Pour cela ils proposent deux techniques pour rétroconcevoir l'association d'une adresse mémoire avec la mémoire physique, afin de déterminer notamment les *channel*, *rank* et *bank* correspondant à une adresse.

De même, Tatar *et al.* ont développé RAMSES [282], un outil permettant de modéliser la correspondance entre les adresses physiques et l'organisation de la DRAM. Cet outil a été réalisé à partir d'informations disponibles sur les sites des constructeurs, complétées par un travail de rétroconception. Les auteurs utilisent cet outil pour améliorer à la fois les attaques Rowhammer et leurs contre-mesures.

Les ECC (Error-Correcting Code, code correcteur d'erreurs) ont été considérés comme une contre-mesure possible face aux attaques Rowhammer. Afin de vérifier si ce méca-

nisme est efficace, Cojocar *et al.* ont proposé une technique permettant de rétroconcevoir l'algorithme utilisé dans les ECC de certaines mémoires DRAM [283]. A l'aide de ce travail, ils démontrent que des attaques Rowhammer sont toujours possibles, en dépit de l'ECC.

Frigo *et al.* ont réalisé un travail de rétroconception du TRR (Target Row Refresh), un ensemble de contre-mesures contre Rowhammer implémenté dans les circuits imprimés de DRAM [284]. Ce travail leur permet d'évaluer la robustesse du mécanisme. Les auteurs démontrent que des variantes de l'attaque Rowhammer sont toujours possibles sur les circuits de DRAM, y compris sur les modèles les plus récents qui sont protégés par le TRR.

Des travaux se sont également intéressés à retrouver la correspondance entre adresses virtuelles et physiques. Cette information est utile pour un attaquant qui vise les caches adressés physiquement et qui ne dispose pas des privilèges du noyau de OS. Islam *et al.* ont notamment proposé d'exploiter le mécanisme de résolution des dépendances entre lectures et écritures mémoire, utilisé par le processeur pour spéculer les lectures, afin d'accélérer la rétroconception de la correspondance entre adresses physiques et virtuelles [285].

2.2.5.4 Rétroconception du microcode et du décodage du jeu d'instruction

Les processeurs x86 développés par Intel et AMD utilisent un étage de traduction des instructions en micro-instruction. Cet étage réalise la traduction à l'aide d'un *microcode* chargé dans une mémoire interne du processeur. Chaque processeur possède une version originale du *microcode*, située dans une mémoire ROM, qui a été développée lors de sa fabrication. Toutefois, il est possible de mettre à jour logiciellement ce *microcode*, lors de l'exécution. Les processeurs comprennent un mécanisme permettant de mettre à jour la mémoire interne utilisée pour la traduction, à partir d'une nouvelle version du *microcode* située dans la DRAM. Cette mise à jour est éphémère et sera perdue lors de l'extinction du processeur. Il est donc nécessaire de recommencer le processus à chaque redémarrage de la machine.

L'intégrité du *microcode* est fondamentale pour la sécurité du processeur car certaines instructions, notamment celles permettant de gérer les enclaves SGX, sont entièrement réalisées par des programmes du *microcode*. Les mises à jour du *microcode* permettent en outre de corriger certaines erreurs des processeurs ou d'implémenter des contre-mesures.

Afin de se prémunir des attaques qui viseraient à charger un *microcode* malveillant ou défectueux, celui-ci est signé et le processeur vérifie la signature avant le chargement. En outre, sur les versions récentes des processeurs, ce *microcode* est chiffré. Son format est propriétaire et les constructeurs ne documentent pas précisément les opérations réalisées par le *microcode*.

Des chercheurs se sont intéressés à la rétroconception du *microcode*. Cette étape permet notamment d'analyser les contre-mesures proposées par les constructeurs et d'identifier de potentielles vulnérabilités voire des portes dérobées placées par les constructeurs.

Des chercheurs de l'université de Bochum se sont notamment intéressés à la rétroconception du *microcode* des processeurs AMD [286, 287]. Leur travail s'est d'abord focalisé sur la la procédure de mise-à-jour du *microcode* puis sur la rétroconception du format. Ils proposent également d'implémenter différents mécanismes, légitimes ou malveillants, en

modifiant ce *microcode*. Toutefois, ce travail ne peut être réalisé que sur d'anciennes versions des processeurs AMD. En effet, dans les versions plus récentes des microprocesseurs, les constructeurs signent et chiffrent leur *microcode*.

Récemment, des ingénieurs de l'entreprise russe Positive Technology ont toutefois réussi à obtenir une version en clair du *microcode* des processeurs Intel Atom. Ils ont pour cela exploité des vulnérabilités dans certaines cartes mères et dans le *firmware* du CSME qui permettent de déverrouiller la protection empêchant la mise au point du processeur et du *chipset*, via le protocole USB [288]. L'exploitation de ces vulnérabilités leur permet *in fine* d'accéder à un bus interne des processeurs, d'extraire le *microcode* et même de le modifier sur le processeur [289]. Cela nécessite toutefois que le processeur soit dans un état spécifique de mise au point et de disposer d'un accès physique à l'un des ports USB de la machine. Les auteurs ont depuis initié un travail de rétroconception de ce *microcode*.

Christopher Domas s'est également intéressé à la sécurité des microprocesseurs au niveau de l'ISA afin d'identifier des instructions non documentées ou des portes dérobées [290, 291]. Toutefois, les résultats de ces travaux concernent des versions obsolètes de processeurs implémentant l'architecture x86.

2.3 Contre-mesures logicielles face aux attaques contre la microarchitecture

Face aux différentes attaques contre la microarchitecture, qui exploitent des vulnérabilités matérielles, des contre-mesures logicielles ont été proposées. Ces contre-mesures sont essentielles quand le matériel ne peut être mis à jour. En outre, certaines contre-mesures matérielles nécessitent d'être configurées par le logiciel.

Certaines contre-mesures permettent de lutter contre plusieurs types d'attaques. Ainsi, Oliverio *et al.* ont proposé un mécanisme robuste de déduplication des pages qui permet de se prévenir ou de complexifier à la fois des attaques par canaux auxiliaires utilisant le cache et des attaques Rawhammer [292].

De même le projet JavaScript Zero vise à limiter les possibilités offertes par ce langage qui permettent de réaliser différentes attaques contre la microarchitecture [293]. Ces protections sont efficaces contre un large spectre de vulnérabilités mais ne concernent que leur exploitation depuis JavaScript.

Irazoqui *et al.* ont également développé MASCAT, un outil d'analyse statique des programmes qui visent à détecter un large spectre d'attaques contre la microarchitecure [294]. L'objectif est de fournir un outil qui pourrait être utilisé par les fournisseurs de *cloud computing* ou d'applications pour *smartphone* afin de vérifier si les applications sont malveillantes avant de les déployer ou de les publier.

Afin de lutter efficacement contre les attaques logicielles par canaux auxiliaires et celles utilisant l'exécution transitoire, il apparaît de plus en plus nécessaire d'établir un nouveau contrat entre le logiciel et le matériel. En effet, certaines contre-mesures nécessitent un support matériel. Toutefois, ces contre-mesures nécessitent un paramétrage qui doit être réalisé par le logiciel. Cela permet entre autres de limiter l'impact des contre-mesures en ne les utilisant que lorsque cela est nécessaire. En outre, ces contre-mesures nécessitent de définir des domaines d'isolation ou d'identifier des données confidentielles à protéger. En

général, ces éléments dépendent du contexte et ne peuvent être spécifiés que par le code de l'application ou celui de l'OS.

Ge *et al.* ont notamment milité pour l'établissement d'un tel contrat afin que l'OS puisse assurer une isolation temporelle [295, 296]. Plus récemment, Guarnieri *et al.* ont proposé une approche reposant sur un tel contrat pour lutter contre l'exécution spéculative [297].

La plupart des contre-mesures logicielles sont spécifiques à une classe d'attaques. Par la suite, nous détaillons les contre-mesures logicielles qui ont été proposées pour lutter contre les attaques logicielles par canaux auxiliaires (section 2.3.1). Nous évoquons ensuite en section 2.3.2 les contre-mesures qui visent plus spécifiquement les attaques exploitant l'exécution transitoire dont l'exploitation d'un canal auxiliaire n'est qu'une étape. Enfin nous présentons les protections logicielles permettant de lutter contre les attaques Rawhammer 2.3.3.

2.3.1 Contre-mesures logicielles face aux attaques par canaux auxiliaires

Des travaux se sont intéressés à dresser un état de l'art des protections contre les attaques par canaux auxiliaires temporels [298] et plus spécifiquement celles utilisant le cache [299].

Par la suite nous détaillons différents types de contre-mesures logicielles :

- l'analyse statique et dynamique des programmes afin d'identifier les programmes vulnérables à ces attaques ou de les protéger (section 2.3.1.1) ;
- les contre-mesures liées à la gestion du cache (section 2.3.1.2) ;
- les contre-mesures au niveau de l'OS (section 2.3.1.3) ;
- la détection des attaques par canaux auxiliaires (section 2.3.1.4).

2.3.1.1 Identification et protection des programmes vulnérables

Les attaques par canaux auxiliaires reposent généralement sur la présence de vulnérabilités dans le code des programmes (applications, OS, etc.) qui réalisent des opérations (notamment des accès mémoire) qui dépendent de données secrètes. Des approches ont été proposées afin d'identifier ces vulnérabilités par analyse statique ou dynamique ou de protéger le code des applications contre ces vulnérabilités.

Une des approches permettant d'éviter ces vulnérabilités consiste à développer des applications *constant-time*, c'est à dire dont le temps d'exécution ne doit pas varier en fonction des valeurs des secrets. Cela implique que les accès mémoire qui utilisent le cache partagé ne doivent pas dépendre des valeurs secrètes. Cette approche doit être appliquée aux programmes critiques pour la sécurité, notamment ceux réalisant des opérations cryptographiques, mais elle est difficile à généraliser à tous les programmes. En outre, ces approches dépendent de la configuration de la microarchitecture car, par exemple, les temps d'exécution des instructions peuvent varier d'une microarchitecture à une autre.

Analyses statiques Différentes approches d'analyse statique ont été proposées pour lutter contre les attaques par canaux auxiliaires. Ainsi, Doychev *et al.* ont proposé CacheAudit [300], un outil d'analyse statique qui permet de vérifier si des programmes sont

vulnérables à des attaques par canaux auxiliaires utilisant le cache. Le programme analyse le code binaire des applications et repose sur une description de la configuration des caches. Il réalise une surapproximation des canaux possibles afin de s'assurer qu'aucune fuite n'est possible. Il peut en revanche refuser des programmes qui ne sont pas, en pratique, vulnérables à ces attaques.

Barthe *et al.* ont formalisé une propriété de non-interférence qui permet de capturer les canaux auxiliaires utilisant le cache [301]. Cette propriété garantit l'absence de fuite d'information pour des programmes qui appliquent une approche *constant-time* et utilisent une mémoire dédiée pour manipuler des secrets. Les auteurs proposent également une analyse statique permettant de garantir cette propriété. Ils apportent la preuve de la correction de l'analyse à l'aide de l'assistant à la preuve Coq.

L'outil Valgrind d'analyse du code binaire des application dispose d'un module permettant de vérifier que le code d'une fonction est *constant-time* [302]. Almeida *et al.* ont également proposé une approche permettant de vérifier automatiquement la propriété de *constant-time* au niveau du compilateur LLVM [303].

Analyses dynamiques Des approches dynamiques ont également été proposées afin d'analyser dynamiquement les applications et d'identifier des vulnérabilités aux attaques par canaux auxiliaires. Ainsi Wang *et al.* ont proposé un outil permettant d'identifier de potentielles vulnérabilités à l'aide d'une analyse des traces de l'application, d'une exécution symbolique et d'un modèle du cache [304].

Xiao *et al.* proposent une analyse différentielle qui permet dynamiquement d'identifier de potentielles vulnérabilités dans les implémentations TLS exécutées dans l'enclave SGX [305]. Toutefois l'approche se concentre sur les fuites liées au flot de contrôle. Weiser *et al.* adoptent une approche similaire qui prend également en compte les fuites liées au données [306].

Wichelmann *et al.* utilisent l'instrumentation dynamique de binaires et une analyse de l'information mutuelle pour identifier des vulnérabilités dans le code binaire d'implémentations propriétaires [307].

Wang *et al.* proposent une analyse des traces, obtenues avec le mécanisme Intel PT, afin d'identifier des canaux auxiliaires qui sont liés aux variations temporelles entre deux accès mémoire, à l'aide d'une analyse de graphes [308].

Diversification Des approches de diversification, qui visent à faire varier aléatoirement les accès mémoire et le flot d'exécution d'un programme à chaque exécution, ont également été proposées. Ces approches visent à générer du bruit, afin de masquer les différences mesurables dans l'exécution d'un programme qui sont liées à des fuites d'information. Ainsi, Crane *et al.* proposent de générer différentes variantes des fonctions d'un programme, appelées fragments, puis de compiler l'application de manière à ce qu'elle choisisse aléatoirement le fragment à exécuter pour chaque exécution du programme [309].

De même, Rane *et al.* proposent une technique d'obfuscation au niveau du code source du programme, afin de donner l'illusion que différents chemins d'exécution ont été empruntés [310].

Brasser *et al.* proposent une technique de déplacement aléatoire des données lors de

l'exécution, afin que l'attaquant ne puisse inférer les données en fonction des accès mémoire [311]. Ils implémentent automatiquement leur approche au niveau du compilateur afin de protéger le code des applications exécutées en enclaves SGX.

2.3.1.2 Gestion du cache

Les différents caches sont utilisés par de nombreuses attaques par canaux auxiliaires. Même si d'autres canaux existent, ils constituent une source de vulnérabilités importante. Des approches ont donc été proposées pour empêcher ou limiter les canaux auxiliaires utilisant le cache.

Une première approche consiste à verrouiller ou partitionner le cache. Ainsi, STEALTHMEM [312] propose un système, implémenté au niveau de l'OS, qui permet de verrouiller certaines lignes de caches, qui ne sont jamais évincées. Ces lignes peuvent être utilisées par les applications pour y stocker des données confidentielles.

Zhou *et al.* s'intéressent à la protection du partage de pages physiques entre différentes machines virtuelles [313]. Ce partage est utilisé par de nombreuses attaques par canaux auxiliaires qui visent le LLC, notamment FLUSH+RELOAD. Afin de permettre ce partage, les auteurs implémentent un mécanisme de sécurité au niveau de l'OS afin de s'assurer que ces pages ne partagent pas les mêmes lignes du LLC.

Des approches ont également proposé d'utiliser la technologie Intel CAT [314] qui permet de partitionner le LLC. Il s'agit d'une fonctionnalité matérielle mais qui doit être configurée par le logiciel. Cette technologie a été introduite pour limiter la latence des applications qui le nécessitent. Liu *et al.* ont cependant démontré qu'elle pouvait également être utilisée pour protéger les applications des attaques par canaux auxiliaires [315].

CAT est un mécanisme qui a l'avantage d'être disponible sur les processeurs x86 existants. Toutefois, il ne fournit pas un partitionnement strict du cache. Kiriansky *et al.* ont proposé DAWG, une approche qui implémente un partitionnement robuste du cache mais nécessite quelques modifications au niveau matériel [316]. Un support au niveau de l'OS est également nécessaire pour isoler les processus dans différents domaines.

Les approches de partitionnement du cache ont un impact sur les performances puisque la taille du cache disponible pour une application est réduite. Les approches précédentes impactent l'intégralité du code des applications. Dessouky *et al.* proposent une approche à grain fin permettant de partitionner dynamiquement le cache, seulement lorsque des composants logiciels critiques, qui manipulent par exemple des clés cryptographiques, sont exécutés [317]. Cela permet, le reste du temps, d'éviter d'impacter les applications qui ne contiennent pas de données confidentielles. Leur approche nécessite des modifications matérielles et un support au niveau de l'OS.

Une autre approche consiste à introduire de l'aléa dans la fonction permettant d'associer une adresse physique à un ensemble de lignes du cache. Cette contre-mesure permet de lutter contre les attaques qui doivent réaliser l'éviction d'un ensemble de lignes du LLC (notamment EVICT+RELOAD et PRIME+PROBE). Toutefois ces contre-mesures nécessitent des modifications importantes au niveau matériel ou des changements fréquents de clés. Werner *et al.* ont proposé une approche qui repose sur une clé et un identifiant du domaine de sécurité [318]. Cette approche nécessite toutefois des modifications matérielles et un support logiciel pour associer les domaines de sécurité aux composants logiciels.

L'utilisation de la mémoire transactionnelle, notamment fournie par Intel TSX, permet également, par effet de bord, de contrer les attaques par canaux auxiliaires utilisant le cache [319].

2.3.1.3 Contre-mesures au niveau de l'OS

Certaines attaques par canal auxiliaire visent plus particulièrement l'OS ou un des services qu'il fournit. Des contre-mesures logicielles ont donc été proposées au niveau du noyau de l'OS.

Protection de l'ASLR Certaines attaques par canaux auxiliaires, présentées en section 2.2.1.1, visent à casser l'ASLR mise en place au niveau du noyau de l'OS [208-210]. Ce type d'attaques permet ensuite de réaliser facilement des attaque de type ROP exploitant des vulnérabilités du noyau.

Des contre-mesures ont été proposées pour lutter contre ces attaques, notamment KAISER [320] et LAZARUS [321]. KAISER supprime les pages du noyau de l'espace virtuel des applications. Les auteurs ont proposé une modification du noyau Linux implémentant cette approche. Cette modification a également permis de lutter efficacement contre les attaques Meltdown, pour les processeurs qui sont vulnérables à cette attaque. Elle a été adoptée par l'équipe de développement du noyau Linux dans sa solution KPIT [322].

KPTI est toutefois désactivée par défaut pour les processeurs qui ne sont pas vulnérables, en raison de son impact sur les performances. Cependant, Canella *et al.* ont démontré que les contre-mesures matérielles des derniers processeurs Intel permettaient tout de même de casser l'ASLR [323]. Les auteurs ont proposé une nouvelle contre-mesure logicielle plus robuste afin de lutter contre ce type d'attaques.

Génération de bruit Zhang *et al.* proposent que l'OS nettoie régulièrement les caches L1 et L2 afin d'ajouter du bruit, ce qui limite l'efficacité des attaques PRIME+PROBE [324]. Leur objectif est de fournir un moyen d'autodéfense pour les machines virtuelles déployées dans le *cloud*, sans nécessiter d'effort spécifique pour le développement des applications.

Varadarajan *et al.* combinent le nettoyage du cache L1 avec des contraintes sur l'ordonnancement qui imposent une durée minimale d'exécution d'une machine virtuelle [325].

HYPERRACE [326] vise à protéger les enclaves SGX contre les attaques par canal caché de type *port contention*. Ces attaques doivent être conduites par un programme s'exécutant simultanément sur le même cœur que l'enclave. Cela est possible en raison de l'*hyperthreading*. Les auteurs proposent d'exécuter un fil d'exécution maîtrisé sur le même cœur physique que l'enclave et de s'assurer que ce fil d'exécution est bien colocalisé avec le code de l'enclave. La solution proposée évite de désactiver l'*hyperthreading* de manière globale, ce qui impacte les performances de toutes les applications. Cette contre-mesure nécessite le support de l'OS pour ordonnancer les fils d'exécution mais ne suppose pas que l'OS est de confiance, grâce au test de colocalisation.

2.3.1.4 Détection des attaques contre le cache

Les attaques par canaux auxiliaires qui utilisent le cache génèrent un comportement d'accès au cache anormal, notamment ence qui concerne le ratio entre les *cache hit* et les *cache*

miss. Des approches ont proposé de détecter ces comportements.

CloudRadar [327] combine une détection par signature, qui permet d'identifier l'exécution d'une machine virtuelle exécutant une application à protéger (typiquement, une application réalisant des opérations cryptographiques) et une détection d'anomalies afin d'identifier des attaques potentielles. L'approche utilise les compteurs de performances pour réaliser la détection. La corrélation entre les deux types de détection permet d'identifier des attaques par canaux auxiliaires contre l'application.

CacheShield [328] propose une approche similaire mais ne nécessite que de surveiller l'application cible. De même Mushtaq *et al.* évaluent les capacités de détection d'une telle approche dans des conditions proches de celles rencontrées dans un environnement de type *cloud*, en présence de différentes applications et d'une charge système réaliste [329].

2.3.2 Protections contre les attaques exploitant l'exécution transitoire

Les attaques exploitant l'exécution transitoire sont à distinguer des attaques par canaux auxiliaires. En effet, l'utilisation d'un canal auxiliaire ne représente que la dernière étape de ce type d'attaque. En outre, ce canal est mis en place directement par le code de l'attaque (dans le cadre de Meltdown) ou par du code contrôlé par l'attaquant (dans le cadre de Spectre). Cette dernière étape correspond donc plus à un canal caché.

De ce fait, certaines contre-mesures contre les canaux auxiliaires ne permettent pas de lutter contre les attaques exploitant l'exécution transitoire. Ainsi, la programmation *constant-time* ne concerne que le code légitime de l'application et ne permet pas, en l'état, de lutter contre les attaques de type Spectre ou Meltdown.

Canella *et al.* dressent un état de l'art de ces différentes attaques et des contre-mesures qui ont été proposées pour s'en prémunir [217].

Guanciale *et al.* établissent un modèle formel de la microarchitecture permettant de représenter différentes formes d'exécution spéculative et *out-of-order* pour un processeur monocœur [330]. Ce modèle permet d'identifier de nouvelles vulnérabilités et de raisonner formellement sur le niveau de protection des contre-mesures.

2.3.2.1 Identification des codes vulnérables

Les attaques de type Spectre reposent sur la présence, dans le code de l'application victime, de suites d'instructions, les *gadgets*, que l'attaquant va chercher à faire exécuter de manière spéculative.

Des travaux se sont donc intéressés à identifier la présence de ces *gadgets* dans le code des applications afin d'évaluer leur état de vulnérabilité.

Ainsi, Wang *et al.* utilisent le moteur d'exécution symbolique KLEE qui, avec un modèle du cache et de l'exécution spéculative, leur permet d'identifier dans un programme donné, des fuites liées à l'exécution spéculative [331].

Oleksenko *et al.* utilisent une approche de *fuzzing* afin de tester une application et d'identifier les variantes V1 de Spectre (Spectre-PHT) [332].

Speculator [333] est un outil qui permet d'étudier le comportement spéculatif d'une suite d'instructions à l'aide des compteurs de performance. Il s'agit d'un outil d'analyse dynamique qui ne permet pas d'identifier directement les vulnérabilités dans le code d'une application mais plutôt d'étudier précisément le comportement spéculatif afin d'identifier de nouvelles attaques ou d'étudier l'efficacité des contre-mesures.

2.3.2.2 Prévention des attaques de type Spectre

Différentes approches ont été proposées afin de se protéger des attaques de type Spectre. A la différence des attaques de type Meltdown, les attaques de type Spectre peuvent difficilement être résolues par une approche purement matérielle, sans modifications profondes de la microarchitecture qui impacteraient fortement les performances.

Ainsi, Intel [334], ARM [335] et AMD [336] ont publié des recommandations à l'attention des développeurs afin de lutter contre ces attaques, suivant les possibilités offertes par leurs architectures respectives et leur niveau de vulnérabilité.

L'équipe de développement du noyau Linux a notamment mis en place certaines de ces contre-mesures, qu'il est possible d'activer ou de désactiver [337]

Ces approches peuvent être classées selon trois catégories, qui correspondent aux différentes étapes de l'attaque :

— supprimer ou limiter la spéculation ;
— empêcher le code spéculé à accéder à des données sensibles ;
— supprimer ou limiter le canal caché ;

Suppression ou contrôle de la spéculation La spéculation est la cause première des attaques de type Spectre. Une approche naïve pour les contrer serait de supprimer la spéculation. Cette approche ne peut malheureusement être retenue que pour des microarchitectures simples, pour des cas d'usage qui ne nécessitent pas de performances importantes.

Afin de limiter l'impact de cette contre-mesure, les constructeurs recommandent de la supprimer localement aux endroits du code potentiellement dangereux. Ce contrôle de la spéculation peut être effectué à l'aide d'instructions de sérialisation comme `LFENCE` pour les architectures x86 ou `CSDB` pour les architectures ARM qui le supportent.

Toutefois ces instruction ne bloquent pas l'exécution spéculative de toutes les instructions mais seulement de celles qui réalisent des lectures en mémoire. Elles sont donc seulement efficaces contre les attaques qui utilisent le cache comme canal caché. En outre, il s'agit d'une protection locale. Elle n'est donc pas efficace pour lutter contre Spectre-BTB, pour laquelle l'attaquant peut sauter à n'importe quel endroit de la mémoire. Elle est surtout utile pour se protéger de la variante 1 (Spectre-PHT).

En outre, cette solution impacte sensiblement les performances. Des approches ont donc été proposées pour identifier précisément les branchements qui sont susceptibles d'être vulnérables afin de ne pas déployer cette solution pour tous les branchements.

Microsoft adopte une approche par liste noire. Le compilateur place des protections uniquement lorsqu'il détecte des motifs de code qui pourraient permettre l'exécution d'une attaque [338]. oo7 [339] suit une approche similaire mais permet d'analyser le code binaire des applications.

Ces approches permettent de limiter l'impact sur les performance mais elles ne sont pas robustes car d'autres motifs de code peuvent permettre de réaliser l'attaque.

Guarnieri *et al.* adoptent une approche conservatrice [340]. Ils analysent statiquement le code de l'application et essaient de prouver que l'exécution spéculative respecte une propriété de non-interférence. Dans le cas contraire, le branchement doit être protégé. L'analyse réalise une surapproximation, ce qui peut conduire à protéger plus de branchements que nécessaire. En revanche, les branchements vulnérables sont systématiquement protégés.

Afin de lutter contre les autres variantes de Spectre, des techniques ont également été proposées afin d'empêcher l'empoisonnement des *buffer* utilisés par le processeur pour la spéculation.

Ainsi, Google a proposé Retpoline [341], une solution purement logicielle contre les attaques Spectre-BTB. Elle consiste à remplacer, dans le code des applications, les branchements indirects par des retours de fonction, en ayant pris soin de pousser l'adresse du saut sur la pile. Le comportement spéculatif sur les retours de fonction étant contrôlé par le *Return Stack Buffer*, l'attaquant ne peut plus contrôler le comportement spéculatif en empoisonnant le *Branch Target Buffer*. L'approche sature également le *Return Stack Buffer* afin de s'assurer que le comportement spéculatif est dirigé vers une boucle infinie.

Les concepteurs de microprocesseurs ont également proposé de nouvelles instructions permettant de contrôler la spéculation. Ainsi, sur l'architecture x86 IBRS (Indirect Branch Restricted Speculation) est un mode qui garantit que la spéculation sur les branchements ne peut être influencée par du code exécuté en dehors de ce mode. De même, STIBP (Single Thread Indirect Branch Prediction) permet de s'assurer que différents fils d'exécutions s'exécutant sur deux cœurs virtuels d'un même cœur logique ne partagent pas le *Branch Target Buffer*. L'instruction IBPB (Indirect Branch Predictor Barrier) permet de vider le *Branch Target Buffer*. Des instructions similaires sont disponibles sur l'architecture ARM. Ces différents mécanismes doivent être utilisés par le code de l'application ou de l'OS pour contrôler la spéculation.

Schwarz *et al.* proposent des modifications de la microarchitecture permettant de s'assurer que des données confidentielles ne peuvent être utilisées dans les exécutions transitoires [342]. Toutefois, cette approche nécessite des modifications dans la microarchitecture, l'OS, les compilateurs et les applications.

Empêcher l'accès aux données sensibles Une autre approche complémentaire consiste à empêcher le code exécuté de manière transitoire à accéder aux données sensibles.

Une première approche consiste à séparer le code des applications en différents composants et à isoler ces composants entre eux. En effet, dans les attaques de type Spectre, le code exécuté de manière spéculative respecte les frontières d'isolation.

Par exemple Google a isolé le traitement des différentes page Web consultées en parallèle dans les différents onglets de son navigateur Chrome [343]. L'isolation est réalisée en utilisant différents processus. Cette isolation permet également de limiter d'autres types d'attaques.

Une autre approche consiste à limiter le comportement du code exécuté spéculativement.

Ainsi, Koruyeh *et al.* proposent d'utiliser le CFI pour limiter les attaques de type Spectre-BTB en empêchant les sauts vers des *gadgets* [344].

Une approche pour lutter contre les variantes de Spectre-PHT dues à un contrôle des bornes d'un pointeur consiste à utiliser un masquage du pointeur [345]. Toutefois, ce masquage est en général imprécis mais il fixe une limite sur la valeur du pointeur. Cela limite donc fortement la zone de mémoire que l'attaquant peut faire fuir. Une autre approche consiste à mettre en œuvre le *Speculative Load Hardening* proposé par LLVM [346].

Supprimer ou limiter le canal caché Certaines techniques utilisées pour limiter les canaux auxiliaires peuvent être utilisées pour limiter ou supprimer le canal caché, qui est la dernière étape de l'attaque.

Une approche plus globale consiste à supprimer le canal caché pour l'exécution spéculative [347, 348]. Cette solution consiste à utiliser une mémoire interne pour réaliser l'exécution spéculative, sans accéder au cache. Cette solution n'est toutefois pas implémentée dans les processeurs disponibles sur le marché.

2.3.3 Contre-mesures logicielles face aux attaques par martellement de la mémoire

Des contre-mesures logicielles ont également été proposées pour lutter contre les attaques Rowhammer.

Les mesures proposées classiquement consistent à augmenter la vitesse de rafraîchissement de la DRAM et à empêcher l'attaquant d'exécuter l'instruction `CLFLUSH` permettant de vider le cache. Aweke *et al.* montrent que ces mesures ne sont pas suffisantes [349]. Ils réalisent en effet des attaques sans utiliser `CLFLUSH`, malgré le doublement de la fréquence de rafraîchissement. Ils proposent un mécanisme de défense qui surveille les accès mémoire fréquents à une adresse donnée et provoque un rafraîchissement des cellules mémoires adjacentes.

Brasser *et al.* s'intéressent plus particulièrement aux attaques visant le noyau de l'OS afin de réaliser une escalade de privilèges [350]. Ils proposent un mécanisme de défense spécifique à cette attaque qui confine l'attaquant de manière à ce qu'il ne puisse réaliser de changement d'état que dans l'espace mémoire qu'il contrôle déjà. Pour cela, les auteurs modifient le mécanisme d'allocation de la mémoire de l'OS afin de s'assurer que les pages de l'espace utilisateur correspondent à des adresses physiques qui ne permettent pas d'impacter les adresses physiques du noyau.

Konoth *et al.* généralisent ce principe. Ils intercallent des lignes de garde entre chaque ligne contenant des données dans la mémoire. Les lignes de garde absorbent les effets d'une attaque Rowhammer et évitent que les données, situées dans les lignes suivantes, soient affectées [351]. Van der Veen *et al.* utilisent la même approche pour protéger les *buffer* utilisés dans une attaque Rohammer utilisant le DMA sur l'architecture ARM [352].

RADAR est un projet qui permet de détecter toutes les formes d'attaques Rohammer en analysant les émissions électromagnétiques que produisent ces attaques [353]. Les auteurs

utilisent pour cela un dispositif externe de radio logicielle (*Software Defined Radio*) leur permettant de faire l'acquisition des signaux.

Chapitre 3

Perspectives

3.1 Défis et perspectives scientifiques

3.1.1 Fossé sémantique et isolation du moniteur de sécurité

L'utilisation d'un moniteur de sécurité pour surveiller l'exécution d'un composant logiciel (une application utilisateur, un OS ou un hyperviseur) est une approche intéressante. Dans une démarche de défense en profondeur, elle permet de s'assurer de l'intégrité du code et des données du composant lors de son exécution. Dans ce type d'approche, il convient d'isoler le moniteur pour le protéger des attaques visant le système surveillé. Toutefois, plus le système est isolé et moins il dispose facilement d'information sur le système surveillé. Cela est particulièrement le cas lorsque le moniteur s'exécute sur un processeur externe. L'isolation crée un fossé sémantique qui, à ce jour, reste un défi majeur pour ce type d'approches [47].

Les solutions les plus prometteuses consistent à combiner les approches qui s'appuient sur un modèle du composant surveillé, établi au préalable (par exemple lors de la compilation), et sur un support inséré au sein même du composant surveillé (par exemple en instrumentant le composant surveillé). Dans ce cas, le composant surveillé participe à sa propre surveillance. Se pose alors le problème de la confiance dans l'observation, l'attaquant pouvant exploiter une vulnérabilité dans le composant surveillé pour contourner la surveillance. Des mécanismes matériels pourraient être utilisés pour protéger la partie du moniteur intégrée aux composants surveillés. Une piste complémentaire consiste à utiliser des approches de *paraverification*, qui permettent au moniteur de s'assurer que les renseignements fournis par la partie intégrée aux composants surveillés sont cohérentes [44]. Ces approches, encore préliminaires, méritent d'être explorées plus en profondeur.

3.1.2 Support matériel pour la réponse aux intrusions

La majorité des travaux sur l'utilisation d'un moniteur externe pour surveiller un composant logiciel, notamment ceux implémentant le moniteur dans un hyperviseur afin de surveiller le système exécuté dans les machines virtuelles (VMI), se concentre sur la détection des intrusions. Peu de travaux se sont intéressés à tirer parti d'un moniteur isolé matériellement pour implémenter des contre-mesures, en modifiant l'état du système surveillé [44].

Les approches permettant de maintenir un certain niveau de disponibilité du composant surveillé, en mettant en place un mode de fonctionnement dégradé, tout en garantissant la résilience du composant face aux attaques, sont particulièrement intéressantes. De telles approches ont jusqu'à présent essentiellement été étudiées à l'échelle d'un réseau informatique et s'appuient généralement sur des contre-mesures à gros grains (par exemple filtrer l'intégralité du trafic à destination de la machine).

L'utilisation d'un support matériel pour mettre en œuvre des contre-mesures à grain fin (par exemple, modifier l'état d'une application) et isoler le mécanisme de réaction constitue une perspective intéressante, tant du point de vue scientifique que des applications industrielles. Les constructeurs d'ordinateurs cherchent en effet de plus en plus à doter les ordinateurs de capacités de réaction automatique aux intrusions, et ce de manière autonome (sans nécessiter l'utilisation de composants externes de gestion et d'administration de la sécurité). Cela permet de garantir la résilience de la machine, y compris lorsqu'elle n'est plus intégrée dans un parc informatique faisant l'objet d'une administration et d'une supervision centralisées. Cela correspond notamment à des scénarios d'itinérance et de télétravail, qui sont particulièrement d'actualité.

3.1.3 Contrat matériel/logiciel pour lutter contre les canaux auxiliaires

Les attaques logicielles par canaux auxiliaires exploitent, de manière générale, des vulnérabilités liées au choix de conception de la microarchitecture. Ces vulnérabilités conduisent à des différences de temps d'exécution qui sont mesurables par un attaquant pouvant exécuter du code sur la machine cible. Ces variations sont en grande partie dues à des optimisations réalisées par le matériel à l'exécution afin d'optimiser le débit d'instructions exécutées. Il n'est donc pas envisageable de les supprimer totalement, tout du moins dans le cas général. Des contre-mesures matérielles ont été proposées dans la littérature pour pallier aux attaques contre la microarchitecture. Toutefois, la plupart d'entre elles réduisent de manière importante les performances à l'exécution.

Afin de préserver les performances, il est nécessaire que le processeur adopte un comportement conservateur seulement lorsqu'il manipule des données confidentielles. Pour cela, il doit impérativement s'appuyer sur le logiciel, qui doit notamment lui indiquer quelles sont les données à protéger. De manière générale, il est nécessaire de spécifier et d'implémenter un contrat entre le matériel et le logiciel pour garantir une isolation forte [295]. Le matériel doit proposer des contre-mesures paramétrables par le logiciel tout en spécifiant explicitement le niveau de sécurité garanti par ces mécanismes. Le logiciel doit pouvoir tirer parti de ces mécanismes pour protéger les données confidentielles.

Il paraît assez naturel d'implémenter un tel contrat au niveau du jeu d'instructions, qui constitue l'interface entre le logiciel et le matériel. Toutefois, l'ISA est une API (*Application Programming Interface*, interface de programmation) qui a pour objectif de masquer les détails d'implémentation de la microarchitecture. Cela permet de garantir que le logiciel développé sera compatible avec différentes implémentations d'une même architecture. Cependant, les contre-mesures matérielles permettant de lutter contre les canaux auxiliaires dépendent de la microarchitecture. Étendre le jeu d'instructions pour lutter contre les canaux auxiliaires, tout en garantissant l'indépendance de l'architecture vis-à-vis des choix d'implémentation, constitue un défi important.

L'ISA des processeurs actuels ne permet par d'implémenter un tel contrat [295], même si les constructeurs étendent de plus en plus les jeux d'instructions de leurs processeurs pour exposer les mécanismes de sécurité au niveau de l'ISA. Des travaux récents se sont intéressés à étendre les jeux d'instructions, notamment celui des processeurs RISC-V, afin d'implémenter un tel contrat [159]. Il s'agit encore de travaux préliminaires. En outre, il convient également de s'intéresser à l'utilisation d'un tel jeu d'instruction étendu par les chaînes de compilation et les composants logiciels privilégiés (OS, hyperviseurs, gestionnaires d'enclaves, etc.).

3.1.4 Processeur générique résistant aux attaques par observation de la consommation de courant

Aujourd'hui, la résistance aux attaques physiques est l'apanage de petits coprocesseurs dédiés à des tâches critiques, surtout pour des applications mettant en œuvre de la cryptographie. Ainsi un processeur, tel que ceux présents dans un téléphone portable, n'offre pas de protection vis à vis de ces attaques pour les cœurs dit "génériques" (excluant les coprocesseurs). Cela ne les empêche toutefois pas de manipuler des données confidentielles, par conséquent à la merci de telles attaques. Les stratégies classiques pour protéger des attaques par observation de la consommation de courant (masquage, génération de bruit analogique, logique différentielle, ...) ne marchent plus : elles nécessitent des chemins de données simples et maîtrisés. Le problème de concevoir un processeur capable de se prémunir d'un attaquant observant sa consommation de courant reste donc ouvert.

3.1.5 Automatisation de l'analyse des attaques logicielles contre la microarchitecture

A l'heure actuelle, la recherche d'attaques logicielles exploitant des vulnérabilités de la microarchitecture nécessite une analyse approfondie des mécanismes matériels susceptibles d'être vulnérables. Cette analyse est en grande partie manuelle et s'appuie généralement sur un travail de rétroconception, les détails d'implémentation de la microarchitecture n'étant pas rendus publics par les constructeurs. Ce travail long et fastidieux doit être itéré pour chaque microarchitecture. En outre, une analyse doit également être réalisée au niveau du logiciel exécuté sur cette microarchitecture afin de déterminer si ces vulnérabilités matérielles pourront être exploitées. Là aussi, cela nécessite une analyse complexe, notamment dans le cadre des attaques exploitant l'exécution spéculative.

L'automatisation de l'analyse des attaques par canaux auxiliaires constitue encore aujourd'hui un problème ouvert. Il s'agit notamment d'évaluer si des vulnérabilités matérielles sont exploitables, étant donné le code d'une application. Une approche en boite noire, qui ne s'appuie pas sur un modèle explicite de la microarchitecture, mais repose sur des techniques issues de l'intelligence artificielle, semble une piste prometteuse [189].

Cette automatisation offre non seulement des perspectives scientifiques intéressantes, mais elle correspond également à un besoin industriel important. En effet, dans le cadre d'évaluations de sécurité réalisées en temps contraint (notamment les évaluations suivant le schéma français CSPN), l'expert n'a pas les ressources nécessaires pour réaliser une analyse approfondie de la microarchitecture.

3.1.6 Modélisation et preuve formelles des mécanismes de sécurité matériels

Afin de renforcer la confiance dans les plateformes matérielles, il paraît important de proposer des approches permettant de développer des composants matériels tels que les processeurs et permettant de s'assurer :

1. que ces composants se comportent selon une spécification clairement établie ;
2. que ces composants permettent de garantir des propriétés de sécurité clairement spécifiées.

Ce dernier point nécessite d'implémenter des mécanismes de sécurité au sein de ces composants et de vérifier que les implémentations permettent effectivement d'assurer les propriétés de sécurité attendues. Cette tâche est d'autant plus complexe que certains mécanismes de sécurité nécessitent une coopération entre le matériel, qui fournit des primitives de sécurité, et le logiciel qui doit les utiliser correctement pour assurer la propriété de sécurité attendue. Il est donc nécessaire de spécifier et vérifier formellement le matériel, le logiciel et leurs interactions.

Les spécifications des ISA des processeurs actuels sont complexes et souvent rédigées de manière informelle. Toutefois, des travaux récents se sont intéressés à la modélisation formelle des interfaces logiciel/matériel, notamment dans le cadre du projet SAIL de l'université de Cambridge. Cette formalisation a été adoptée par la fondation RISC-V. ARM publie également une spécification formelle de l'ISA de ces processeurs.

Cette formalisation de l'ISA n'est cependant qu'une première étape. Il convient également de formaliser des propriétés de sécurité au niveau de ces interfaces, et de vérifier les composants impliqués de part et d'autre de ces interfaces.

Plusieurs initiatives se sont intéressées récemment à définir des spécifications formelles de l'architecture des microprocesseurs [178, 179], afin notamment de pouvoir vérifier formellement certaines propriétés. Cette spécification s'exprime à l'aide d'un HDL de haut niveau. Un des défis consiste à s'assurer que l'implémentation matérielle, qu'il s'agisse d'un circuit ASIC (*Application-Specific Integrated Circuit*, circuit intégré spécialisé) ou d'un soft-core implémenté sur FPGA, respecte cette spécification et que les propriétés vérifiées sur le HDL de haut niveau sont préservées lors de la génération du *bitstream* ou du masque. Ces travaux sont prometteurs, mais les implémentations correspondent pour l'instant à des microarchitectures simples. En outre, ces travaux s'intéressent pour l'instant à la spécification fonctionnelle de l'ISA, qui, comme souligné en section 3.1.3, ne permet pas de raisonner sur les canaux auxiliaires. La prise en compte de ces canaux auxiliaires dans les modèles formels et le passage à l'échelle constituent des perspectives importantes.

3.1.7 Micro-isolation

Les microprocesseurs implémentent de plus en plus de mécanismes permettant d'isoler des composants logiciels qui partagent les ressources matérielles de la plateforme (CPU, mémoire, caches, périphériques, etc.). Quels que soient les mécanismes d'isolation, force est de constater qu'en pratique, les composants isolés (OS, hyperviseur, processus, etc.) représentent souvent une taille et une complexité importantes en ce qui concerne la quantité de code et de fonctionnalités implémentées. Ils offrent donc une surface d'attaque

importante, ce qui est particulièrement critique lorsqu'il s'agit de composants privilégiés comme l'OS, l'hyperviseur ou l'environnement d'exécution au sein d'une enclave. De fait, de nombreuses vulnérabilités sont régulièrement découvertes dans ces composants.

Afin de limiter les conséquences de l'exploitation d'une vulnérabilité dans un composant logiciel, une approche prometteuse consiste à étendre le principe de cloisonnement au maximum via une approche de microcloisonnement. Cela permet d'appliquer le principe, bien connu en sécurité informatique, de séparation des tâches et, *in fine*, de rendre plus efficace l'application du principe de moindre privilège. Des travaux se sont déjà intéressés à appliquer cette approche au niveau de l'OS (micronoyau) et de l'hyperviseur [41]. Il convient de les généraliser à tous les composants logiciels, notamment les applications utilisateurs et les environnements d'exécutions des enclaves. Cela nécessite un support matériel (par exemple Intel MPK [143]). Un des enjeux consiste à minimiser le coût à l'exécution lié à la mise en place du cloisonnement et aux changements de contexte. En outre, une attention particulière doit être attachée aux canaux auxiliaires pour que ces mécanismes puissent permettre d'assurer une isolation forte. Cette approche présente à la fois des perspectives scientifiques et industrielles intéressantes au niveau du matériel, afin de proposer de nouveaux mécanismes de cloisonnement, et au niveau logiciel, en développant des composants logiciels qui peuvent tirer parti de ces nouveaux mécanismes d'isolation à grain fin.

3.1.8 Identification de nouvelles attaques contre la microarchitecture

De nombreuses attaques contre la microarchitecture des processeurs disponibles sur le marché ont été découvertes ces dernières années, en particulier celles exploitant l'exécution spéculative. Toutefois, en raison de la complexité de la microarchitecture des processeurs modernes, et du fait que les détails d'implémentation ne sont pas toujours rendus publics, il est fort probable que d'autres vulnérabilités existent, dans les architectures existantes et à venir, même si les concepteurs de microprocesseurs sont aujourd'hui pleinement conscients de cette menace. Des travaux récents [217, 218] se sont intéressés à classifier et systématiser les approches mises en œuvre pour réaliser des attaques contre la spéculation. Ces études mettent en évidence que différents types d'attaques, qui n'ont pas encore été démontrées, sont possibles.

Si la plupart des attaques contre la microarchitecture se sont focalisées sur l'exploitation de canaux auxiliaires, notamment ceux liés à la gestion des caches, d'autres attaques sont à redouter. En particulier, les attaques logicielles qui permettent d'injecter des fautes, par exemple en contrôlant la fréquence du processeur [235] ou son alimentation électrique [236, 238, 239], constituent une menace sérieuse. Ces attaques ont, pour l'instant, fait l'objet de moins de travaux. L'étude de telles attaques constitue donc une perspective de recherche intéressante.

3.1.9 Environnement d'exécution réellement de confiance

Les processeurs modernes embarquent de plus en plus des TEE ou des enclaves, qui sont des environnements d'exécution isolés au sein du processeur et qui permettent en théorie de garantir la confidentialité et l'intégrité des données hébergées et manipulées dans ces environnements. Toutefois, contrairement aux approches qui utilisent des SE

(*Secure Element*, élément sécurisé), qui sont des composants externes, les TEE partagent des ressources (CPU, cache, mémoire, périphériques) avec l'environnement d'exécution classique (typiquement, le système d'exploitation et les applications). Ils n'offrent pas le même niveau d'isolation que les SE mais permettent de tirer partie des capacités de la plateforme matérielle, qui sont en général bien plus importantes que celles des SE. Cela permet notamment de déployer des applications nécessitant des ressources matérielles plus conséquentes.

Toutefois, la détermination des garanties de sécurité réellement assurées par ces environnements reste un problème ouvert. Les travaux récents ont confirmé que la plupart des implémentations matérielles ne résistent pas aux attaques par canaux auxiliaires. Bien que ce type d'attaques ne soit généralement pas considéré dans le modèle d'attaques des concepteurs, il limite l'intérêt de ces environnements pour assurer la confidentialité des données hébergées. Cette limitation constitue une perspective de recherche importante, tant au niveau des contre-mesures logicielles permettant de protéger le code exécuté sur les environnements existants que des contre-mesures matérielles qui pourraient être déployées sur de futures architectures.

En outre, afin de faciliter le déploiement et l'intégration d'applications sécurisées au sein des TEE, des environnements d'exécutions logiciels ont été proposés, dans le monde académique et industriel [77-86, 88]. Ces environnements permettent de faciliter les échanges et les changements de contexte entre le mode d'exécution de l'application et le mode d'exécution sécurisé, au sein de l'enclave. La sécurisation de ces échanges est particulièrement cruciale car l'OS n'est pas nécessairement de confiance. Ces environnements fournissent une API entre les deux mondes et ils sont chargés de sécuriser les échanges, notamment en filtrant ou en chiffrant les données échangées lors des changements de contexte. Toutefois, des études ont montré que les environnements existants ne sont pas suffisamment robustes et que des vulnérabilités existent au niveau du filtrage qui devrait être réalisé au niveau de l'API [254]. La sécurisation des interfaces et l'identification de vulnérabilités au niveau du filtrage d'interface est donc une perspective de recherche importante.

3.2 Perspectives d'application et transfert de technologie

3.2.1 Architecture CHERI

CHERI (*Capability Hardware Enhanced RISC Instructions*) est un projet de recherche issu d'une collaboration entre différents laboratoires de l'université de Cambridge et SRI International[1]. Cette approche consiste à étendre le jeu d'instructions du processeur afin de gérer des capacités. Il s'agit d'un nouveau type de données permettant d'associer des permissions à des pointeurs. Cette approche permet de se prémunir contre un large spectre d'attaques logicielles exploitant les vulnérabilités liées à la gestion de la mémoire. Outre la définition de l'ISA, les chercheurs ont également proposé des microarchitectures implémentant cette approche (BERI, RISC-V). Ils ont également adapté un OS (CheriBSD) et un compilateur (Clang/LLVM) afin de supporter cette architecture et utiliser les capacités pour implémenter des mécanismes de protection de la mémoire. Le développement de ces différents prototypes leur ont permis de démontrer la viabilité de leur approche.

1. https://www.cl.cam.ac.uk/research/security/ctsrd/cheri/

Ces travaux ont intéressé ARM [354] et Microsoft [355] qui, avec le soutient financier du gouvernement britannique dans le cadre du *UK Industrial Strategy Challenge Fund (ISCF)* et de la DARPA, vont travailler avec les chercheurs de l'université afin de développer un démonstrateur comprenant une plateforme matérielle ARM et une adaptation des OS Linux et Android [356] ainsi que de l'hyperviseur Hafnium [357] développé par Google. ARM a initié le projet Morello [354], dont l'objectif est d'étendre le jeu d'instructions ARM arm64-v8a pour supporter le système de capacités de CHERI ainsi que développer un prototype de processeur implémentant cette architecture. La sortie du prototype matériel est annoncée pour début 2022 [358]. A ce jour, la spécification de l'ISA a été rendue publique. ARM travaille également en collaboration avec Linaro afin de proposer un support à la communauté de développeurs open-source [358]. L'objectif est que des contributeurs externes puissent réaliser des développements logiciels compatibles avec cet écosystème. Même si ARM annonce clairement ne pas s'engager à adopter CHERI dans ces produits, il s'agit d'une des initiatives de transfert technologique les plus marquantes du domaine.

3.2.2 Introspection et vérification d'intégrité à l'exécution

Les acteurs industriels ont depuis plusieurs années déployé des mécanismes leur permettant de s'assurer de l'intégrité des composants logiciels (*firmware*, hyperviseur, OS) lors du démarrage de la machine. Ces approches reposent notamment sur des composants matériels tels que le TPM, qui permettent de mesurer ou de vérifier l'intégrité des composants logiciels exécutés successivement depuis la phase de démarrage. Toutefois cette approche ne permet de s'assurer de l'intégrité qu'au démarrage et n'empêche pas les atteintes à l'intégrité à l'issue de cette étape. Comme les redémarrages tendent à s'espacer, notamment en raison de la gestion d'énergie et des mécanismes de mise en veille, il existe une fenêtre d'attaque importante qui peut être exploitée par un attaquant pour compromettre l'intégrité de la plateforme.

Pour cette raison, les industriels s'intéressent de près aux approches matérielles permettant de vérifier l'intégrité de la plateforme durant la phase d'exécution. Par exemple, les laboratoires de recherche de HP Inc. ont collaboré avec l'équipe CIDRE afin de vérifier l'intégrité du code privilégié du *firmware* UEFI qui s'exécute en SMM, à l'aide d'un moniteur de sécurité isolé matériellement [121]. Ces travaux ont fait l'objet de demandes de brevets. Samsung implémente également dans sa solution Knox [61], qui s'exécute de manière isolée dans le TEE TrustZone, des mécanismes de vérifications d'intégrité du système Android (RKP et TIMA) [62]. Ces solutions sont issues des travaux réalisés par des chercheurs de l'université d'État de Caroline du Nord (NC State) [64], qui ont fait l'objet d'un transfert technologique à Samsung [63]. Enfin Microsoft a développé des approches similaires dans sa technologie *Virtualization Based Security* [33] qui utilise son hyperviseur Hyper-V pour isoler des fonctions de sécurité. La fonctionnalité *Hypervisor-Protected Code Integrity (HVCI)* [2] permet notamment de vérifier l'intégrité du code du noyau de l'OS à l'exécution.

2. https://docs.microsoft.com/en-us/windows-hardware/drivers/bringup/device-guard-and-credential-guard

Bibliographie

[1] Lianying Zhao, He Shuang, Shengjie Xu, Wei Huang, Rongzhen Cui, Pushkar Bettadpur et David Lie. *SoK : Hardware Security Support for Trustworthy Execution*. 2019. arXiv : 1910.04957 [cs.CR] (pages 8, 9).

[2] G. Dessouky, T. Frassetto, P. Jauernig, A. -R. Sadeghi et E. Stapf. « With Great Complexity Comes Great Vulnerability : From Stand-Alone Fixes to Reconfigurable Security ». In : *IEEE Security Privacy* 18.5 (2020), p. 57-66. doi : 10.1109/MSEC.2020.2994978 (page 8).

[3] G. E. Suh, C. W. O'Donnell et S. Devadas. « Aegis : A Single-Chip Secure Processor ». In : *IEEE Design Test of Computers* 24.6 (2007), p. 570-580. doi : 10.1109/MDT.2007.179 (page 8).

[4] Fengwei Zhang et Hongwei Zhang. « SoK : A Study of Using Hardware-Assisted Isolated Execution Environments for Security ». In : *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016*. HASP 2016. Seoul, Republic of Korea : Association for Computing Machinery, 2016. isbn : 9781450347693. doi : 10.1145/2948618.2948621. url : https://doi.org/10.1145/2948618.2948621 (page 9).

[5] P. Maene, J. Götzfried, R. de Clercq, T. Müller, F. Freiling et I. Verbauwhede. « Hardware-Based Trusted Computing Architectures for Isolation and Attestation ». In : *IEEE Transactions on Computers* 67.3 (2018), p. 361-374. doi : 10.1109/TC.2017.2647955 (page 9).

[6] Will Arthur et Kenneth Goldman. *A Practical Guide to TPM 2.0*. Jan. 2015. doi : 10.1007/978-1-4302-6584-9 (page 10).

[7] *How Windows 10 uses the Trusted Platform Module*. Microsoft. 2017. url : https://docs.microsoft.com/en-us/windows/security/information-protection/tpm/how-windows-uses-the-tpm (page 10).

[8] Reiner Sailer, Xiaolan Zhang, Trent Jaeger et Leendert van Doorn. « Design and Implementation of a TCG-Based Integrity Measurement Architecture ». In : *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*. SSYM'04. San Diego, CA : USENIX Association, 2004, p. 16 (page 10).

[9] Jake Edge. *Toward measured boot out of the box*. LWM.net. 2016. url : https://lwn.net/Articles/699551/ (page 10).

[10] Mimi Zohar. *LSS-EU 2018 : Overview and Recent DevelopmentsLinux Integrity Subsystem*. IBM, 2018. url : https://events19.linuxfoundation.org/wp-content/uploads/2017/12/LSS2018-EU-LinuxIntegrityOverview_Mimi-Zohar.pdf (page 10).

[11] Stefan Berger, Ramón Cáceres, Kenneth A. Goldman, Ronald Perez, Reiner Sailer et Leendert van Doorn. « VTPM : Virtualizing the Trusted Platform Module ». In : *Proceedings of the 15th Conference on USENIX Security Symposium*

- *Volume 15*. USENIX-SS'06. Vancouver, B.C., Canada : USENIX Association, 2006 (page 10).

[12] *Virtual Trusted Platform Module*. Google. URL : https://cloud.google.com/security/shielded-cloud/shielded-vm#vtpm (page 10).

[13] David EDMONDSON. *Introduction to Late Launch*. URL : https://github.com/TrenchBoot/documentation/blob/master/documentation/Late_Launch_Overview.md (page 10).

[14] *Intel Trusted Execution Technology (Intel TXT) - Software Development Guide*. Version 315168-16.3. Intel Corporation. 2020. URL : https://www.intel.com/content/dam/www/public/us/en/documents/guides/intel-txt-software-development-guide.pdf (page 10).

[15] *Trusted Boot*. URL : https://sourceforge.net/projects/tboot/ (page 11).

[16] *Design and usage of OpenDTeX DRTM Secure Boot*. AMOSSYS. 2015. URL : https://blog.amossys.fr/Design%20and%20usage%20of%20OpenDTeX%20DRTM%20Secure%20Boot.html (page 11).

[17] Jonathan M. MCCUNE, Bryan J. PARNO, Adrian PERRIG, Michael K. REITER et Hiroshi ISOZAKI. « Flicker : An Execution Infrastructure for TCB Minimization ». In : Eurosys '08. Glasgow, Scotland UK : Association for Computing Machinery, 2008, p. 315-328. ISBN : 9781605580135. DOI : 10.1145/1352592.1352625. URL : https://doi.org/10.1145/1352592.1352625 (pages 11, 15, 16, 18).

[18] Lianying ZHAO et Mohammad MANNAN. « TEE-aided Write Protection Against Privileged Data Tampering ». In : *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society, 2019. URL : https://www.ndss-symposium.org/ndss-paper/tee-aided-write-protection-against-privileged-data-tampering/ (page 11).

[19] Rafal WOJTCZUK et Joanna RUTKOWSKA. *Attacking Intel® Trusted Execution Technology*. 2009. URL : https://invisiblethingslab.com/resources/bh09dc/Attacking%20Intel%20TXT%20-%20paper.pdf (page 11).

[20] Rafal WOJTCZUK, Joanna RUTKOWSKA et Alexander TERESHKIN. *Another Way to Circumvent Intel® Trusted Execution Technolog*. 2009. URL : https://invisiblethingslab.com/resources/misc09/Another%20TXT%20Attack.pdf (page 11).

[21] Rafal WOJTCZUK et Joanna RUTKOWSKA. *Attacking Intel TXT® via SINIT code execution hijacking*. 2011. URL : https://invisiblethingslab.com/resources/2011/Attacking_Intel_TXT_via_SINIT_hijacking.pdf (page 11).

[22] *CVE-2019-0184*. MITRE. 2019. URL : https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-0184 (page 11).

[23] *CVE-2019-0151*. MITRE. 2019. URL : https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-0151 (page 11).

[24] *CVE-2019-0152*. MITRE. 2019. URL : https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-0152 (page 11).

[25] Seunghun HAN, Wook SHIN, Jun-Hyeok PARK et HyoungChun KIM. « A Bad Dream : Subverting Trusted Platform Module While You Are Sleeping ». In : *Proceedings of the 27th USENIX Conference on Security Symposium*. SEC'18. Baltimore, MD, USA : USENIX Association, 2018, p. 1229-1246. ISBN : 9781931971461 (page 11).

[26] Daniel MOGHIMI, Berk SUNAR, Thomas EISENBARTH et Nadia HENINGER. « TPM-FAIL : TPM meets Timing and Lattice Attacks ». In : *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, août 2020, p. 2057-2073. ISBN : 978-1-939133-17-5. URL : https://www.usenix.org/conference/usenixsecurity20/presentation/moghimi-tpm (page 11).

[27] Alessandro REINA, Aristide FATTORI, Fabio PAGANI, Lorenzo CAVALLARO et Danilo BRUSCHI. « When Hardware Meets Software : A Bulletproof Solution to Forensic Memory Acquisition ». In : *Proceedings of the 28th Annual Computer Security Applications Conference*. ACSAC '12. Orlando, Florida, USA : Association for Computing Machinery, 2012, p. 79-88. ISBN : 9781450313124. DOI : 10.1145/2420950.2420962. URL : https://doi.org/10.1145/2420950.2420962 (page 12).

[28] Kun SUN, Jiang WANG, Fengwei ZHANG et Angelos STAVROU. « SecureSwitch : BIOS-Assisted Isolation and Switch between Trusted and Untrusted Commodity OSes ». In : *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*. The Internet Society, 2012. URL : https://www.ndss-symposium.org/ndss2012/secureswitch-bios-assisted-isolation-and-switch-between-trusted-and-untrusted-commodity-oses (page 12).

[29] F. ZHANG, K. LEACH, K. SUN et A. STAVROU. « SPECTRE : A dependable introspection framework via System Management Mode ». In : *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 2013, p. 1-12. DOI : 10.1109/DSN.2013.6575343 (page 12).

[30] B. DELGADO, T. VIBHUTE, J. FASTABEND et K. KARAVANIC. « EPA-RIMM : An Efficient, Performance-Aware Runtime Integrity Measurement Mechanism for Modern Server Platforms ». In : *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 2019, p. 422-434. DOI : 10.1109/DSN.2019.00051 (page 12).

[31] Fengwei ZHANG, Haining WANG, Kevin LEACH et Angelos STAVROU. « A Framework to Secure Peripherals at Runtime ». In : *Computer Security - ESORICS 2014*. Sous la dir. de Mirosław KUTYŁOWSKI et Jaideep VAIDYA. Cham : Springer International Publishing, 2014, p. 219-238. ISBN : 978-3-319-11203-9 (page 12).

[32] F. ZHANG, K. LEACH, A. STAVROU, H. WANG et K. SUN. « Using Hardware Features for Increased Debugging Transparency ». In : *2015 IEEE Symposium on Security and Privacy*. 2015, p. 55-69. DOI : 10.1109/SP.2015.11 (page 12).

[33] *Virtualization-based Security (VBS)*. Microsoft. 2017. URL : https://docs.microsoft.com/en-us/windows-hardware/design/device-experiences/oem-vbs (pages 13, 68).

[34] Ronald PEREZ, Leendert van DOORN et Reiner SAILER. « Virtualization and Hardware-Based Security ». In : *IEEE Security and Privacy* 6.5 (sept. 2008), p. 24-31. ISSN : 1540-7993. DOI : 10.1109/MSP.2008.135. URL : https://doi.org/10.1109/MSP.2008.135 (page 14).

[35] Tamas K. LENGYEL, Steve MARESCA, Bryan D. PAYNE, George D. WEBSTER, Sebastian VOGL et Aggelos KIAYIAS. « Scalability, Fidelity and Stealth in the DRAKVUF Dynamic Malware Analysis System ». In : *Proceedings of the 30th Annual Computer Security Applications Conference*. ACSAC '14. New Orleans, Louisiana, USA : Association for Computing Machinery, 2014, p. 386-395. ISBN : 9781450330053. DOI : 10.1145/2664243.2664252. URL : https://doi.org/10.1145/2664243.2664252 (pages 14, 15).

[36] *AMD Secure Encrypted Virtualization (SEV)*. URL : https://developer.amd.com/sev/ (page 14).

[37] *Intel architecture : Memory encryption technologies specification*. Intel Corporation. 2019. URL : https://software.intel.com/content/dam/develop/external/us/en/documents/multi-key-total-memory-encryption-spec-753926.pdf (page 14).

[38] Mengyuan LI, Yinqian ZHANG, Zhiqiang LIN et Yan SOLIHIN. « Exploiting Unprotected I/O Operations in AMD's Secure Encrypted Virtualization ». In : *Proceedings of the 28th USENIX Conference on Security Symposium*. SEC'19. Santa Clara, CA, USA : USENIX Association, 2019, p. 1257-1272. ISBN : 9781939133069 (page 14).

[39] Michal MOSKAL, Thomas SANTEN et Wolfram SCHULTE. « VCC : A Practical System for Verifying Concurrent C ». In : *Theorem Proving in Higher Order Logics, 22nd International Conference, TPHOLs 2009*. T. 5674. Lecture Notes in Computer Science. Jan. 2009, p. 23-42. ISBN : 978-3-642-03358-2. URL : https://www.microsoft.com/en-us/research/publication/vcc-a-practical-system-for-verifying-concurrent-c/ (page 15).

[40] Chiachih WU, Zhi WANG et Xuxian JIANG. « Taming Hosted Hypervisors with (Mostly) Deprivileged Execution ». In : *20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24-27, 2013*. The Internet Society, 2013. URL : https://www.ndss-symposium.org/ndss2013/taming-hosted-hypervisors-mostly-deprivileged-execution (page 15).

[41] Zeyu MI, Dingji LI, Haibo CHEN, Binyu ZANG et Haibing GUAN. « (Mostly) Exitless VM Protection from Untrusted Hypervisor through Disaggregated Nested Virtualization ». In : *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, août 2020, p. 1695-1712. ISBN : 978-1-939133-17-5. URL : https://www.usenix.org/conference/usenixsecurity20/presentation/mi (pages 15, 66).

[42] J. M. MCCUNE, Y. LI, N. QU, Z. ZHOU, A. DATTA, V. GLIGOR et A. PERRIG. « TrustVisor : Efficient TCB Reduction and Attestation ». In : *2010 IEEE Symposium on Security and Privacy*. 2010, p. 143-158. DOI : 10.1109/SP.2010.17 (page 15).

[43] Abhinav SRIVASTAVA et Jonathon T. GIFFIN. « Efficient Monitoring of Untrusted Kernel-Mode Execution ». In : *Proceedings of the Network and Distributed System Security Symposium, NDSS 2011, San Diego, California, USA, 6th February - 9th February 2011*. The Internet Society, 2011. URL : https://www.ndss-symposium.org/ndss2011/efficient-monitoring-untrusted-kernel-mode-execution (page 15).

[44] Erick BAUMAN, Gbadebo AYOADE et Zhiqiang LIN. « A Survey on Hypervisor-Based Monitoring : Approaches, Applications, and Evolutions ». In : *ACM Comput. Surv.* 48.1 (août 2015). ISSN : 0360-0300. DOI : 10.1145/2775111. URL : https://doi.org/10.1145/2775111 (pages 15, 62).

[45] Sergej PROSKURIN, Tamas LENGYEL, Marius MOMEU, Claudia ECKERT et Apostolis ZARRAS. « Hiding in the Shadows : Empowering ARM for Stealthy Virtual Machine Introspection ». In : *Proceedings of the 34th Annual Computer Security Applications Conference*. ACSAC '18. San Juan, PR, USA : Association for Compu-

ting Machinery, 2018, p. 407-417. ISBN : 9781450365697. DOI : 10.1145/3274694.3274698. URL : https://doi.org/10.1145/3274694.3274698 (page 15).

[46] Bin SHI, Lei CUI, Bo LI, Xudong LIU, Zhiyu HAO et Haiying SHEN. « ShadowMonitor : An Effective In-VM Monitoring Framework with Hardware-Enforced Isolation ». In : *Research in Attacks, Intrusions, and Defenses*. Sous la dir. de Michael BAILEY, Thorsten HOLZ, Manolis STAMATOGIANNAKIS et Sotiris IOANNIDIS. Cham : Springer International Publishing, 2018, p. 670-690. ISBN : 978-3-030-00470-5 (pages 15, 16).

[47] B. JAIN, M. B. BAIG, D. ZHANG, D. E. PORTER et R. SION. « SoK : Introspections on Trust and the Semantic Gap ». In : *2014 IEEE Symposium on Security and Privacy*. 2014, p. 605-620. DOI : 10.1109/SP.2014.45 (pages 15, 62).

[48] Martim CARBONE, Matthew CONOVER, Bruce MONTAGUE et Wenke LEE. « Secure and Robust Monitoring of Virtual Machines through Guest-Assisted Introspection ». In : *Proceedings of the 15th International Conference on Research in Attacks, Intrusions, and Defenses*. RAID'12. Amsterdam, The Netherlands : Springer-Verlag, 2012, p. 22-41. ISBN : 9783642333378. DOI : 10.1007/978-3-642-33338-5_2. URL : https://doi.org/10.1007/978-3-642-33338-5_2 (page 15).

[49] Quan CHEN, Ahmed M. AZAB, Guruprasad GANESH et Peng NING. « PrivWatcher : Non-Bypassable Monitoring and Protection of Process Credentials from Memory Corruption Attacks ». In : *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. ASIA CCS '17. Abu Dhabi, United Arab Emirates : Association for Computing Machinery, 2017, p. 167-178. ISBN : 9781450349444. DOI : 10.1145/3052973.3053029. URL : https://doi.org/10.1145/3052973.3053029 (page 16).

[50] Maxwell RENKE et Chris RIGGS. *Virtualization-based security (VBS) memory enclaves : Data protection through isolation*. Microsoft. 2018. URL : https://www.microsoft.com/security/blog/2018/06/05/virtualization-based-security-vbs-memory-enclaves-data-protection-through-isolation/ (page 16).

[51] Emmanuel OWUSU, Jorge GUAJARDO, Jonathan MCCUNE, Jim NEWSOME, Adrian PERRIG et Amit VASUDEVAN. « OASIS : On Achieving a Sanctuary for Integrity and Secrecy on Untrusted Platforms ». In : CCS '13. Berlin, Germany : Association for Computing Machinery, 2013, p. 13-24. ISBN : 9781450324779. DOI : 10.1145/2508859.2516678. URL : https://doi.org/10.1145/2508859.2516678 (page 16).

[52] Job NOORMAN, Pieter AGTEN, Wilfried DANIELS, Raoul STRACKX, Anthony Van HERREWEGE, Christophe HUYGENS, Bart PRENEEL, Ingrid VERBAUWHEDE et Frank PIESSENS. « Sancus : Low-cost Trustworthy Extensible Networked Devices with a Zero-software Trusted Computing Base ». In : *22nd USENIX Security Symposium (USENIX Security 13)*. Washington, D.C. : USENIX Association, août 2013, p. 479-498. ISBN : 978-1-931971-03-4. URL : https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/noorman (pages 16, 46, 48).

[53] *TrustZone – Arm Developer*. ARM Limited. URL : https://developer.arm.com/ip-products/security-ip/trustzone (page 16).

[54] *GlobalPlatform Trusted Execution Environment (TEE) Committee*. GlobalPlatform. URL : https://globalplatform.org/technical-committees/trusted-execution-environment-tee-committee/ (page 16).

[55] M. Sabt, M. Achemlal et A. Bouabdallah. « Trusted Execution Environment : What It is, and What It is Not ». In : *2015 IEEE Trustcom/BigDataSE/ISPA*. T. 1. 2015, p. 57-64. doi : 10.1109/Trustcom.2015.357 (page 17).

[56] Thomas Nyman, Jan-Erik Ekberg, Lucas Davi et N. Asokan. « CFI CaRE : Hardware-Supported Call and Return Enforcement for Commercial Microcontrollers ». In : *Research in Attacks, Intrusions, and Defenses*. Sous la dir. de Marc Dacier, Michael Bailey, Michalis Polychronakis et Manos Antonakakis. Cham : Springer International Publishing, 2017, p. 259-284. isbn : 978-3-319-66332-6 (page 17).

[57] *Introduction to Trusted Execution Environment : ARM's TrustZone*. url : https://blog.quarkslab.com/introduction-to-trusted-execution-environment-arms-trustzone.html (page 17).

[58] Maria Markstedter. *Trusted Execution Environments and Arm TrustZone*. 2020. url : https://azeria-labs.com/trusted-execution-environments-tee-and-trustzone/ (page 17).

[59] *Open Portable Trusted Execution Environment - OP-TEE*. Linaro Limited. url : https://www.op-tee.org/ (page 17).

[60] *Samsung Teegris*. url : https://developer.samsung.com/teegris/overview.html (page 17).

[61] *The Knox Ecosystem*. Samsung Electronics Co., Ltd. 2020. url : https://docs.samsungknox.com/admin/fundamentals/welcome.htm (pages 17, 68).

[62] *Real-time Kernel Protection (RKP)*. Samsung Electronics Co. Ltd. 2020. url : https://docs.samsungknox.com/admin/whitepaper/kpe/real-time-kernel-protection.htm (pages 17, 68).

[63] *TIMA technology is core to Samsung's state-of-the-art Knox platform*. NC State University. 2014. url : https://www.engr.ncsu.edu/news/2014/11/19/tima-technology-is-core-to-samsungs-state-of-the-art-knox-platform/ (pages 17, 68).

[64] Ahmed M. Azab, Peng Ning, Jitesh Shah, Quan Chen, Rohan Bhutkar, Guruprasad Ganesh, Jia Ma et Wenbo Shen. « Hypervision Across Worlds : Real-Time Kernel Protection from the ARM TrustZone Secure World ». In : CCS '14. Scottsdale, Arizona, USA : Association for Computing Machinery, 2014, p. 90-102. isbn : 9781450329576. doi : 10.1145/2660267.2660350. url : https://doi.org/10.1145/2660267.2660350 (pages 18, 68).

[65] Jin Soo Jang, Sunjune Kong, Minsu Kim, Daegyeong Kim et Brent ByungHoon Kang. « SeCReT : Secure Channel between Rich Execution Environment and Trusted Execution Environment ». In : *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*. The Internet Society, 2015. url : https://www.ndss-symposium.org/ndss2015/secret-secure-channel-between-rich-execution-environment-and-trusted-execution-environment (pages 18, 48).

[66] Shengye Wan, Mingshen Sun, Kun Sun, Ning Zhang et Xu He. « RusTEE : Developing Memory-Safe ARM TrustZone Applications ». In : *Proceedings of the 36th Annual Computer Security Applications Conference*. ACSAC '20. Déc. 2020 (page 18).

[67] H. Sun, K. Sun, Y. Wang, J. Jing et H. Wang. « TrustICE : Hardware-Assisted Isolated Computing Environments on Mobile Devices ». In : *2015 45th Annual*

IEEE/IFIP International Conference on Dependable Systems and Networks. 2015, p. 367-378. DOI : 10.1109/DSN.2015.11 (pages 18, 47).

[68] Ferdinand BRASSER, David GENS, Patrick JAUERNIG, Ahmad-Reza SADEGHI et Emmanuel STAPF. « SANCTUARY : ARMing TrustZone with User-space Enclaves ». In : *26th Annual Network & Distributed System Security Symposium (NDSS)*. Fév. 2019. URL : http://tubiblio.ulb.tu-darmstadt.de/110125/ (pages 18, 47).

[69] Shijun ZHAO, Qianying ZHANG, Yu QIN, Wei FENG et Dengguo FENG. « SecTEE : A Software-Based Approach to Secure Enclave Architecture Using TEE ». In : *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS '19. London, United Kingdom : Association for Computing Machinery, 2019, p. 1723-1740. ISBN : 9781450367479. DOI : 10.1145/3319535.3363205. URL : https://doi.org/10.1145/3319535.3363205 (page 18).

[70] Shijun ZHAO, Qianying ZHANG, Yu QIN, Wei FENG et Dengguo FENG. « Minimal Kernel : An Operating System Architecture for TEE to Resist Board Level Physical Attacks ». In : *22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019)*. Chaoyang District, Beijing : USENIX Association, sept. 2019, p. 105-120. ISBN : 978-1-939133-07-6. URL : https://www.usenix.org/conference/raid2019/presentation/zhao (page 18).

[71] Zhichao HUA, Jinyu GU, Yubin XIA, Haibo CHEN, Binyu ZANG et Haibing GUAN. « vTZ : Virtualizing ARM TrustZone ». In : *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC : USENIX Association, août 2017, p. 541-556. ISBN : 978-1-931971-40-9. URL : https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/hua (page 18).

[72] *Intel® Software Guard Extensions*. Intel Corporation. URL : https://software.intel.com/content/www/us/en/develop/topics/software-guard-extensions.html (page 18).

[73] Hyunyoung OH, Adil AHMAD, Seonghyun PARK, Byoungyoung LEE et Yunheung PAEK. « TRUSTORE : Side-Channel Resistant Storage for SGX Using Intel Hybrid CPU-FPGA ». In : CCS '20. Virtual Event, USA : Association for Computing Machinery, 2020, p. 1903-1918. ISBN : 9781450370899. DOI : 10.1145/3372297.3417265. URL : https://doi.org/10.1145/3372297.3417265 (page 19).

[74] Ming-Wei SHIH, Sangho LEE, Taesoo KIM et Marcus PEINADO. « T-SGX : Eradicating Controlled-Channel Attacks Against Enclave Programs ». In : *Network and Distributed System Security Symposium 2017 (NDSS'17)*. Internet Society, fév. 2017. URL : https://www.microsoft.com/en-us/research/publication/t-sgx-eradicating-controlled-channel-attacks-enclave-programs/ (page 19).

[75] *Library for Intel® Software Guard Extensions - Academic Research*. Intel Corporation. URL : https://software.intel.com/content/www/us/en/develop/topics/software-guard-extensions/academic-research.html (page 19).

[76] Victor COSTAN et Srinivas DEVADAS. « Intel SGX Explained ». In : *IACR Cryptol. ePrint Arch.* 2016 (2016), p. 86. URL : http://eprint.iacr.org/2016/086 (page 19).

[77] *SDK for Intel® Software Guard Extensions*. Intel Corporation. URL : https://software.intel.com/content/www/us/en/develop/topics/software-guard-extensions/sdk.html (pages 19, 46, 67).

[78] *Open Enclave SDK*. URL : https://openenclave.io/sdk/ (pages 19, 46, 67).

[79] *Introducing Asylo : an open-source framework for confidential computing*. URL : https://cloud.google.com/blog/products/gcp/introducing-asylo-an-open-source-framework-for-confidential-computing (pages 19, 46, 67).

[80] *Fortanix Rust Enclave Development Platform*. URL : https://github.com/fortanix/rust-sgx (pages 19, 67).

[81] Huibo WANG, Pei WANG, Yu DING, Mingshen SUN, Yiming JING, Ran DUAN, Long LI, Yulong ZHANG, Tao WEI et Zhiqiang LIN. « Towards Memory Safe Enclave Programming with Rust-SGX ». In : *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS '19. London, United Kingdom : Association for Computing Machinery, 2019, p. 2333-2350. ISBN : 9781450367479. DOI : 10.1145/3319535.3354241. URL : https://doi.org/10.1145/3319535.3354241 (pages 19, 67).

[82] Andrew BAUMANN, Marcus PEINADO et Galen HUNT. « Shielding applications from an untrusted cloud with Haven ». In : *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI '14)*. USENIX - Advanced Computing Systems Association, oct. 2014. URL : https://www.microsoft.com/en-us/research/publication/shielding-applications-from-an-untrusted-cloud-with-haven/ (pages 19, 47, 67).

[83] Christian PRIEBE, Divya MUTHUKUMARAN, Joshua LIND, Huanzhou ZHU, Shujie CUI, Vasily A. SARTAKOV et Peter PIETZUCH. *SGX-LKL : Securing the Host OS Interface for Trusted Execution*. 2020. arXiv : 1908.11143 [cs.OS]. URL : https://arxiv.org/abs/1908.11143 (pages 19, 46, 67).

[84] Sergei ARNAUTOV, Bohdan TRACH, Franz GREGOR, Thomas KNAUTH, Andre MARTIN, Christian PRIEBE, Joshua LIND, Divya MUTHUKUMARAN, Dan O'KEEFFE, Mark L. STILLWELL, David GOLTZSCHE, Dave EYERS, Rüdiger KAPITZA, Peter PIETZUCH et Christof FETZER. « SCONE : Secure Linux Containers with Intel SGX ». In : *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA : USENIX Association, nov. 2016, p. 689-703. ISBN : 978-1-931971-33-1. URL : https://www.usenix.org/conference/osdi16/technical-sessions/presentation/arnautov (pages 19, 67).

[85] Chia-che TSAI, Donald E. PORTER et Mona VIJ. « Graphene-SGX : A Practical Library OS for Unmodified Applications on SGX ». In : *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. Santa Clara, CA : USENIX Association, juill. 2017, p. 645-658. ISBN : 978-1-931971-38-6. URL : https://www.usenix.org/conference/atc17/technical-sessions/presentation/tsai (pages 19, 46, 67).

[86] Shweta SHINDE, Dat Le TIEN, Shruti TOPLE et Prateek SAA. « Panoply : Low-TCB Linux Applications With SGX Enclaves ». In : *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society, 2017. URL : https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/panoply-low-tcb-linux-applications-sgx-enclaves/ (pages 19, 67).

[87] Samuel WEISER, Luca MAYR, Michael SCHWARZ et Daniel GRUSS. « SGXJail : Defeating Enclave Malware via Confinement ». In : *22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019)*. Chaoyang District, Beijing : USENIX Association, sept. 2019, p. 353-366. ISBN : 978-1-939133-07-6. URL : https://www.usenix.org/conference/raid2019/presentation/weiser (page 19).

[88] Jaebaek SEO, Byoungyoung LEE, Seong Min KIM, Ming-Wei SHIH, Insik SHIN, Dongsu HAN et Taesoo KIM. « SGX-Shield : Enabling Address Space Layout Randomization for SGX Programs ». In : *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society, 2017. URL : https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/sgx-shield-enabling-address-space-layout-randomization-sgx-programs/ (pages 19, 67).

[89] Ghada DESSOUKY, Ahmad-Reza SADEGHI et Emmanuel STAPF. « Enclave Computing on RISC-V : A Brighter Future for Security ? » In : *1st International Workshop on Secure RISC-V Architecture Design Exploration (SECRISC-V), co-located with ISPASS-2020*. Avr. 2020. URL : http://tubiblio.ulb.tu-darmstadt.de/119595/ (page 20).

[90] Victor COSTAN, Ilia LEBEDEV et Srinivas DEVADAS. « Sanctum : Minimal Hardware Extensions for Strong Software Isolation ». In : *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX : USENIX Association, août 2016, p. 857-874. ISBN : 978-1-931971-32-4. URL : https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/costan (page 20).

[91] Dayeol LEE, David KOHLBRENNER, Shweta SHINDE, Krste ASANOVIC et Dawn SONG. « Keystone : An Open Framework for Architecting Trusted Execution Environments ». In : *Proceedings of the Fifteenth European Conference on Computer Systems*. EuroSys '20. 2020 (pages 20, 46).

[92] Samuel WEISER, Mario WERNER, Ferdinand BRASSER, Maja MALENKO, Stefan MANGARD et Ahmad-Reza SADEGHI. « TIMBER-V : Tag-Isolated Memory Bringing Fine-grained Enclaves to RISC-V ». In : *26th Annual Network & Distributed System Security Symposium (NDSS)*. Fév. 2019. URL : http://tubiblio.ulb.tu-darmstadt.de/108751/ (page 20).

[93] Muhammad Abdul WAHAB, Pascal COTRET, Mounir Nasr ALLAH, Guillaume HIET, Vianney LAPOTRE et Guy GOGNIAT. « ARMHEx : A hardware extension for DIFT on ARM-based SoCs ». In : *Field Programmable Logic (FPL)*. 2017 (pages 20, 24).

[94] R. N. M. WATSON, J. WOODRUFF, P. G. NEUMANN, S. W. MOORE, J. ANDERSON, D. CHISNALL, N. DAVE, B. DAVIS, K. GUDKA, B. LAURIE, S. J. MURDOCH, R. NORTON, M. ROE, S. SON et M. VADERA. « CHERI : A Hybrid Capability-System Architecture for Scalable Software Compartmentalization ». In : *2015 IEEE Symposium on Security and Privacy*. 2015, p. 20-37. DOI : 10.1109/SP.2015.9 (pages 20, 29).

[95] Raad BAHMANI, Ferdinand BRASSER, Ghada DESSOUKY, Patrick JAUERNIG, Matthias KLIMMEK, Ahmad-Reza SADEGHI et Emmanuel STAPF. « CURE : A Security Architecture with CUstomizable and Resilient Enclaves ». In : *30th USENIX Security Symposium (USENIX Security'21)*. 2021. URL : http://tubiblio.ulb.tu-darmstadt.de/123465/ (page 20).

[96] *Hex Five Security*. URL : https://hex-five.com/ (page 20).

[97] J. G. DYER, M. LINDEMANN, R. PEREZ, R. SAILER, L. VAN DOORN et S. W. SMITH. « Building the IBM 4758 secure coprocessor ». In : *Computer* 34.10 (2001), p. 57-66. DOI : 10.1109/2.955100 (page 20).

[98] Xiaolan ZHANG, Leendert van DOORN, Trent JAEGER, Ronald PEREZ et Reiner SAILER. « Secure Coprocessor-Based Intrusion Detection ». In : *Proceedings of the 10th Workshop on ACM SIGOPS European Workshop*. EW 10. Saint-

Emilion, France : Association for Computing Machinery, 2002, p. 239-242. ISBN : 9781450378062. DOI : 10.1145/1133373.1133423. URL : https://doi.org/10.1145/1133373.1133423 (page 20).

[99] Tarjei MANDT, Mathew SOLNIK et David WANG. *Demystifying the Secure Enclave Processor*. URL : http://mista.nu/research/sep-paper.pdf (page 21).

[100] Alexander EICHNER et Robert BUHREN. *All you ever wanted to know about the AMD Platform Security Processor and were afraid to emulate*. 2020. URL : https://i.blackhat.com/USA-20/Wednesday/us-20-Buhren-All-You-Ever-Wanted-To-Know-About-The-AMD-Platform-Security-Processor-And-Were-Afraid-To-Emulate.pdf (page 21).

[101] Uri FARKAS et Ido Li ON. *AMDFlaws - A Technical DeepDive*. CTS-Labs, 2018. URL : https://cts-labs.com/past-publications (page 21).

[102] *HP Sure Start - Automatic firmware intrusion detection and repair*. HP Development Company, L.P. 2019. URL : http://h10032.www1.hp.com/ctg/Manual/c06216928 (page 21).

[103] *HP Sure Start Hardware Root of Trust version A2, embarqué sur les puces NPCE586HA2MX, NPCE586HA2BX, NPCE576HA2YX*. URL : https://www.ssi.gouv.fr/entreprise/certification_cspn/hp-sure-start-hardware-root-of-trust-version-a2-embarque-sur-les-puces-npce586ha2mx-npce586ha2bx-npce576ha2yx/ (page 21).

[104] *Intel Converged Security and Management Engine (Intel CSME)*. Intel Corporation. 2020. URL : https://www.intel.com/content/dam/www/public/us/en/security-advisory/documents/intel-csme-security-white-paper.pdf (page 21).

[105] Peter BOSCH. *Introduction to the Intel Management Engine OS*. 2019. URL : https://pbx.sh/intelme-sw1/ (page 21).

[106] Mark ERMOLOV et Maxim GORYACHY. *How to hack a turned-off computer, or running unsigned code in Intel ME*. Positive Technology, 2019. URL : https://www.blackhat.com/docs/eu-17/materials/eu-17-Goryachy-How-To-Hack-A-Turned-Off-Computer-Or-Running-Unsigned-Code-In-Intel-Management-Engine-wp.pdf (page 21).

[107] Maxim GORYACHY et Mark ERMOLOV. *Intel VISA : Through the Rabbit Hole*. Positive Technology, 2019. URL : https://i.blackhat.com/asia-19/Thu-March-28/bh-asia-Goryachy-Ermolov-Intel-Visa-Through-the-Rabbit-Hole.pdf (page 21).

[108] Maxim GORYACHY et Mark ERMOLOV. *Where there's a JTAG, there's a way : obtaining full system access via USB*. Positive Technology. 2017. URL : https://www.ptsecurity.com/ww-en/analytics/where-theres-a-jtag-theres-a-way/ (page 21).

[109] Nick L. PETRONI, Timothy FRASER, Jesus MOLINA et William A. ARBAUGH. « Copilot - a Coprocessor-Based Kernel Runtime Integrity Monitor ». In : SSYM'04. San Diego, CA : USENIX Association, 2004, p. 13 (page 22).

[110] Nick L. PETRONI, Timothy FRASER, AAron WALTERS et William A. ARBAUGH. « An Architecture for Specification-Based Detection of Semantic Integrity Violations in Kernel Dynamic Data ». In : *Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15*. USENIX-SS'06. Vancouver, B.C., Canada : USENIX Association, 2006 (page 22).

[111] Yuriy Bulygin et David Samyde. « Chipset based approach to detect virtualization malware ». Black Hat USA. 2008. URL : http://www.c7zero.info/stuff/bh-usa-08-bulygin.ppt (page 22).

[112] Jiang Wang, Angelos Stavrou et Anup Ghosh. « HyperCheck : A Hardware-Assisted Integrity Monitor ». In : *Recent Advances in Intrusion Detection*. Sous la dir. de Somesh Jha, Robin Sommer et Christian Kreibich. Berlin, Heidelberg : Springer Berlin Heidelberg, 2010, p. 158-177. ISBN : 978-3-642-15512-3 (page 22).

[113] Ahmed M. Azab, Peng Ning, Zhi Wang, Xuxian Jiang, Xiaolan Zhang et Nathan C. Skalsky. « HyperSentry : enabling stealthy in-context measurement of hypervisor integrity ». In : *Proceedings of the 17th ACM Conference on Computer and Communications Security* (Chicago, IL, USA). CCS '10. ACM, oct. 2010, p. 38-49. DOI : 10.1145/1866307.1866313 (page 22).

[114] Hyungon Moon, Hojoon Lee, Jihoon Lee, Kihwan Kim, Yunheung Paek et Brent Byunghoon Kang. « Vigilare : Toward Snoop-based Kernel Integrity Monitor ». In : *Proceedings of the 2012 ACM Conference on Computer and Communications Security*. CCS '12. Raleigh, North Carolina, USA : ACM, 2012, p. 28-37. ISBN : 978-1-4503-1651-4. DOI : 10.1145/2382196.2382202. URL : http://doi.acm.org/10.1145/2382196.2382202 (page 22).

[115] Hojoon Lee, HyunGon Moon, DaeHee Jang, Kihwan Kim, Jihoon Lee, Yunheung Paek et Brent ByungHoon Kang. « KI-Mon : A Hardware-assisted Event-triggered Monitoring Platform for Mutable Kernel Object ». In : *22nd USENIX Security Symposium (USENIX Security 13)*. Washington, D.C. : USENIX Association, août 2013, p. 511-526. ISBN : 978-1-931971-03-4. URL : https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/lee (page 22).

[116] Jinyong Lee, Yongje Lee, Hyungon Moon, Ingoo Heo et Yunheung Paek. « Extrax : Security Extension to Extract Cache Resident Information for Snoop-based External Monitors ». In : *DATE '15*. 2015 (page 22).

[117] Lazaros Koromilas, Giorgos Vasiliadis, Ilias Athanasopoulos et Sotiris Ioannidis. « GRIM : Leveraging GPUs for Kernel integrity monitoring ». English. In : *Research in Attacks, Intrusions, and Defenses - 19th International Symposium, RAID 2016, Proceedings*. T. 9854 LNCS. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Springer/Verlag, 2016, p. 3-23. ISBN : 9783319457185. DOI : 10.1007/978-3-319-45719-2_1 (page 23).

[118] Lei Zhou, Jidong Xiao, Kevin Leach, Westley Weimer, Fengwei Zhang et Guojun Wang. « Nighthawk : Transparent System Introspection from Ring -3 ». In : *Computer Security – ESORICS 2019*. Sous la dir. de Kazue Sako, Steve Schneider et Peter Y. A. Ryan. Cham : Springer International Publishing, 2019, p. 217-238. ISBN : 978-3-030-29962-0 (page 23).

[119] Michaël Timbert, Jean-Luc Danger, Sylvain Guilley, Thibault Porteboeuf et Florian Praden. « HCODE : Hardware-Enhanced Real-Time CFI ». In : *PPREW@ACSAC 2014*. New Orleans, United States, déc. 2014. DOI : 10.1145/2689702.2689708. URL : https://hal.archives-ouvertes.fr/hal-01575947 (page 23).

[120] Yongje Lee, Ingoo Heo, Dongil Hwang, Kyungmin Kim et Yunheung Paek. « Towards a Practical Solution to Detect Code Reuse Attacks on ARM Mobile Devices ». In : *Proceedings of the Fourth Workshop on Hardware and Architectural*

Support for Security and Privacy. HASP '15. Portland, Oregon : Association for Computing Machinery, 2015. ISBN : 9781450334839. DOI : 10.1145/2768566.2768569. URL : https://doi.org/10.1145/2768566.2768569 (page 23).

[121] Ronny CHEVALIER, Maugan VILLATEL, David PLAQUIN et Guillaume HIET. « Co-processor-based Behavior Monitoring : Application to the Detection of Attacks Against the System Management Mode ». In : *ACSAC'17*. 2017 (pages 23, 68).

[122] H. KANNAN, M. DALTON et C. KOZYRAKIS. « Decoupling Dynamic Information Flow Tracking with a dedicated coprocessor ». In : *2009 IEEE/IFIP International Conference on Dependable Systems Networks*. 2009, p. 105-114. DOI : 10.1109/DSN.2009.5270347 (page 24).

[123] Jinyong LEE, Ingoo HEO, Yongje LEE et Yunheung PAEK. « Efficient Dynamic Information Flow Tracking on a Processor with Core Debug Interface ». In : *DAC '15*. 2015 (page 24).

[124] Leila DELSHADTEHRANI, Sadullah CANAKCI, Boyou ZHOU, Schuyler ELDRIDGE, Ajay JOSHI et Manuel EGELE. « PHMon : A Programmable Hardware Monitor and Its Security Use Cases ». In : *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, août 2020, p. 807-824. ISBN : 978-1-939133-17-5. URL : https://www.usenix.org/conference/usenixsecurity20/presentation/delshadtehrani (page 24).

[125] J. WANG, KUN SUN et A. STAVROU. « A dependability analysis of hardware-assisted polling integrity checking systems ». In : *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*. 2012, p. 1-12. DOI : 10.1109/DSN.2012.6263962 (page 24).

[126] Daehee JANG, Hojoon LEE, Minsu KIM, Daehyeok KIM, Daegyeong KIM et Brent Byunghoon KANG. « ATRA : Address Translation Redirection Attack Against Hardware-based External Monitors ». In : *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. CCS '14. Scottsdale, Arizona, USA : ACM, 2014, p. 167-178. ISBN : 978-1-4503-2957-6. DOI : 10.1145/2660267.2660303. URL : http://doi.acm.org/10.1145/2660267.2660303 (page 24).

[127] N. ZHANG, H. SUN, K. SUN, W. LOU et Y. T. HOU. « CacheKit : Evading Memory Introspection Using Cache Incoherence ». In : *2016 IEEE European Symposium on Security and Privacy (EuroS P)*. 2016, p. 337-352. DOI : 10.1109/EuroSP.2016.34 (page 24).

[128] S. WAN, J. SUN, K. SUN, N. ZHANG et Q. LI. « SATIN : A Secure and Trustworthy Asynchronous Introspection on Multi-Core ARM Processors ». In : *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 2019, p. 289-301. DOI : 10.1109/DSN.2019.00040 (page 24).

[129] Vasileios P. KEMERLIS, Michalis POLYCHRONAKIS et Angelos D. KEROMYTIS. « ret2dir : Rethinking Kernel Isolation ». In : *23rd USENIX Security Symposium (USENIX Security 14)*. San Diego, CA : USENIX Association, août 2014, p. 957-972. ISBN : 978-1-931971-15-7. URL : https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/kemerlis (page 25).

[130] Z. WANG, C. WU, M. XIE, Y. ZHANG, K. LU, X. ZHANG, Y. LAI, Y. KANG et M. YANG. « SEIMI : Efficient and Secure SMAP-Enabled Intra-process Memory Isolation ». In : *2020 IEEE Symposium on Security and Privacy (SP)*. 2020, p. 592-607. DOI : 10.1109/SP40000.2020.00087 (page 25).

[131] Lucas Davi, Patrick Koeberl et Ahmad-Reza Sadeghi. « Hardware-Assisted Fine-Grained Control-Flow Integrity : Towards Efficient Protection of Embedded Systems Against Software Exploitation ». In : *Proceedings of the 51st Annual Design Automation Conference*. DAC '14. San Francisco, CA, USA : Association for Computing Machinery, 2014, p. 1-6. isbn : 9781450327305. doi : 10.1145/2593069.2596656. url : https://doi.org/10.1145/2593069.2596656 (page 26).

[132] L. Davi, M. Hanreich, D. Paul, A. Sadeghi, P. Koeberl, D. Sullivan, O. Arias et Y. Jin. « HAFIX : Hardware-Assisted Flow Integrity eXtension ». In : *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. 2015, p. 1-6. doi : 10.1145/2744769.2744847 (page 26).

[133] Nick Christoulakis, George Christou, Elias Athanasopoulos et Sotiris Ioannidis. « HCFI : Hardware-Enforced Control-Flow Integrity ». In : *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*. CODASPY '16. New Orleans, Louisiana, USA : Association for Computing Machinery, 2016, p. 38-49. isbn : 9781450339353. doi : 10.1145/2857705.2857722. url : https://doi.org/10.1145/2857705.2857722 (page 26).

[134] *Control-flow EnforcementTechnology Specification*. Intel Corporation. 2019. url : https://software.intel.com/sites/default/files/managed/4d/2a/control-flow-enforcement-technology-preview.pdf (page 26).

[135] Vedvyas Shanbhogue, Deepak Gupta et Ravi Sahita. « Security Analysis of Processor Instruction Set Architecture for Enforcing Control-Flow Integrity ». In : *Proceedings of the 8th International Workshop on Hardware and Architectural Support for Security and Privacy*. HASP '19. Phoenix, AZ, USA : Association for Computing Machinery, 2019. isbn : 9781450372268. doi : 10.1145/3337167.3337175. url : https://doi.org/10.1145/3337167.3337175 (page 26).

[136] *Providing protection for complex software*. Arm Limited. 2020. url : https://developer.arm.com/architectures/learn-the-architecture/providing-protection-for-complex-software (page 26).

[137] Hojoon Lee, Chihyun Song et Brent Byunghoon Kang. « Lord of the X86 Rings : A Portable User Mode Privilege Separation Architecture on X86 ». In : *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. CCS '18. Toronto, Canada : Association for Computing Machinery, 2018, p. 1441-1454. isbn : 9781450356930. doi : 10.1145/3243734.3243748. url : https://doi.org/10.1145/3243734.3243748 (page 26).

[138] Soyeon Park, Sangho Lee, Wen Xu, HyunGon Moon et Taesoo Kim. « libmpk : Software Abstraction for Intel Memory Protection Keys (Intel MPK) ». In : *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. Renton, WA : USENIX Association, juill. 2019, p. 241-254. isbn : 978-1-939133-03-8. url : https://www.usenix.org/conference/atc19/presentation/park-soyeon (page 27).

[139] Anjo Vahldiek-Oberwagner, Eslam Elnikety, Nuno O. Duarte, Michael Sammler, Peter Druschel et Deepak Garg. « ERIM : Secure, Efficient In-process Isolation with Protection Keys (MPK) ». In : *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA : USENIX Association, août 2019, p. 1221-1238. isbn : 978-1-939133-06-9. url : https://www.usenix.org/conference/usenixsecurity19/presentation/vahldiek-oberwagner (page 27).

[140] Xiaoguang Wang, SengMing Yeoh, Pierre Olivier et Binoy Ravindran. « Secure and Efficient In-Process Monitor (and Library) Protection with Intel MPK ». In : *Proceedings of the 13th European Workshop on Systems Security*. EuroSec '20.

Heraklion, Greece : Association for Computing Machinery, 2020, p. 7-12. ISBN : 9781450375238. DOI : 10.1145/3380786.3391398. URL : https://doi.org/10.1145/3380786.3391398 (page 27).

[141] R. Joseph CONNOR, Tyler MCDANIEL, Jared M. SMITH et Max SCHUCHARD. « PKU Pitfalls : Attacks on PKU-based Memory Isolation Systems ». In : *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, août 2020, p. 1409-1426. ISBN : 978-1-939133-17-5. URL : https://www.usenix.org/conference/usenixsecurity20/presentation/connor (page 27).

[142] Lucian MOGOSANU, Ashay RANE et Nathan DAUTENHAHN. « MicroStache : A Lightweight Execution Context for In-Process Safe Region Isolation ». In : *Research in Attacks, Intrusions, and Defenses*. Sous la dir. de Michael BAILEY, Thorsten HOLZ, Manolis STAMATOGIANNAKIS et Sotiris IOANNIDIS. Cham : Springer International Publishing, 2018, p. 359-379. ISBN : 978-3-030-00470-5 (page 27).

[143] David SCHRAMMEL, Samuel WEISER, Stefan STEINEGGER, Martin SCHWARZL, Michael SCHWARZ, Stefan MANGARD et Daniel GRUSS. « Donky : Domain Keys – Efficient In-Process Isolation for RISC-V and x86 ». In : *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, août 2020, p. 1677-1694. ISBN : 978-1-939133-17-5. URL : https://www.usenix.org/conference/usenixsecurity20/presentation/schrammel (pages 27, 66).

[144] Albert KWON, Udit DHAWAN, Jonathan M. SMITH, Thomas F. KNIGHT et Andre DEHON. « Low-Fat Pointers : Compact Encoding and Efficient Gate-Level Implementation of Fat Pointers for Spatial Safety and Capability-Based Security ». In : *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*. CCS '13. Berlin, Germany : Association for Computing Machinery, 2013, p. 721-732. ISBN : 9781450324779. DOI : 10.1145/2508859.2516713. URL : https://doi.org/10.1145/2508859.2516713 (page 27).

[145] Udit DHAWAN, Catalin HRITCU, Raphael RUBIN, Nikos VASILAKIS, Silviu CHIRICESCU, Jonathan M. SMITH, Thomas F. KNIGHT, Benjamin C. PIERCE et Andre DEHON. « Architectural Support for Software-Defined Metadata Processing ». In : *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '15. Istanbul, Turkey : Association for Computing Machinery, 2015, p. 487-502. ISBN : 9781450328357. DOI : 10.1145/2694344.2694383. URL : https://doi.org/10.1145/2694344.2694383 (page 27).

[146] A. A. d. AMORIM, M. DÉNÈS, N. GIANNARAKIS, C. HRITCU, B. C. PIERCE, A. SPECTOR-ZABUSKY et A. TOLMACH. « Micro-Policies : Formally Verified, Tag-Based Security Monitors ». In : *2015 IEEE Symposium on Security and Privacy*. 2015, p. 813-830. DOI : 10.1109/SP.2015.55 (page 28).

[147] N. ROESSLER et A. DEHON. « Protecting the Stack with Metadata Policies and Tagged Hardware ». In : *2018 IEEE Symposium on Security and Privacy (SP)*. 2018, p. 478-495. DOI : 10.1109/SP.2018.00066 (page 28).

[148] Wei HUANG, Zhen HUANG, Dhaval MIYANI et David LIE. « LMP : Light-Weighted Memory Protection with Hardware Assistance ». In : ACSAC '16. Los Angeles, California, USA : Association for Computing Machinery, 2016, p. 460-470. ISBN : 9781450347716. DOI : 10.1145/2991079.2991089. URL : https://dl.acm.org/doi/10.1145/2991079.2991089 (page 28).

[149] Oleksii OLEKSENKO, Dmitrii KUVAISKII, Pramod BHATOTIA, Pascal FELBER et Christof FETZER. « Intel MPX Explained : A Cross-Layer Analysis of the Intel

MPX System Stack ». In : *Proc. ACM Meas. Anal. Comput. Syst.* 2.2 (juin 2018). DOI : 10.1145/3224423. URL : https://doi.org/10.1145/3224423 (page 28).

[150] Hans LILJESTRAND, Thomas NYMAN, Kui WANG, Carlos Chinea PEREZ, Jan-Erik EKBERG et N. ASOKAN. « PAC it up : Towards Pointer Integrity using ARM Pointer Authentication ». In : *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA : USENIX Association, août 2019, p. 177-194. ISBN : 978-1-939133-06-9. URL : https://www.usenix.org/conference/usenixsecurity19/presentation/liljestrand (page 29).

[151] N. WESLEY FILARDO, B. F. GUTSTEIN, J. WOODRUFF, S. AINSWORTH, L. PAUL-TRIFU, B. DAVIS, H. XIA, E. TOMASZ NAPIERALA, A. RICHARDSON, J. BALDWIN, D. CHISNALL, J. CLARKE, K. GUDKA, A. JOANNOU, A. THEODORE MARKETTOS, A. MAZZINGHI, R. M. NORTON, M. ROE, P. SEWELL, S. SON, T. M. JONES, S. W. MOORE, P. G. NEUMANN et R. N. M. WATSON. « Cornucopia : Temporal Safety for CHERI Heaps ». In : *2020 IEEE Symposium on Security and Privacy (SP)*. 2020, p. 608-625. DOI : 10.1109/SP40000.2020.00098 (page 29).

[152] Brooks DAVIS, Robert N. M. WATSON, Alexander RICHARDSON, Peter G. NEUMANN, Simon W. MOORE, John BALDWIN, David CHISNALL, James CLARKE, Nathaniel Wesley FILARDO, Khilan GUDKA, Alexandre JOANNOU, Ben LAURIE, A. Theodore MARKETTOS, J. Edward MASTE, Alfredo MAZZINGHI, Edward Tomasz NAPIERALA, Robert M. NORTON, Michael ROE, Peter SEWELL, Stacey SON et Jonathan WOODRUFF. « CheriABI : Enforcing Valid Pointer Provenance and Minimizing Pointer Privilege in the POSIX C Run-Time Environment ». In : ASPLOS '19. Providence, RI, USA : Association for Computing Machinery, 2019, p. 379-393. ISBN : 9781450362405. DOI : 10.1145/3297858.3304042. URL : https://doi.org/10.1145/3297858.3304042 (page 29).

[153] *Morello Platform Model Reference Guide*. ARM Limited. 2020. URL : https://developer.arm.com/documentation/102225/0100/ (page 29).

[154] Drew ZAGIEBOYLO, G. Edward SUH et Andrew C. MYERS. « Using Information Flow to Design an ISA that Controls Timing Channels ». In : *32nd IEEE Computer Security Foundations Symposium, CSF 2019, Hoboken, NJ, USA, June 25-28, 2019*. IEEE, 2019, p. 272-287. DOI : 10.1109/CSF.2019.00026. URL : https://doi.org/10.1109/CSF.2019.00026 (page 29).

[155] C. SONG, H. MOON, M. ALAM, I. YUN, B. LEE, T. KIM, W. LEE et Y. PAEK. « HDFI : Hardware-Assisted Data-Flow Isolation ». In : *2016 IEEE Symposium on Security and Privacy (SP)*. 2016, p. 1-17. DOI : 10.1109/SP.2016.9 (page 29).

[156] Brian BELLEVILLE, Hyungon MOON, Jangseop SHIN, Dongil HWANG, Joseph M. NASH, Seonhwa JUNG, Yeoul NA, Stijn VOLCKAERT, Per LARSEN, Yunheung PAEK et Michael FRANZ. « Hardware Assisted Randomization of Data ». In : *Research in Attacks, Intrusions, and Defenses*. Sous la dir. de Michael BAILEY, Thorsten HOLZ, Manolis STAMATOGIANNAKIS et Sotiris IOANNIDIS. Cham : Springer International Publishing, 2018, p. 337-358. ISBN : 978-3-030-00470-5 (page 29).

[157] Xiaoguang WANG, SengMing YEOH, Robert LYERLY, Pierre OLIVIER, Sang-Hoon KIM et Binoy RAVINDRAN. « A Framework for Software Diversification with ISA Heterogeneity ». In : *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*. San Sebastian : USENIX Association, oct. 2020, p. 427-442. ISBN : 978-1-939133-18-2. URL : https://www.usenix.org/conference/raid2020/presentation/wang-xiaoguang (page 30).

[158] Ruan DE CLERCQ, Johannes GÖTZFRIED, David ÜBLER, Pieter MAENE et Ingrid VERBAUWHEDE. « SOFIA : Software and control flow integrity architecture ». In : *Computers & Security* 68 (2017), p. 16-35. ISSN : 0167-4048. DOI : https://doi.org/10.1016/j.cose.2017.03.013. URL : http://www.sciencedirect.com/science/article/pii/S0167404817300664 (page 30).

[159] Jiyong YU, Lucas HSIUNG, Mohamad El HAJJ et Christopher W. FLETCHER. « Data Oblivious ISA Extensions for Side Channel-Resistant and High Performance Computing ». In : *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society, 2019. URL : https://www.ndss-symposium.org/ndss-paper/data-oblivious-isa-extensions-for-side-channel-resistant-and-high-performance-computing/ (pages 30, 64).

[160] Cristiano PEREIRA, Beeman STRONG et Gilles POKAM. *Intel PT Micro Tutorial*. Intel Corporation. 2015. URL : https://sites.google.com/site/intelptmicrotutorial/home (page 30).

[161] Xinyang GE, Weidong CUI et Trent JAEGER. « GRIFFIN : Guarding Control Flows Using Intel Processor Trace ». In : *ASPLOS '17*. 2017 (page 30).

[162] Yufei GU, Qingchuan ZHAO, Yinqian ZHANG et Zhiqiang LIN. « PT-CFI : Transparent Backward-Edge Control Flow Violation Detection Using Intel Processor Trace ». In : *7th ACM on Conference on Data and Application Security and Privacy*. CODASPY '17. Scottsdale, Arizona, USA : ACM, 2017, p. 173-184 (page 30).

[163] Sergej SCHUMILO, Cornelius ASCHERMANN, Robert GAWLIK, Sebastian SCHINZEL et Thorsten HOLZ. « kAFL : Hardware-Assisted Feedback Fuzzing for OS Kernels ». In : *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC : USENIX Association, août 2017, p. 167-182. ISBN : 978-1-931971-40-9. URL : https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/schumilo (page 30).

[164] *CoreSight Architecture*. Arm Limited. URL : https://www.kernel.org/doc/html/latest/trace/coresight/coresight.html (page 30).

[165] Mathieu POIRIER. *Coresight - HW Assisted Tracing on ARM*. Linaro. URL : https://www.kernel.org/doc/html/latest/trace/coresight/coresight.html (page 30).

[166] Zhenyu NING et Fengwei ZHANG. « Ninja : Towards Transparent Tracing and Debugging on ARM ». In : *USENIX Security'17*. 2017 (page 31).

[167] Hyungon MOON, Jinyong LEE, Dongil HWANG, Seonhwa JUNG, Jiwon SEO et Yunheung PAEK. « Architectural Supports to Protect OS Kernels from Code-Injection Attacks and Their Applications ». In : *ACM Trans. Des. Autom. Electron. Syst.* 23.1 (août 2017). ISSN : 1084-4309. DOI : 10.1145/3110223. URL : https://doi.org/10.1145/3110223 (page 31).

[168] Yunlan DU, Zhenyu NING, Jun XU, Zhilong WANG, Yueh-Hsun LIN, Fengwei ZHANG, Xinyu XING et Bing MAO. « HART : Hardware-Assisted Kernel Module Tracing on Arm ». In : *Computer Security – ESORICS 2020*. Sous la dir. de Liqun CHEN, Ninghui LI, Kaitai LIANG et Steve SCHNEIDER. Cham : Springer International Publishing, 2020, p. 316-337. ISBN : 978-3-030-58951-6 (page 31).

[169] Konstantin SEREBRYANY, Derek BRUENING, Alexander POTAPENKO et Dmitriy VYUKOV. « AddressSanitizer : A Fast Address Sanity Checker ». In : *2012 USENIX Annual Technical Conference (USENIX ATC 12)*. Boston, MA : USENIX

Association, juin 2012, p. 309-318. URL : https://www.usenix.org/conference/atc12/technical-sessions/presentation/serebryany (page 31).

[170] Z. NING et F. ZHANG. « Understanding the Security of ARM Debugging Features ». In : *2019 IEEE Symposium on Security and Privacy (SP)*. 2019, p. 602-619. DOI : 10.1109/SP.2019.00061 (page 31).

[171] Adrian TANG, Simha SETHUMADHAVAN et Salvatore J. STOLFO. « Unsupervised Anomaly-Based Malware Detection Using Hardware Features ». In : *Research in Attacks, Intrusions and Defenses*. Sous la dir. d'Angelos STAVROU, Herbert BOS et Georgios PORTOKALIDIS. Cham : Springer International Publishing, 2014, p. 109-129. ISBN : 978-3-319-11379-1 (page 31).

[172] S. DAS, J. WERNER, M. ANTONAKAKIS, M. POLYCHRONAKIS et F. MONROSE. « SoK : The Challenges, Pitfalls, and Perils of Using Hardware Performance Counters for Security ». In : *2019 IEEE Symposium on Security and Privacy (SP)*. 2019, p. 20-38. DOI : 10.1109/SP.2019.00021 (page 31).

[173] Alastair REID. « Trustworthy Specifications of ARM v8-A and v8-M System Level Architecture ». In : *FMCAD'16*. 2016 (page 32).

[174] Alastair REID, Rick CHEN, Anastasios DELIGIANNIS, David GILDAY, David HOYES, Will KEEN, Ashan PATHIRANE, Owen SHEPHERD, Peter VRABEL et Ali ZAIDI. « End-to-End Verification of Processors with ISA-Formal ». In : *Computer Aided Verification*. Sous la dir. de Swarat CHAUDHURI et Azadeh FARZAN. Cham : Springer International Publishing, 2016, p. 42-58. ISBN : 978-3-319-41540-6 (page 32).

[175] *A-Profile Architectures | Exploration tools*. ARM Limited. URL : https://developer.arm.com/architectures/cpu-architecture/a-profile/exploration-tools (page 32).

[176] Sandeep DASGUPTA, Daejun PARK, Theodoros KASAMPALIS, Vikram S. ADVE et Grigore ROŞU. « A Complete Formal Semantics of X86-64 User-Level Instruction Set Architecture ». In : *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI 2019. Phoenix, AZ, USA : Association for Computing Machinery, 2019, p. 1133-1148. ISBN : 9781450367127. DOI : 10.1145/3314221.3314601. URL : https://doi.org/10.1145/3314221.3314601 (page 32).

[177] Alasdair ARMSTRONG, Thomas BAUEREISS, Brian CAMPBELL, Alastair REID, Kathryn E. GRAY, Robert M. NORTON, Prashanth MUNDKUR, Mark WASSELL, Jon FRENCH, Christopher PULTE, Shaked FLUR, Ian STARK, Neel KRISHNASWAMI et Peter SEWELL. « ISA Semantics for ARMv8-a, RISC-v, and CHERI-MIPS ». In : *Proc. ACM Program. Lang.* 3.POPL (jan. 2019). DOI : 10.1145/3290384. URL : https://doi.org/10.1145/3290384 (page 32).

[178] Joonwon CHOI, Muralidaran VIJAYARAGHAVAN, Benjamin SHERMAN, Adam CHLIPALA et ARVIND. « Kami : A Platform for High-Level Parametric Hardware Specification and Its Modular Verification ». In : 1.ICFP (août 2017). DOI : 10.1145/3110268. URL : https://doi.org/10.1145/3110268 (pages 32, 65).

[179] Thomas BOURGEAT, Clément PIT-CLAUDEL, Adam CHLIPALA et ARVIND. « The Essence of Bluespec : A Core Language for Rule-Based Hardware Design ». In : *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI 2020. London, UK : Association for Computing Machinery, 2020, p. 243-257. ISBN : 9781450376136. DOI : 10.1145/3385412.3385965. URL : https://doi.org/10.1145/3385412.3385965 (pages 32, 65).

[180] *GitHub - sifive/RiscvSpecForma*. SiFive. URL : https://github.com/sifive/RiscvSpecFormal (page 32).

[181] Thomas LETAN, Pierre CHIFFLIER, Guillaume HIET, Pierre NÉRON et Benjamin MORIN. « SpecCert : Specifying and Verifying Hardware-Based Security Enforcement ». In : *FM 2016 : Formal Methods - 21st International Symposium, Limassol, Cyprus, November 9-11, 2016, Proceedings*. Sous la dir. de John S. FITZGERALD, Constance L. HEITMEYER, Stefania GNESI et Anna PHILIPPOU. T. 9995. Lecture Notes in Computer Science. 2016, p. 496-512. DOI : 10.1007/978-3-319-48989-6_30. URL : https://doi.org/10.1007/978-3-319-48989-6%5C_30 (page 32).

[182] Andrew FERRAIUOLO, Rui XU, Danfeng ZHANG, Andrew C. MYERS et G. Edward SUH. « Verification of a Practical Hardware Security Architecture Through Static Information Flow Analysis ». In : *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '17. Xi'an, China : Association for Computing Machinery, 2017, p. 555-568. ISBN : 9781450344654. DOI : 10.1145/3037697.3037739. URL : https://doi.org/10.1145/3037697.3037739 (page 33).

[183] Shuwen DENG, Doğuhan GÜMÜŞOĞLU, Wenjie XIONG, Y. Serhan GENER, Onur DEMIR et Jakub SZEFER. « SecChisel Framework for Security Verification of Secure Processor Architectures ». In : *Proceedings of the Workshop on Hardware and Architectural Support for Security and Privacy*. HASP. Juin 2019 (page 33).

[184] K. NIENHUIS, A. JOANNOU, T. BAUEREISS, A. FOX, M. ROE, B. CAMPBELL, M. NAYLOR, R. M. NORTON, S. W. MOORE, P. G. NEUMANN, I. STARK, R. N. M. WATSON et P. SEWELL. « Rigorous engineering for hardware security : Formal modelling and proof in the CHERI design and implementation process ». In : *2020 IEEE Symposium on Security and Privacy (SP)*. 2020, p. 1003-1020. DOI : 10.1109/SP40000.2020.00055 (page 33).

[185] Ghada DESSOUKY, David GENS, Patrick HANEY, Garrett PERSYN, Arun KANUPARTHI, Hareesh KHATTRI, Jason M. FUNG, Ahmad-Reza SADEGHI et Jeyavijayan RAJENDRAN. « HardFails : Insights into Software-Exploitable Hardware Bugs ». In : *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA : USENIX Association, août 2019, p. 213-230. ISBN : 978-1-939133-06-9. URL : https://www.usenix.org/conference/usenixsecurity19/presentation/dessouky (page 34).

[186] Jakub SZEFER. « Survey of Microarchitectural Side and Covert Channels, Attacks, and Defenses ». In : *Journal of Hardware and Systems Security* 3.3 (sept. 2019), p. 219-234. ISSN : 2509-3436. DOI : 10.1007/s41635-018-0046-1. URL : https://doi.org/10.1007/s41635-018-0046-1 (page 35).

[187] R. SPREITZER, V. MOONSAMY, T. KORAK et S. MANGARD. « Systematic Classification of Side-Channel Attacks : A Case Study for Mobile Devices ». In : *IEEE Communications Surveys Tutorials* 20.1 (2018), p. 465-488. DOI : 10.1109/COMST.2017.2779824 (page 35).

[188] Alejandro CABRERA ALDAYA, Billy B. BRUMLEY, Sohaib UL HASSAN, Cesar PEREIDA GARCÍA et Nicola TUVERI. « Port Contention for Fun and Profit ». English. In : *2019 IEEE Symposium on Security and Privacy (SP) (2019)*. jufoid=57507. IEEE, mai 2019, p. 1037-1054. DOI : 10.1109/SP.2019.00066 (page 36).

[189] Ben GRAS, Cristiano GIUFFRIDA, Michael KURTH, Herbert BOS et Kaveh RAZAVI. « ABSynthe : Automatic Blackbox Side-channel Synthesis on Commodity Microar-

chitectures ». In : *NDSS*. Fév. 2020. URL : https://download.vusec.net/papers/absynthe_ndss20.pdf (pages 36, 64).

[190] Dag Arne Osvik, Adi Shamir et Eran Tromer. « Cache Attacks and Countermeasures : The Case of AES ». In : *Topics in Cryptology – CT-RSA 2006*. Sous la dir. de David Pointcheval. Berlin, Heidelberg : Springer Berlin Heidelberg, 2006, p. 1-20. ISBN : 978-3-540-32648-9 (page 36).

[191] Yinqian Zhang, Ari Juels, Michael K. Reiter et Thomas Ristenpart. « Cross-VM side channels and their use to extract private keys ». In : *19th ACM conference on Computer and communications security*. Raleigh, North Carolina, USA : ACM, 2012, p. 305-316 (page 36).

[192] Yuval Yarom et Katrina Falkner. « FLUSH+RELOAD : A High Resolution, Low Noise, L3 Cache Side-Channel Attack ». In : *23rd USENIX Security Symposium (USENIX Security 14)*. San Diego, CA : USENIX Association, août 2014, p. 719-732. ISBN : 978-1-931971-15-7. URL : https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/yarom (page 36).

[193] Gorka Irazoqui Apecechea, Mehmet Sinan Inci, Thomas Eisenbarth et Berk Sunar. « Wait a Minute ! A fast, Cross-VM Attack on AES ». In : *Research in Attacks, Intrusions and Defenses - 17th International Symposium, RAID 2014, Gothenburg, Sweden, September 17-19, 2014. Proceedings*. Sous la dir. d'Angelos Stavrou, Herbert Bos et Georgios Portokalidis. T. 8688. Lecture Notes in Computer Science. Springer, 2014, p. 299-319. DOI : 10.1007/978-3-319-11379-1_15. URL : https://doi.org/10.1007/978-3-319-11379-1%5C_15 (page 36).

[194] Xiaokuan Zhang, Yuan Xiao et Yinqian Zhang. « Return-Oriented Flush-Reload Side Channels on ARM and Their Implications for Android Devices ». In : *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. CCS '16. Vienna, Austria : ACM, 2016, p. 858-870. DOI : 10.1145/2976749.2978360 (page 36).

[195] F. Liu, Y. Yarom, Q. Ge, G. Heiser et R. B. Lee. « Last-Level Cache Side-Channel Attacks are Practical ». In : *2015 IEEE Symposium on Security and Privacy*. 2015, p. 605-622. DOI : 10.1109/SP.2015.43 (page 37).

[196] Gorka Irazoqui Apecechea, Thomas Eisenbarth et Berk Sunar. « S$A : A Shared Cache Attack That Works across Cores and Defies VM Sandboxing - and Its Application to AES ». In : *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*. IEEE Computer Society, 2015, p. 591-604. DOI : 10.1109/SP.2015.42. URL : https://doi.org/10.1109/SP.2015.42 (page 37).

[197] Clémentine Maurice, Manuel Weber, Michael Schwarz, Lukas Giner, Daniel Gruss, Carlo Alberto Boano, Stefan Mangard et Kay Römer. « Hello from the Other Side : SSH over Robust Cache Covert Channels in the Cloud ». In : *NDSS'17*. 2017 (page 37).

[198] Daniel Gruss, Clémentine Maurice, Klaus Wagner et Stefan Mangard. « Flush+Flush : A Fast and Stealthy Cache Attack ». In : DIMVA 2016. San Sebastián, Spain : Springer-Verlag, 2016, p. 279-299. ISBN : 9783319406664. DOI : 10.1007/978-3-319-40667-1_14. URL : https://doi.org/10.1007/978-3-319-40667-1_14 (page 37).

[199] Samira Briongos, Pedro Malagon, Jose M. Moya et Thomas Eisenbarth. « RELOAD+REFRESH : Abusing Cache Replacement Policies to Perform Steal-

thy Cache Attacks ». In : *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, août 2020, p. 1967-1984. ISBN : 978-1-939133-17-5. URL : https://www.usenix.org/conference/usenixsecurity20/presentation/briongos (page 37).

[200] Moritz LIPP, Daniel GRUSS, Raphael SPREITZER, Clémentine MAURICE et Stefan MANGARD. « ARMageddon : Last-Level Cache Attacks on Mobile Devices ». In : *USENIX Security'16*. 2016 (page 37).

[201] Craig DISSELKOEN, David KOHLBRENNER, Leo PORTER et Dean TULLSEN. « Prime+Abort : A Timer-Free High-Precision L3 Cache Attack using Intel TSX ». In : *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC : USENIX Association, août 2017, p. 51-67. ISBN : 978-1-931971-40-9. URL : https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/disselkoen (page 37).

[202] Daniel GRUSS, Raphael SPREITZER et Stefan MANGARD. « Cache Template Attacks : Automating Attacks on Inclusive Last-Level Caches ». In : *24th USENIX Security Symposium (USENIX Security 15)*. Washington, D.C. : USENIX Association, août 2015, p. 897-912. ISBN : 978-1-939133-11-3. URL : https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/gruss (page 37).

[203] Gorka IRAZOQUI, Thomas EISENBARTH et Berk SUNAR. « Cross Processor Cache Attacks ». In : *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, AsiaCCS 2016, Xi'an, China, May 30 - June 3, 2016*. Sous la dir. de Xiaofeng CHEN, XiaoFeng WANG et Xinyi HUANG. ACM, 2016, p. 353-364. DOI : 10.1145/2897845.2897867. URL : https://doi.org/10.1145/2897845.2897867 (page 38).

[204] Thomas ALLAN, Billy Bob BRUMLEY, Katrina E. FALKNER, Joop van de POL et Yuval YAROM. « Amplifying side channels through performance degradation ». In : *Proceedings of the 32nd Annual Conference on Computer Security Applications, ACSAC 2016, Los Angeles, CA, USA, December 5-9, 2016*. Sous la dir. de Stephen SCHWAB, William K. ROBERTSON et Davide BALZAROTTI. ACM, 2016, p. 422-435. URL : http://dl.acm.org/citation.cfm?id=2991084 (page 38).

[205] M. YAN, R. SPRABERY, B. GOPIREDDY, C. FLETCHER, R. CAMPBELL et J. TORRELLAS. « Attack Directories, Not Caches : Side Channel Attacks in a Non-Inclusive World ». In : *2019 IEEE Symposium on Security and Privacy (SP)*. 2019, p. 888-904. DOI : 10.1109/SP.2019.00004 (page 38).

[206] Ben GRAS, Kaveh RAZAVI, Herbert BOS et Cristiano GIUFFRIDA. « Translation Leak-aside Buffer : Defeating Cache Side-channel Protections with TLB Attacks ». In : *USENIX Security*. Pwnie Award Nomination for Most Innovative Research. Août 2018. URL : Paper=https://download.vusec.net/papers/tlbleed_sec18.pdf%20Slides=https://www.usenix.org/sites/default/files/conference/protected-files/security18_slides_gras.pdf%20Web=https://www.vusec.net/projects/tlbleed%20Code=https://github.com/vusec/tlbkit%20Press=https://goo.gl/eepq1y (page 38).

[207] Stephan van SCHAIK, Cristiano GIUFFRIDA, Herbert BOS et Kaveh RAZAVI. « Malicious Management Unit : Why Stopping Cache Attacks in Software is Harder Than You Think ». In : *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD : USENIX Association, août 2018, p. 937-954. ISBN : 978-1-

939133-04-5. URL : https://www.usenix.org/conference/usenixsecurity18/presentation/van-schaik (page 38).

[208] Ben GRAS, Kaveh RAZAVI, Erik BOSMAN, Herbert BOS et Cristiano GIUFFRIDA. « ASLR on the Line : Practical Cache Attacks on the MMU ». In : *NDSS*. Pwnie Award for Most Innovative Research, DCSR Paper Award. Fév. 2017. URL : Paper=https://download.vusec.net/papers/anc_ndss17.pdf%20Slides=https://vusec.net/wp-content/uploads/2016/11/TalkGras.pdf%20Web=https://www.vusec.net/projects/anc%20Code=https://github.com/vusec/revanc%20Press=https://goo.gl/KL4Bta (pages 39, 56).

[209] R. HUND, C. WILLEMS et T. HOLZ. « Practical Timing Side Channel Attacks against Kernel Space ASLR ». In : *2013 IEEE Symposium on Security and Privacy*. 2013, p. 191-205. DOI : 10.1109/SP.2013.23 (pages 39, 56).

[210] Daniel GRUSS, Clémentine MAURICE, Anders FOGH, Moritz LIPP et Stefan MANGARD. « Prefetch Side-Channel Attacks : Bypassing SMAP and Kernel ASLR ». In : *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. CCS '16. Vienna, Austria : Association for Computing Machinery, 2016, p. 368-379. ISBN : 9781450341394. DOI : 10.1145/2976749.2978356. URL : https://doi.org/10.1145/2976749.2978356 (pages 39, 56).

[211] Yossef OREN, Vasileios P. KEMERLIS, Simha SETHUMADHAVAN et Angelos D. KEROMYTIS. « The Spy in the Sandbox : Practical Cache Attacks in JavaScript and Their Implications ». In : *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. CCS '15. Denver, Colorado, USA : Association for Computing Machinery, 2015, p. 1406-1418. ISBN : 9781450338325. DOI : 10.1145/2810103.2813708. URL : https://doi.org/10.1145/2810103.2813708 (page 39).

[212] Anatoly SHUSTERMAN, Lachlan KANG, Yarden HASKAL, Yosef MELTSER, Prateek MITTAL, Yossi OREN et Yuval YAROM. « Robust Website Fingerprinting Through the Cache Occupancy Channel ». In : *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA : USENIX Association, août 2019, p. 639-656. ISBN : 978-1-939133-06-9. URL : https://www.usenix.org/conference/usenixsecurity19/presentation/shusterman (page 39).

[213] Ramya Jayaram MASTI, Devendra RAI, Aanjhan RANGANATHAN, Christian MÜLLER, Lothar THIELE et Srdjan CAPKUN. « Thermal Covert Channels on Multi-core Platforms ». In : *24th USENIX Security Symposium (USENIX Security 15)*. Washington, D.C. : USENIX Association, août 2015, p. 865-880. ISBN : 978-1-939133-11-3. URL : https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/masti (page 39).

[214] Moritz LIPP, Andreas KOGLER, David OSWALD, Michael SCHWARZ, Catherine EASDON, Claudio CANELLA et Daniel GRUSS. « PLATYPUS : Software-based Power Side-Channel Attacks on x86 ». In : *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2021 (page 39).

[215] Moritz LIPP, Michael SCHWARZ, Daniel GRUSS, Thomas PRESCHER, Werner HAAS, Anders FOGH, Jann HORN, Stefan MANGARD, Paul KOCHER, Daniel GENKIN, Yuval YAROM et Mike HAMBURG. « Meltdown : Reading Kernel Memory from User Space ». In : *27th USENIX Security Symposium (USENIX Security 18)*. 2018 (pages 40, 42).

[216] Paul KOCHER, Jann HORN, Anders FOGH, Daniel GENKIN, Daniel GRUSS, Werner HAAS, Mike HAMBURG, Moritz LIPP, Stefan MANGARD, Thomas PRESCHER,

Michael SCHWARZ et Yuval YAROM. « Spectre Attacks : Exploiting Speculative Execution ». In : *40th IEEE Symposium on Security and Privacy (S&P'19)*. 2019 (pages 40, 41).

[217] Claudio CANELLA, Jo Van BULCK, Michael SCHWARZ, Moritz LIPP, Benjamin von BERG, Philipp ORTNER, Frank PIESSENS, Dmitry EVTYUSHKIN et Daniel GRUSS. « A Systematic Evaluation of Transient Execution Attacks and Defenses ». In : *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA : USENIX Association, août 2019, p. 249-266. ISBN : 978-1-939133-06-9. URL : https://www.usenix.org/conference/usenixsecurity19/presentation/canella (pages 40, 48, 57, 66).

[218] Wenjie XIONG et Jakub SZEFER. *Survey of Transient Execution Attacks*. 2020. arXiv : 2005.13435 [cs.CR]. URL : https://arxiv.org/pdf/2005.13435.pdf (pages 40, 66).

[219] *SafeSide*. Google. URL : https://github.com/google/safeside (page 40).

[220] *Transient Execution Attacks*. Graz University of Technology. URL : https://transient.fail/ (page 40).

[221] Giorgi MAISURADZE et Christian ROSSOW. « Ret2spec : Speculative Execution Using Return Stack Buffers ». In : *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. CCS '18. Toronto, Canada : Association for Computing Machinery, 2018, p. 2109-2122. ISBN : 9781450356930. DOI : 10.1145/3243734.3243761. URL : https://doi.org/10.1145/3243734.3243761 (page 41).

[222] Vladimir KIRIANSKY et Carl WALDSPURGER. *Speculative Buffer Overflows : Attacks and Defenses*. 2018. arXiv : 1807.03757 [cs.CR]. URL : https://arxiv.org/abs/1807.03757 (page 41).

[223] Atri BHATTACHARYYA, Alexandra SANDULESCU, Matthias NEUGSCHWANDTNER, Alessandro SORNIOTTI, Babak FALSAFI, Mathias PAYER et Anil KURMUS. « SMoTherSpectre : Exploiting Speculative Execution through Port Contention ». In : *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS '19. London, United Kingdom : Association for Computing Machinery, 2019, p. 785-800. ISBN : 9781450367479. DOI : 10.1145/3319535.3363194. URL : https://doi.org/10.1145/3319535.3363194 (page 41).

[224] Michael SCHWARZ, Martin SCHWARZL, Moritz LIPP, Jon MASTERS et Daniel GRUSS. « NetSpectre : Read Arbitrary Memory over Network ». In : *Computer Security – ESORICS 2019*. Sous la dir. de Kazue SAKO, Steve SCHNEIDER et Peter Y. A. RYAN. Cham : Springer International Publishing, 2019, p. 279-299. ISBN : 978-3-030-29959-0 (page 41).

[225] Jack WAMPLER, Ian MARTINY et Eric WUSTROW. « ExSpectre : Hiding Malware in Speculative Execution ». In : *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society, 2019. URL : https://www.ndss-symposium.org/ndss-paper/exspectre-hiding-malware-in-speculative-execution/ (page 41).

[226] Atri BHATTACHARYYA, Andrés SÁNCHEZ, Esmaeil M. KORUYEH, Nael ABU-GHAZALEH, Chengyu SONG et Mathias PAYER. « SpecROP : Speculative Exploitation of ROP Chains ». In : *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*. San Sebastian : USENIX Association, oct. 2020, p. 1-16. ISBN : 978-1-939133-18-2. URL : https://www.usenix.org/conference/raid2020/presentation/bhattacharyya (page 41).

[227] Enes GOKTAS, Kaveh RAZAVI, Georgios PORTOKALIDIS, Herbert BOS et Cristiano GIUFFRIDA. « Speculative Probing : Hacking Blind in the Spectre Era ». In : *CCS*. Nov. 2020. URL : Paper=https://download.vusec.net/papers/blindside_ccs20.pdf%20Web=https://www.vusec.net/projects/blindside%20Code=https://github.com/vusec/blindside%20Press=https://bit.ly/3c4MkhU (page 42).

[228] Julian STECKLINA et Thomas PRESCHER. *LazyFP : Leaking FPU Register State using Microarchitectural Side-Channels*. 2018. arXiv : 1806.07480 [cs.OS]. URL : https://arxiv.org/abs/1806.07480 (page 42).

[229] Claudio CANELLA, Daniel GENKIN, Lukas GINER, Daniel GRUSS, Moritz LIPP, Marina MINKIN, Daniel MOGHIMI, Frank PIESSENS, Michael SCHWARZ, Berk SUNAR, Jo Van BULCK et Yuval YAROM. « Fallout : Leaking Data on Meltdown-resistant CPUs ». In : *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*. Sous la dir. de Lorenzo CAVALLARO, Johannes KINDER, XiaoFeng WANG et Jonathan KATZ. ACM, 2019, p. 769-784. DOI : 10.1145/3319535.3363219. URL : https://doi.org/10.1145/3319535.3363219 (page 42).

[230] Stephan van SCHAIK, Alyssa MILBURN, Sebastian ÖSTERLUND, Pietro FRIGO, Giorgi MAISURADZE, Kaveh RAZAVI, Herbert BOS et Cristiano GIUFFRIDA. « RIDL : Rogue In-flight Data Load ». In : *S&P*. Intel Bounty Reward (Highest To Date), Pwnie Award Nomination for Most Innovative Research, CSAW Best Paper Award Runner-up, DCSR Paper Award. Mai 2019. URL : Paper=https://mdsattacks.com/files/ridl.pdf%20Slides=https://mdsattacks.com/slides/slides.html%20Web=https://mdsattacks.com%20Code=https://github.com/vusec/ridl%20Press=http://mdsattacks.com (page 42).

[231] Michael SCHWARZ, Moritz LIPP, Daniel MOGHIMI, Jo VAN BULCK, Julian STECKLINA, Thomas PRESCHER et Daniel GRUSS. « ZombieLoad : Cross-Privilege-Boundary Data Sampling ». In : *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS '19. London, United Kingdom : Association for Computing Machinery, 2019, p. 753-768. ISBN : 9781450367479. DOI : 10.1145/3319535.3354252. URL : https://doi.org/10.1145/3319535.3354252 (page 42).

[232] Hany RAGAB, Alyssa MILBURN, Kaveh RAZAVI, Herbert BOS et Cristiano GIUFFRIDA. « CrossTalk : Speculative Data Leaks Across Cores Are Real ». In : *S&P*. Intel Bounty Reward. Mai 2021. URL : Paper=https://download.vusec.net/papers/crosstalk_sp21.pdf%20Web=https://www.vusec.net/projects/crosstalk%20Code=https://github.com/vusec/ridl%20Press=https://bit.ly/3frdRuV (page 42).

[233] Daniel MOGHIMI, Moritz LIPP, Berk SUNAR et Michael SCHWARZ. « Medusa : Microarchitectural Data Leakage via Automated Attack Synthesis ». In : *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, août 2020, p. 1427-1444. ISBN : 978-1-939133-17-5. URL : https://www.usenix.org/conference/usenixsecurity20/presentation/moghimi-medusa (page 42).

[234] J. VAN BULCK, D. MOGHIMI, M. SCHWARZ, M. LIPPI, M. MINKIN, D. GENKIN, Y. YAROM, B. SUNAR, D. GRUSS et F. PIESSENS. « LVI : Hijacking Transient Execution through Microarchitectural Load Value Injection ». In : *2020 IEEE Symposium on Security and Privacy (SP)*. 2020, p. 54-72. DOI : 10.1109/SP40000.2020.00089 (page 42).

[235] Adrian TANG, Simha SETHUMADHAVAN et Salvatore STOLFO. « CLKSCREW : Exposing the Perils of Security-Oblivious Energy Management ». In : *USENIX Security 17*. 2017 (pages 43, 66).

[236] Pengfei QIU, Dongsheng WANG, Yongqiang LYU et Gang QU. « VoltJockey : Breaching TrustZone by Software-Controlled Voltage Manipulation over Multi-Core Frequencies ». In : *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS '19. London, United Kingdom : Association for Computing Machinery, 2019, p. 195-209. ISBN : 9781450367479. DOI : 10.1145/3319535.3354201. URL : https://doi.org/10.1145/3319535.3354201 (pages 43, 66).

[237] Pengfei QIU, Dongsheng WANG, Yongqiang LYU et Gang QU. « VoltJockey : Breaking SGX by Software-Controlled Voltage-Induced Hardware Faults ». In : *Asian Hardware Oriented Security and Trust Symposium, AsianHOST 2019, Xi'an, China, December 16-17, 2019*. IEEE, 2019, p. 1-6. DOI : 10.1109/AsianHOST47458.2019.9006701. URL : https://doi.org/10.1109/AsianHOST47458.2019.9006701 (page 43).

[238] Kit MURDOCK, David OSWALD, Flavio D. GARCIA, Jo VAN BULCK, Daniel GRUSS et Frank PIESSENS. « Plundervolt : Software-based Fault Injection Attacks against Intel SGX ». In : *41st IEEE Symposium on Security and Privacy (S&P'20)*. 2020 (pages 43, 66).

[239] Zijo KENJAR, Tommaso FRASSETTO, David GENS, Michael FRANZ et Ahmad-Reza SADEGHI. « V0LTpwn : Attacking x86 Processor Integrity from Software ». In : *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, août 2020, p. 1445-1461. ISBN : 978-1-939133-17-5. URL : https://www.usenix.org/conference/usenixsecurity20/presentation/kenjar (pages 44, 66).

[240] Yoongu KIM, Ross DALY, Jeremie KIM, Chris FALLIN, Ji Hye LEE, Donghyuk LEE, Chris WILKERSON, Konrad LAI et Onur MUTLU. « Flipping Bits in Memory without Accessing Them : An Experimental Study of DRAM Disturbance Errors ». In : *Proceeding of the 41st Annual International Symposium on Computer Architecuture*. ISCA '14. Minneapolis, Minnesota, USA : IEEE Press, 2014, p. 361-372. ISBN : 9781479943944 (page 44).

[241] Mark SEABORN et Thomas DULLIEN. *Project Zero : Exploiting the DRAM rowhammer bug to gain kernel privileges*. Google. 2015. URL : https://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html (page 44).

[242] Daniel GRUSS, Clémentine MAURICE et Stefan MANGARD. « Rowhammer.js : A Remote Software-Induced Fault Attack in JavaScript ». In : *DIMVA'16*. 2016 (page 44).

[243] E. BOSMAN, K. RAZAVI, H. BOS et C. GIUFFRIDA. « Dedup Est Machina : Memory Deduplication as an Advanced Exploitation Vector ». In : *2016 IEEE Symposium on Security and Privacy (SP)*. 2016, p. 987-1004. DOI : 10.1109/SP.2016.63 (page 44).

[244] Kaveh RAZAVI, Ben GRAS, Erik BOSMAN, Bart PRENEEL, Cristiano GIUFFRIDA et Herbert BOS. « Flip Feng Shui : Hammering a Needle in the Software Stack ». In : *USENIX Security*. Pwnie Award Nomination for Best Cryptographic Attack. Août 2016. URL : Paper=https://download.vusec.net/papers/flip-feng-shui_sec16.pdf%20Slides=https://vusec.net/wp-content/uploads/2016/

06/presentation.pdf%20Web=https://www.vusec.net/projects/flip-feng-shui%20Press=https://goo.gl/owOb2m (page 44).

[245] Yuan Xiao, Xiaokuan Zhang, Yinqian Zhang et Radu Teodorescu. « One Bit Flips, One Cloud Flops : Cross-VM Row Hammer Attacks and Privilege Escalation ». In : *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX : USENIX Association, août 2016, p. 19-35. ISBN : 978-1-931971-32-4. URL : https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/xiao (page 44).

[246] Victor van der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clementine Maurice, Giovanni Vigna, Herbert Bos, Kaveh Razavi et Cristiano Giuffrida. « Drammer : Deterministic Rowhammer Attacks on Mobile Platforms ». In : *CCS*. Pwnie Award for Best Privilege Escalation Bug, Android Security Reward, CSAW Best Paper Award, DCSR Paper Award. Oct. 2016. URL : Paper=https://vvdveen.com/publications/drammer.pdf%20Web=https://www.vusec.net/projects/drammer%20Code=https://github.com/vusec/drammer%20Press=https://goo.gl/y0lZ3l (page 44).

[247] D. Gruss, M. Lipp, M. Schwarz, D. Genkin, J. Juffinger, S. O'Connell, W. Schoechl et Y. Yarom. « Another Flip in the Wall of Rowhammer Defenses ». In : *2018 IEEE Symposium on Security and Privacy (SP)*. 2018, p. 245-261. DOI : 10.1109/SP.2018.00031 (page 45).

[248] L. Cojocar, J. Kim, M. Patel, L. Tsai, S. Saroiu, A. Wolman et O. Mutlu. « Are We Susceptible to Rowhammer ? An End-to-End Methodology for Cloud Providers ». In : *2020 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA : IEEE Computer Society, mai 2020, p. 712-728. DOI : 10.1109/SP40000.2020.00085. URL : https://doi.ieeecomputersociety.org/10.1109/SP40000.2020.00085 (page 45).

[249] A. Kwong, D. Genkin, D. Gruss et Y. Yarom. « RAMBleed : Reading Bits in Memory Without Accessing Them ». In : *2020 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA : IEEE Computer Society, mai 2020, p. 695-711. DOI : 10.1109/SP40000.2020.00020. URL : https://doi.ieeecomputersociety.org/10.1109/SP40000.2020.00020 (page 45).

[250] Nico Weichbrodt, Anil Kurmus, Peter Pietzuch et Rüdiger Kapitza. « AsyncShock : Exploiting Synchronisation Bugs in Intel SGX Enclaves ». In : *Computer Security – ESORICS 2016*. Sous la dir. d'Ioannis Askoxylakis, Sotiris Ioannidis, Sokratis Katsikas et Catherine Meadows. Cham : Springer International Publishing, 2016, p. 440-457 (page 45).

[251] Jaehyuk Lee, Jinsoo Jang, Yeongjin Jang, Nohyun Kwak, Yeseul Choi, Changho Choi, Taesoo Kim, Marcus Peinado et Brent ByungHoon Kang. « Hacking in Darkness : Return-oriented Programming against Secure Enclaves ». In : *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC : USENIX Association, août 2017, p. 523-539. ISBN : 978-1-931971-40-9. URL : https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/lee-jaehyuk (page 46).

[252] Andrea Biondo, Mauro Conti, Lucas Davi, Tommaso Frassetto et Ahmad-Reza Sadeghi. « The Guard's Dilemma : Efficient Code-Reuse Attacks Against Intel SGX ». In : *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD : USENIX Association, août 2018, p. 1213-1227. ISBN : 978-1-

939133-04-5. URL : https://www.usenix.org/conference/usenixsecurity18/presentation/biondo (page 46).

[253] Tobias CLOOSTERS, Michael RODLER et Lucas DAVI. « TeeRex : Discovery and Exploitation of Memory Corruption Vulnerabilities in SGX Enclaves ». In : *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, août 2020, p. 841-858. ISBN : 978-1-939133-17-5. URL : https://www.usenix.org/conference/usenixsecurity20/presentation/cloosters (page 46).

[254] Jo VAN BULCK, David OSWALD, Eduard MARIN, Abdulla ALDOSERI, Flavio D. GARCIA et Frank PIESSENS. « A Tale of Two Worlds : Assessing the Vulnerability of Enclave Shielding Runtimes ». In : *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS '19. London, United Kingdom : Association for Computing Machinery, 2019, p. 1741-1758. ISBN : 9781450367479. DOI : 10.1145/3319535.3363206. URL : https://doi.org/10.1145/3319535.3363206 (pages 46, 67).

[255] Aravind MACHIRY, Eric GUSTAFSON, Chad SPENSKY, Christopher SALLS, Nick STEPHENS, Ruoyu WANG, Antonio BIANCHI, Yung Ryn CHOE, Christopher KRUEGEL et Giovanni VIGNA. « BOOMERANG : Exploiting the Semantic Gap in Trusted Execution Environments ». In : *Proceedings of the Network and Distributed System Security Symposium*. 2017 (page 46).

[256] Darius SUCIU, Stephen MCLAUGHLIN, Laurent SIMON et Radu SION. « Horizontal Privilege Escalation in Trusted Applications ». In : *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, août 2020. URL : https://www.usenix.org/conference/usenixsecurity20/presentation/suciu (page 46).

[257] Di SHEN. *Defeating Samsung KNOX with zero privilege*. URL : https://www.blackhat.com/docs/us-17/thursday/us-17-Shen-Defeating-Samsung-KNOX-With-Zero-Privilege-wp.pdf (page 46).

[258] *Breaking Samsung's ARM Trustzone*. Quarkslab, 2019. URL : https://i.blackhat.com/USA-19/Thursday/us-19-Peterlin-Breaking-Samsungs-ARM-TrustZone.pdf (page 47).

[259] Jie WANG, Kun SUN, Lingguang LEI, Shengye WAN, Yuewu WANG et Jiwu JING. « Cache-in-the-Middle (CITM) Attacks : Manipulating Sensitive Data in Isolated Execution Environments ». In : *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. CCS '20. Virtual Event, USA : Association for Computing Machinery, 2020, p. 1001-1015. ISBN : 9781450370899. DOI : 10.1145/3372297.3417886. URL : https://doi.org/10.1145/3372297.3417886 (page 47).

[260] Yuanzhong XU, Weidong CUI et Marcus PEINADO. « Controlled-Channel Attacks : Deterministic Side Channels for Untrusted Operating Systems ». In : *Proceedings of the 2015 IEEE Symposium on Security and Privacy*. SP '15. USA : IEEE Computer Society, 2015, p. 640-656. ISBN : 9781467369497. DOI : 10.1109/SP.2015.45. URL : https://doi.org/10.1109/SP.2015.45 (page 47).

[261] Ahmad MOGHIMI, Gorka IRAZOQUI et Thomas EISENBARTH. « CacheZoom : How SGX Amplifies The Power of Cache Attacks ». In : *CoRR* abs/1703.06986 (2017). arXiv : 1703.06986. URL : http://arxiv.org/abs/1703.06986 (page 47).

[262] Jo Van BULCK, Nico WEICHBRODT, Rüdiger KAPITZA, Frank PIESSENS et Raoul STRACKX. « Telling Your Secrets without Page Faults : Stealthy Page Table-Based Attacks on Enclaved Execution ». In : *26th USENIX Security Symposium (USE-*

NIX Security 17). Vancouver, BC : USENIX Association, août 2017, p. 1041-1056. ISBN : 978-1-931971-40-9. URL : https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/van-bulck (page 47).

[263] Wenhao WANG, Guoxing CHEN, Xiaorui PAN, Yinqian ZHANG, XiaoFeng WANG, Vincent BINDSCHAEDLER, Haixu TANG et Carl A. GUNTER. « Leaky Cauldron on the Dark Land : Understanding Memory Side-Channel Hazards in SGX ». In : *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS '17. Dallas, Texas, USA : Association for Computing Machinery, 2017, p. 2421-2434. ISBN : 9781450349468. DOI : 10.1145/3133956.3134038. URL : https://doi.org/10.1145/3133956.3134038 (page 47).

[264] Daniel MOGHIMI, Jo Van BULCK, Nadia HENINGER, Frank PIESSENS et Berk SUNAR. « CopyCat : Controlled Instruction-Level Attacks on Enclaves ». In : *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, août 2020, p. 469-486. ISBN : 978-1-939133-17-5. URL : https://www.usenix.org/conference/usenixsecurity20/presentation/moghimi-copycat (page 47).

[265] Sangho LEE, Ming-Wei SHIH, Prasun GERA, Taesoo KIM, Hyesoon KIM et Marcus PEINADO. « Inferring Fine-grained Control Flow Inside SGX Enclaves with Branch Shadowing ». In : *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC : USENIX Association, août 2017, p. 557-574. ISBN : 978-1-931971-40-9. URL : https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/lee-sangho (page 48).

[266] Jo VAN BULCK, Frank PIESSENS et Raoul STRACKX. « Nemesis : Studying Microarchitectural Timing Leaks in Rudimentary CPU Interrupt Logic ». In : *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. CCS '18. Toronto, Canada : Association for Computing Machinery, 2018, p. 178-195. ISBN : 9781450356930. DOI : 10.1145/3243734.3243822. URL : https://doi.org/10.1145/3243734.3243822 (page 48).

[267] Wubing WANG, Yinqian ZHANG et Zhiqiang LIN. « Time and Order : Towards Automatically Identifying Side-Channel Vulnerabilities in Enclave Binaries ». In : *22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019)*. Chaoyang District, Beijing : USENIX Association, sept. 2019, p. 443-457. ISBN : 978-1-939133-07-6. URL : https://www.usenix.org/conference/raid2019/presentation/wang-wubing (page 48).

[268] Jo Van BULCK, Marina MINKIN, Ofir WEISSE, Daniel GENKIN, Baris KASIKCI, Frank PIESSENS, Mark SILBERSTEIN, Thomas F. WENISCH, Yuval YAROM et Raoul STRACKX. « Foreshadow : Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution ». In : *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD : USENIX Association, août 2018, 991-1008. ISBN : 978-1-939133-04-5. URL : https://www.usenix.org/conference/usenixsecurity18/presentation/bulck (page 48).

[269] G. CHEN, S. CHEN, Y. XIAO, Y. ZHANG, Z. LIN et T. H. LAI. « SgxPectre : Stealing Intel Secrets from SGX Enclaves Via Speculative Execution ». In : *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. 2019, p. 142-157. DOI : 10.1109/EuroSP.2019.00020 (page 48).

[270] Ben LAPID et Avishai WOOL. « Navigating the Samsung TrustZone and Cache-Attacks on the Keymaster Trustlet ». In : *Computer Security*. Sous la dir. de Javier LOPEZ, Jianying ZHOU et Miguel SORIANO. Cham : Springer International Publishing, 2018, p. 175-196. ISBN : 978-3-319-99073-6 (page 48).

[271] Haehyun Cho, Penghui Zhang, Donguk Kim, Jinbum Park, Choong-Hoon Lee, Ziming Zhao, Adam Doupé et Gail-Joon Ahn. « Prime+Count : Novel Cross-World Covert Channels on ARM TrustZone ». In : *Proceedings of the 34th Annual Computer Security Applications Conference*. ACSAC '18. San Juan, PR, USA : Association for Computing Machinery, 2018, p. 441-452. isbn : 9781450365697. doi : 10.1145/3274694.3274704. url : https://doi.org/10.1145/3274694.3274704 (page 48).

[272] Tianwei Zhang, Yinqian Zhang et Ruby Lee. « Analyzing Cache Side Channels Using Deep Neural Networks ». In : *Annual Computer Security Applications Conference*. 2018 (page 49).

[273] Andreas Abel et Jan Reineke. « uops.info : Characterizing Latency, Throughput, and Port Usage of Instructions on Intel Microarchitectures ». In : *ASPLOS*. ASPLOS '19. Providence, RI, USA : ACM, 2019, p. 673-686. isbn : 978-1-4503-6240-5. doi : 10.1145/3297858.3304062. url : http://doi.acm.org/10.1145/3297858.3304062 (page 49).

[274] Yuan Xiao, Yinqian Zhang et Radu Teodorescu. « SPEECHMINER : A Framework for Investigating and Measuring Speculative Execution Vulnerabilities ». In : *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020*. The Internet Society, 2020. url : https://www.ndss-symposium.org/ndss-paper/speechminer-a-framework-for-investigating-and-measuring-speculative-execution-vulnerabilities/ (page 49).

[275] A. Abel et J. Reineke. « Reverse engineering of cache replacement policies in Intel microprocessors and their evaluation ». In : *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 2014, p. 141-142. doi : 10.1109/ISPASS.2014.6844475 (page 49).

[276] Clémentine Maurice, Nicolas Le Scouarnec, Christoph Neumann, Olivier Heen et Aurélien Francillon. « Reverse Engineering Intel Last-Level Cache Complex Addressing Using Performance Counters ». In : *RAID'15*. 2015 (page 49).

[277] Marc Green, Leandro Rodrigues-Lima, Andreas Zankl, Gorka Irazoqui, Johann Heyszl et Thomas Eisenbarth. « AutoLock : Why Cache Attacks on ARM Are Harder Than You Think ». In : *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC : USENIX Association, août 2017, p. 1075-1091. isbn : 978-1-931971-40-9. url : https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/green (page 49).

[278] Youngjoo Shin, Hyung Chan Kim, Dokeun Kwon, Ji Hoon Jeong et Junbeom Hur. « Unveiling Hardware-Based Data Prefetcher, a Hidden Source of Information Leakage ». In : *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. CCS '18. Toronto, Canada : Association for Computing Machinery, 2018, p. 131-145. isbn : 9781450356930. doi : 10.1145/3243734.3243736. url : https://doi.org/10.1145/3243734.3243736 (page 50).

[279] Wei Song et Peng Liu. « Dynamically Finding Minimal Eviction Sets Can Be Quicker Than You Think for Side-Channel Attacks against the LLC ». In : *22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019)*. Chaoyang District, Beijing : USENIX Association, sept. 2019, p. 427-442. isbn : 978-1-939133-07-6. url : https://www.usenix.org/conference/raid2019/presentation/song (page 50).

[280] M. Kurth, B. Gras, D. Andriesse, C. Giuffrida, H. Bos et K. Razavi. « NetCAT : Practical Cache Attacks from the Network ». In : *2020 IEEE Symposium on Security and Privacy (SP)*. 2020, p. 20-38. DOI : 10.1109/SP40000.2020.00082 (page 50).

[281] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz et Stefan Mangard. « DRAMA : Exploiting DRAM Addressing for Cross-CPU Attacks ». In : *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX : USENIX Association, août 2016, p. 565-581. ISBN : 978-1-931971-32-4. URL : https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/pessl (page 50).

[282] Andrei Tatar, Cristiano Giuffrida, Herbert Bos et Kaveh Razavi. « Defeating Software Mitigations against Rowhammer : A Surgical Precision Hammer ». In : *RAID*. Best Paper Award. Sept. 2018. URL : Paper=https://download.vusec.net/papers/hammertime_raid18.pdf%20Code=https://github.com/vusec/hammertime (page 50).

[283] L. Cojocar, K. Razavi, C. Giuffrida et H. Bos. « Exploiting Correcting Codes : On the Effectiveness of ECC Memory Against Rowhammer Attacks ». In : *2019 IEEE Symposium on Security and Privacy (SP)*. 2019, p. 55-71. DOI : 10.1109/SP.2019.00089 (page 51).

[284] P. Frigo, E. Vannacc, H. Hassan, V. der Veen, O. Mutlu, C. Giuffrida, H. Bos et K. Razavi. « TRRespass : Exploiting the Many Sides of Target Row Refresh ». In : *2020 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA : IEEE Computer Society, mai 2020, p. 747-762. DOI : 10.1109/SP40000.2020.00090. URL : https://doi.ieeecomputersociety.org/10.1109/SP40000.2020.00090 (page 51).

[285] Saad Islam, Ahmad Moghimi, Ida Bruhns, Moritz Krebbel, Berk Gulmezoglu, Thomas Eisenbarth et Berk Sunar. « SPOILER : Speculative Load Hazards Boost Rowhammer and Cache Attacks ». In : *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA : USENIX Association, août 2019, p. 621-637. ISBN : 978-1-939133-06-9. URL : https://www.usenix.org/conference/usenixsecurity19/presentation/islam (page 51).

[286] Philipp Koppe, Benjamin Kollenda, Marc Fyrbiak, Christian Kison, Robert Gawlik, Christof Paar et Thorsten Holz. « Reverse Engineering x86 Processor Microcode ». In : *USENIX Security'17*. 2017 (page 51).

[287] Benjamin Kollenda, Philipp Koppe, Marc Fyrbiak, Christian Kison, Christof Paar et Thorsten Holz. « An Exploratory Analysis of Microcode as a Building Block for System Defenses ». In : CCS '18. Toronto, Canada : Association for Computing Machinery, 2018, p. 1649-1666. ISBN : 9781450356930. DOI : 10.1145/3243734.3243861. URL : https://doi.org/10.1145/3243734.3243861 (page 51).

[288] *chip-red-pill/IntelTXE-PoC : Intel Management Engine JTAG Proof of Concept*. Positive Technology. URL : https://github.com/chip-red-pill/IntelTXE-PoC (page 52).

[289] *chip-red-pill/crbus_scripts : IPC scripts for access to Intel CRBUS*. Positive Technology. URL : https://github.com/chip-red-pill/crbus_scripts (page 52).

[290] Chris Domas. *Breaking the x86 ISA*. 2017. URL : https://www.youtube.com/watch?v=KrksBdWcZgQ (page 52).

[291] Christopher Domas. *Hardware Backdoors in x86 CPUs*. 2018. URL : https://i.blackhat.com/us-18/Thu-August-9/us-18-Domas-God-Mode-Unlocked-Hardware-Backdoors-In-x86-CPUs-wp.pdf (page 52).

[292] Marco Oliverio, Kaveh Razavi, Herbert Bos et Cristiano Giuffrida. « Secure Page Fusion with VUsion : Https ://Www.Vusec.Net/Projects/VUsion ». In : *Proceedings of the 26th Symposium on Operating Systems Principles*. SOSP '17. Shanghai, China : Association for Computing Machinery, 2017, p. 531-545. ISBN : 9781450350853. DOI : 10.1145/3132747.3132781. URL : https://doi.org/10.1145/3132747.3132781 (page 52).

[293] Michael Schwarz, Moritz Lipp et Daniel Gruss. « JavaScript Zero : Real JavaScript and Zero Side-Channel Attacks ». English. In : *Network and Distributed System Security Symposium 2018*. Fév. 2018, p. 15 (page 52).

[294] Gorka Irazoqui, Thomas Eisenbarth et Berk Sunar. « MASCAT : Preventing Microarchitectural Attacks Before Distribution ». In : *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy, CODASPY 2018, Tempe, AZ, USA, March 19-21, 2018*. Sous la dir. de Ziming Zhao, Gail-Joon Ahn, Ram Krishnan et Gabriel Ghinita. ACM, 2018, p. 377-388. DOI : 10.1145/3176258.3176316. URL : https://doi.org/10.1145/3176258.3176316 (page 52).

[295] Qian Ge, Yuval Yarom et Gernot Heiser. « No Security Without Time Protection : We Need a New Hardware-Software Contract ». In : *Proceedings of the 9th Asia-Pacific Workshop on Systems, APSys 2018, Jeju Island, Republic of Korea, August 27-28, 2018*. ACM, 2018, 1 :1-1 :9. DOI : 10.1145/3265723.3265724. URL : https://doi.org/10.1145/3265723.3265724 (pages 53, 63, 64).

[296] Qian Ge, Yuval Yarom, Tom Chothia et Gernot Heiser. « Time Protection : The Missing OS Abstraction ». In : *Proceedings of the Fourteenth EuroSys Conference 2019, Dresden, Germany, March 25-28, 2019*. Sous la dir. de George Candea, Robbert van Renesse et Christof Fetzer. ACM, 2019, 1 :1-1 :17. DOI : 10.1145/3302424.3303976. URL : https://doi.org/10.1145/3302424.3303976 (page 53).

[297] Marco Guarnieri, Boris Köpf, Jan Reineke et Pepe Vila. « Hardware-Software Contracts for Secure Speculation ». In : *2021 IEEE Symposium on Security and Privacy, SP 2021, Proceedings, San Francisco, California, USA*. Mai 2021. URL : https://arxiv.org/abs/2006.03841 (page 53).

[298] Qian Ge, Yuval Yarom, David Cock et Gernot Heiser. « A survey of microarchitectural timing attacks and countermeasures on contemporary hardware ». In : *J. Cryptogr. Eng.* 8.1 (2018), p. 1-27. DOI : 10.1007/s13389-016-0141-6. URL : https://doi.org/10.1007/s13389-016-0141-6 (page 53).

[299] Maria Mushtaq, Muhammad Asim Mukhtar, Vianney Lapotre, Muhammad khurram Bhatti et Guy Gogniat. « Winter is here ! A decade of cache-based side-channel attacks, detection & mitigation for RSA ». In : *Information Systems* (2020). DOI : 10.1016/j.is.2020.101524. URL : https://hal.archives-ouvertes.fr/hal-02537540 (page 53).

[300] Goran Doychev, Dominik Feld, Boris Kopf, Laurent Mauborgne et Jan Reineke. « CacheAudit : A Tool for the Static Analysis of Cache Side Channels ». In : *22nd USENIX Security Symposium (USENIX Security 13)*. Washington, D.C. : USENIX Association, août 2013, p. 431-446. ISBN : 978-1-931971-03-4.

URL : https://www.usenix.org/conference/usenixsecurity13/technical-sessions/paper/doychev (page 53).

[301] Gilles BARTHE, Gustavo BETARTE, Juan CAMPO, Carlos LUNA et David PICHARDIE. « System-Level Non-Interference for Constant-Time Cryptography ». In : *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. CCS '14. Scottsdale, Arizona, USA : Association for Computing Machinery, 2014, p. 1267-1279. ISBN : 9781450329576. DOI : 10.1145/2660267.2660283. URL : https://doi.org/10.1145/2660267.2660283 (page 54).

[302] *agl/ctgrind : Checking that functions are constant time with Valgrind*. URL : https://github.com/agl/ctgrind (page 54).

[303] Jose Bacelar ALMEIDA, Manuel BARBOSA, Gilles BARTHE, François DUPRESSOIR et Michael EMMI. « Verifying Constant-Time Implementations ». In : *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX : USENIX Association, août 2016, p. 53-70. ISBN : 978-1-931971-32-4. URL : https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/almeida (page 54).

[304] Shuai WANG, Pei WANG, Xiao LIU, Danfeng ZHANG et Dinghao WU. « CacheD : Identifying Cache-Based Timing Channels in Production Software ». In : *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC : USENIX Association, août 2017, p. 235-252. ISBN : 978-1-931971-40-9. URL : https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/wang-shuai (page 54).

[305] Yuan XIAO, Mengyuan LI, Sanchuan CHEN et Yinqian ZHANG. « STACCO : Differentially Analyzing Side-Channel Traces for Detecting SSL/TLS Vulnerabilities in Secure Enclaves ». In : *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS '17. Dallas, Texas, USA : Association for Computing Machinery, 2017, p. 859-874. ISBN : 9781450349468. DOI : 10.1145/3133956.3134016. URL : https://doi.org/10.1145/3133956.3134016 (page 54).

[306] Samuel WEISER, Andreas ZANKL, Raphael SPREITZER, Katja MILLER, Stefan MANGARD et Georg SIGL. « DATA – Differential Address Trace Analysis : Finding Address-based Side-Channels in Binaries ». In : *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD : USENIX Association, août 2018, p. 603-620. ISBN : 978-1-939133-04-5. URL : https://www.usenix.org/conference/usenixsecurity18/presentation/weiser (page 54).

[307] Jan WICHELMANN, Ahmad MOGHIMI, Thomas EISENBARTH et Berk SUNAR. « MicroWalk : A Framework for Finding Side Channels in Binaries ». In : *Proceedings of the 34th Annual Computer Security Applications Conference, ACSAC 2018, San Juan, PR, USA, December 03-07, 2018*. ACM, 2018, p. 161-173. DOI : 10.1145/3274694.3274741. URL : https://doi.org/10.1145/3274694.3274741 (page 54).

[308] Wubing WANG, Yinqian ZHANG et Zhiqiang LIN. « Time and Order : Towards Automatically Identifying Side-Channel Vulnerabilities in Enclave Binaries ». In : *22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019)*. Chaoyang District, Beijing : USENIX Association, sept. 2019, p. 443-457. ISBN : 978-1-939133-07-6. URL : https://www.usenix.org/conference/raid2019/presentation/wang-wubing (page 54).

[309] Stephen Crane, Andrei Homescu, Stefan Brunthaler, Per Larsen et Michael Franz. « Thwarting Cache Side-Channel Attacks Through Dynamic Software Diversity ». In : *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*. The Internet Society, 2015. URL : https://www.ndss-symposium.org/ndss2015/thwarting-cache-side-channel-attacks-through-dynamic-software-diversity (page 54).

[310] Ashay Rane, Calvin Lin et Mohit Tiwari. « Raccoon : Closing Digital Side-Channels through Obfuscated Execution ». In : *24th USENIX Security Symposium (USENIX Security 15)*. Washington, D.C. : USENIX Association, août 2015, p. 431-446. ISBN : 978-1-939133-11-3. URL : https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/rane (page 54).

[311] Ferdinand Brasser, Srdjan Capkun, Alexandra Dmitrienko, Tommaso Frassetto, Kari Kostiainen et Ahmad-Reza Sadeghi. « DR.SGX : Automated and Adjustable Side-Channel Protection for SGX Using Data Location Randomization ». In : *Proceedings of the 35th Annual Computer Security Applications Conference*. ACSAC '19. San Juan, Puerto Rico, USA : Association for Computing Machinery, 2019, p. 788-800. ISBN : 9781450376280. DOI : 10.1145/3359789.3359809. URL : https://doi.org/10.1145/3359789.3359809 (page 55).

[312] Taesoo Kim, Marcus Peinado et Gloria Mainar-Ruiz. « STEALTHMEM : System-Level Protection Against Cache-Based Side Channel Attacks in the Cloud ». In : *21st USENIX Security Symposium (USENIX Security 12)*. Bellevue, WA : USENIX Association, août 2012, p. 189-204. URL : https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/kim (page 55).

[313] Ziqiao Zhou, Michael K. Reiter et Yinqian Zhang. « A Software Approach to Defeating Side Channels in Last-Level Caches ». In : *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. CCS'16. Vienna, Austria : ACM, 2016, p. 871-882. DOI : 10.1145/2976749.2978324 (page 55).

[314] *Improving Real-Time Performance by UtilizingCache Allocation Technology*. Intel Corporation. 2015. URL : https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/cache-allocation-technology-white-paper.pdf (page 55).

[315] F. Liu, Q. Ge, Y. Yarom, F. Mckeen, C. Rozas, G. Heiser et R. B. Lee. « CATalyst : Defeating last-level cache side channel attacks in cloud computing ». In : *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 2016, p. 406-418. DOI : 10.1109/HPCA.2016.7446082 (page 55).

[316] Vladimir Kiriansky, Ilia Lebedev, Saman Amarasinghe, Srinivas Devadas et Joel Emer. « DAWG : A Defense against Cache Timing Attacks in Speculative Execution Processors ». In : MICRO-51. Fukuoka, Japan : IEEE Press, 2018, p. 974-987. ISBN : 9781538662403. DOI : 10.1109/MICRO.2018.00083. URL : https://doi.org/10.1109/MICRO.2018.00083 (page 55).

[317] Ghada Dessouky, Tommaso Frassetto et Ahmad-Reza Sadeghi. « HybCache : Hybrid Side-Channel-Resilient Caches for Trusted Execution Environments ». In : *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, août 2020, p. 451-468. ISBN : 978-1-939133-17-5. URL : https:

//www.usenix.org/conference/usenixsecurity20/presentation/dessouky (page 55).

[318] Mario Werner, Thomas Unterluggauer, Lukas Giner, Michael Schwarz, Daniel Gruss et Stefan Mangard. « ScatterCache : Thwarting Cache Attacks via Cache Set Randomization ». In : *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA : USENIX Association, août 2019, p. 675-692. isbn : 978-1-939133-06-9. url : https://www.usenix.org/conference/usenixsecurity19/presentation/werner (page 55).

[319] Daniel Gruss, Julian Lettner, Felix Schuster, Olya Ohrimenko, Istvan Haller et Manuel Costa. « Strong and Efficient Cache Side-Channel Protection using Hardware Transactional Memory ». In : *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC : USENIX Association, août 2017, p. 217-233. isbn : 978-1-931971-40-9. url : https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/gruss (page 56).

[320] Daniel Gruss, Moritz Lipp, Michael Schwarz, Richard Fellner, Clémentine Maurice et Stefan Mangard. « KASLR is Dead : Long Live KASLR ». In : *Engineering Secure Software and Systems*. Sous la dir. d'Eric Bodden, Mathias Payer et Elias Athanasopoulos. Cham : Springer International Publishing, 2017, p. 161-176. isbn : 978-3-319-62105-0 (page 56).

[321] David Gens, Orlando Arias, Dean Sullivan, Christopher Liebchen, Yier Jin et Ahmad-Reza Sadeghi. « LAZARUS : Practical Side-Channel Resilient Kernel-Space Randomization ». In : *Research in Attacks, Intrusions, and Defenses*. Sous la dir. de Marc Dacier, Michael Bailey, Michalis Polychronakis et Manos Antonakakis. Cham : Springer International Publishing, 2017, p. 238-258. isbn : 978-3-319-66332-6 (page 56).

[322] The kernel development community. *Page Table Isolation (PTI) — The Linux Kernel documentation*. url : https://www.kernel.org/doc/html/latest/x86/pti.html (page 56).

[323] Claudio Canella, Michael Schwarz, Martin Haubenwallner, Martin Schwarzl et Daniel Gruss. « KASLR : Break It, Fix It, Repeat ». In : ASIA CCS '20. Taipei, Taiwan : Association for Computing Machinery, 2020, p. 481-493. isbn : 9781450367509. doi : 10.1145/3320269.3384747. url : https://dl.acm.org/doi/abs/10.1145/3320269.3384747 (page 56).

[324] Yinqian Zhang et Michael K. Reiter. « DüPpel : Retrofitting Commodity Operating Systems to Mitigate Cache Side Channels in the Cloud ». In : *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*. CCS '13. Berlin, Germany : Association for Computing Machinery, 2013, p. 827-838. isbn : 9781450324779. doi : 10.1145/2508859.2516741. url : https://doi.org/10.1145/2508859.2516741 (page 56).

[325] Venkatanathan Varadarajan, Thomas Ristenpart et Michael Swift. « Scheduler-based Defenses against Cross-VM Side-channels ». In : *23rd USENIX Security Symposium (USENIX Security 14)*. San Diego, CA : USENIX Association, août 2014, p. 687-702. isbn : 978-1-931971-15-7. url : https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/varadarajan (page 56).

[326] Guoxing Chen, Wenhao Wang, Tianyu Chen, Sanchuan Chen, Yinqian Zhang, XiaoFeng Wang, Ten-Hwang Lai et Dongdai Lin. « Racing in Hyperspace : Closing Hyper-Threading Side Channels on SGX with Contrived Data Races ». In :

39th IEEE Symposium on Security and Privacy. IEEE Computer Society, 2018, p. 388-404 (page 56).

[327] Tianwei ZHANG, Yinqian ZHANG et Ruby B. LEE. « CloudRadar : A Real-Time Side-Channel Attack Detection System in Clouds ». In : *19th International Symposium on Research in Attacks, Intrusions, and Defenses*. RAID'16. Paris, France : Springer International Publishing, 2016, p. 118-140. DOI : 10.1007/978-3-319-45719-2_6 (page 57).

[328] Samira BRIONGOS, Gorka IRAZOQUI, Pedro MALAGÓN et Thomas EISENBARTH. « CacheShield : Detecting Cache Attacks through Self-Observation ». In : *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy, CODASPY 2018, Tempe, AZ, USA, March 19-21, 2018*. Sous la dir. de Ziming ZHAO, Gail-Joon AHN, Ram KRISHNAN et Gabriel GHINITA. ACM, 2018, p. 224-235. DOI : 10.1145/3176258.3176320. URL : https://doi.org/10.1145/3176258.3176320 (page 57).

[329] Maria MUSHTAQ, Ayaz AKRAM, Muhammad Khurram BHATTI, Maham CHAUDHRY, Vianney LAPOTRE et Guy GOGNIAT. « NIGHTs-WATCH : A Cache-Based Side-Channel Intrusion Detector Using Hardware Performance Counters ». In : *Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy*. HASP '18. Los Angeles, California : Association for Computing Machinery, 2018. ISBN : 9781450365000. DOI : 10.1145/3214292.3214293. URL : https://doi.org/10.1145/3214292.3214293 (page 57).

[330] Roberto GUANCIALE, Musard BALLIU et Mads DAM. « InSpectre : Breaking and Fixing Microarchitectural Vulnerabilities by Formal Analysis ». In : *CCS '20 : 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*. Sous la dir. de Jay LIGATTI, Xinming OU, Jonathan KATZ et Giovanni VIGNA. ACM, 2020, p. 1853-1869. DOI : 10.1145/3372297.3417246. URL : https://doi.org/10.1145/3372297.3417246 (page 57).

[331] Guanhua WANG, Sudipta CHATTOPADHYAY, Arnab Kumar BISWAS, Tulika MITRA et Abhik ROYCHOUDHURY. « KLEESpectre : Detecting Information Leakage through Speculative Cache Attacks via Symbolic Execution ». In : 29.3 (juin 2020). ISSN : 1049-331X. DOI : 10.1145/3385897. URL : https://doi.org/10.1145/3385897 (page 57).

[332] Oleksii OLEKSENKO, Bohdan TRACH, Mark SILBERSTEIN et Christof FETZER. « SpecFuzz : Bringing Spectre-type vulnerabilities to the surface ». In : *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, août 2020, p. 1481-1498. ISBN : 978-1-939133-17-5. URL : https://www.usenix.org/conference/usenixsecurity20/presentation/oleksenko (page 57).

[333] Andrea MAMBRETTI, Matthias NEUGSCHWANDTNER, Alessandro SORNIOTTI, Engin KIRDA, William ROBERTSON et Anil KURMUS. « Speculator : A Tool to Analyze Speculative Execution Attacks and Mitigations ». In : *Proceedings of the 35th Annual Computer Security Applications Conference*. ACSAC '19. San Juan, Puerto Rico, USA : Association for Computing Machinery, 2019, p. 747-761. ISBN : 9781450376280. DOI : 10.1145/3359789.3359837. URL : https://doi.org/10.1145/3359789.3359837 (page 58).

[334] *SOFTWARE SECURITY GUIDANCE*. Intel Corporation. URL : https://software.intel.com/security-software-guidance/ (page 58).

[335] *Vulnerability of Speculative Processors to Cache Timing Side-Channel Mechanism*. Arm Limited. 2020. URL : https://developer.arm.com/support/arm-security-updates/speculative-processor-vulnerability (page 58).

[336] *Software techniques for managing speculation on AMD processors*. Advanced Micro Devices, Inc. 2020. URL : http://developer.amd.com/wp-content/resources/Managing-Speculation-on-AMD-Processors.pdf (page 58).

[337] The kernel development COMMUNITY. *Spectre Side Channels — The Linux Kernel documentation*. URL : https://www.kernel.org/doc/html/latest/admin-guide/hw-vuln/spectre.html (page 58).

[338] Andrew PARDOE. *Spectre mitigations in MSVC*. Microsoft. 2018. URL : https://devblogs.microsoft.com/cppblog/spectre-mitigations-in-msvc/ (page 58).

[339] Guanhua WANG, Sudipta CHATTOPADHYAY, Ivan GOTOVCHITS, Tulika MITRA et Abhik ROYCHOUDHURY. *oo7 : Low-overhead Defense against Spectre Attacks via Program Analysis*. 2019. arXiv : 1807.05843 [cs.CR]. URL : https://arxiv.org/abs/1807.05843 (page 58).

[340] M. GUARNIERI, B. KÖPF, J. F. MORALES, J. REINEKE et A. SÁNCHEZ. « Spectector : Principled Detection of Speculative Information Flows ». In : *2020 IEEE Symposium on Security and Privacy (SP)*. 2020, p. 1-19. DOI : 10.1109/SP40000.2020.00011 (page 59).

[341] Paul TURNER. *Retpoline : a software construct for preventing branch-target-injection*. Google. URL : https://support.google.com/faqs/answer/7625886 (page 59).

[342] Michael SCHWARZ, Moritz LIPP, Claudio Alberto CANELLA, Robert SCHILLING, Florian KARGL et Daniel GRUSS. « ConTExT : A Generic Approach for Mitigating Spectre ». In : *Network and Distributed System Security Symposium 2020*. Fév. 2020. DOI : 10.14722/ndss.2020.24271 (page 59).

[343] Charles REIS, Alexander MOSHCHUK et Nasko OSKOV. « Site Isolation : Process Separation for Web Sites within the Browser ». In : *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA : USENIX Association, août 2019, p. 1661-1678. ISBN : 978-1-939133-06-9. URL : https://www.usenix.org/conference/usenixsecurity19/presentation/reis (page 59).

[344] E. KORUYEH, S. Haji Amin SHIRAZI, K. N. KHASAWNEH, C. SONG et N. ABU-GHAZALEH. « SpecCFI : Mitigating Spectre Attacks using CFI Informed Speculation ». In : *2020 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA : IEEE Computer Society, mai 2020, p. 39-53. DOI : 10.1109/SP40000.2020.00033. URL : https://doi.ieeecomputersociety.org/10.1109/SP40000.2020.00033 (page 60).

[345] *What Spectre and Meltdown Mean For WebKit*. 2018. URL : What%20Spectre%20and%20Meltdown%20Mean%20For%20WebKit (page 60).

[346] Chandler CARRUTH. *Speculative Load Hardening*. URL : https://llvm.org/docs/SpeculativeLoadHardening.html (page 60).

[347] Khaled N. KHASAWNEH, Esmaeil Mohammadian KORUYEH, Chengyu SONG, Dmitry EVTYUSHKIN, Dmitry PONOMAREV et Nael ABU-GHAZALEH. « SafeSpec : Banishing the Spectre of a Meltdown with Leakage-Free Speculation ». In : *Proceedings of the 56th Annual Design Automation Conference 2019*. DAC '19. Las Vegas, NV, USA : Association for Computing Machinery, 2019. ISBN : 9781450367257. DOI : 10.1145/3316781.3317903. URL : https://doi.org/10.1145/3316781.3317903 (page 60).

[348] Mengjia YAN, Jiho CHOI, Dimitrios SKARLATOS, Adam MORRISON, Christopher W. FLETCHER et Josep TORRELLAS. « InvisiSpec : Making Speculative Execution Invisible in the Cache Hierarchy ». In : MICRO-51. Fukuoka, Japan : IEEE Press, 2018, p. 428-441. ISBN : 9781538662403. DOI : 10.1109/MICRO.2018.00042. URL : https://doi.org/10.1109/MICRO.2018.00042 (page 60).

[349] Zelalem Birhanu AWEKE, Salessawi Ferede YITBAREK, Rui QIAO, Reetuparna DAS, Matthew HICKS, Yossi OREN et Todd AUSTIN. « ANVIL : Software-Based Protection Against Next-Generation Rowhammer Attacks ». In : *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '16. Atlanta, Georgia, USA : Association for Computing Machinery, 2016, p. 743-755. ISBN : 9781450340915. DOI : 10.1145/2872362.2872390. URL : https://doi.org/10.1145/2872362.2872390 (page 60).

[350] Ferdinand BRASSER, Lucas DAVI, David GENS, Christopher LIEBCHEN et Ahmad-Reza SADEGHI. « CAn't Touch This : Software-only Mitigation against Rowhammer Attacks targeting Kernel Memory ». In : *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC : USENIX Association, août 2017, p. 117-130. ISBN : 978-1-931971-40-9. URL : https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/brasser (page 60).

[351] Radhesh Krishnan KONOTH, Marco OLIVERIO, Andrei TATAR, Dennis ANDRIESSE, Herbert BOS, Cristiano GIUFFRIDA et Kaveh RAZAVI. « ZebRAM : Comprehensive and Compatible Software Protection Against Rowhammer Attacks ». In : *OSDI*. Oct. 2018. URL : Paper=https://download.vusec.net/papers/zebram_osdi18.pdf%20Code=https://github.com/vusec/zebram (page 60).

[352] Victor van der VEEN, Martina LINDORFER, Yanick FRATANTONIO, Harikrishnan PADMANABHA PILLAI, Giovanni VIGNA, Christopher KRUEGEL, Herbert BOS et Kaveh RAZAVI. « GuardION : Practical Mitigation of DMA-based Rowhammer Attacks on ARM ». In : *DIMVA*. Pwnie Award Nomination for Best Privilege Escalation Bug. Juin 2018. URL : Paper=https://vvdveen.com/publications/dimva2018.pdf%20Web=https://rampageattack.com%20Code=https://github.com/vusec/guardion%20Press=https://bit.ly/2H6QhqX (page 60).

[353] Z. ZHANG, Z. ZHAN, D. BALASUBRAMANIAN, B. LI, P. VOLGYESI et X. KOUTSOUKOS. « Leveraging EM Side-Channel Information to Detect Rowhammer Attacks ». In : *2020 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA : IEEE Computer Society, mai 2020, p. 729-746. DOI : 10.1109/SP40000.2020.00060. URL : https://doi.ieeecomputersociety.org/10.1109/SP40000.2020.00060 (page 60).

[354] *Morello – Arm Developer*. 2020. URL : https://developer.arm.com/architectures/cpu-architecture/a-profile/morello (page 68).

[355] Nicolas JOLY, Saif ELSHEREI et Saar AMAR. *Security Analysis of CHERI ISA*. Microsoft Security Response Center. 2020. URL : https://msrc-blog.microsoft.com/2020/10/14/security-analysis-of-cheri-isa/ (page 68).

[356] Robert N. M. WATSON. *CHERI : The Digital Security by Design (DSbD) Initiative*. 2019. URL : https://www.cl.cam.ac.uk/research/security/ctsrd/cheri/cheri-morello.html (page 68).

[357] *Hafnium – opensource.google*. Google. URL : https://opensource.google/projects/hafnium (page 68).

[358] *Morello Platform Open Source Software - Morello Project*. Linaro. 2020. URL : https://www.morello-project.org/ (page 68).