

Titre: Implémentation d'une architecture de routeur sécurisée à l'aide du
Title: Trusted Computing

Auteur: Colin Stéphenne
Author:

Date: 2023

Type: Mémoire ou thèse / Dissertation or Thesis

Référence: Stéphenne, C. (2023). Implémentation d'une architecture de routeur sécurisée à
Citation: l'aide du Trusted Computing [Mémoire de maîtrise, Polytechnique Montréal].
PolyPublie. <https://publications.polymtl.ca/53413/>

 **Document en libre accès dans PolyPublie**
Open Access document in PolyPublie

URL de PolyPublie: <https://publications.polymtl.ca/53413/>
PolyPublie URL:

**Directeurs de
recherche:** Gabriela Nicolescu, & Frédéric Cuppens
Advisors:

Programme: Génie informatique
Program:

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

**Implémentation d'une architecture de routeur sécurisée à l'aide du Trusted
Computing**

COLIN STÉPHENNE

Département de génie informatique et génie logiciel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie informatique

Mai 2023

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Ce mémoire intitulé :

**Implémentation d'une architecture de routeur sécurisée à l'aide du Trusted
Computing**

présenté par **Colin STÉPHENNE**

en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

Alejandro QUINTERO, président

Gabriela NICOLESCU, membre et directrice de recherche

Frédéric CUPPENS, membre et codirecteur de recherche

Michel DAGENAIS, membre

DÉDICACE

*Parfois une tâche peut sembler impossible
Mais la persévérance vient à bout de n'importe quel défi
Et lorsque tu auras réussi
Tu t'exclameras*

Habile !

REMERCIEMENTS

Je souhaite tout d'abord remercier le professeur José Fernandez de m'avoir accepté comme étudiant à la maîtrise et de m'avoir transmis sa vision de ce projet. Je le remercie, ainsi que Militza Jean, de m'avoir donné l'opportunité d'évaluer la sécurité d'un routeur commercial, ce qui m'a fait découvrir une nouvelle passion.

Je souhaite ensuite remercier ma directrice, Gabriela Nicolescu, de m'avoir accepté dans son laboratoire et guidé jusqu'à l'achèvement de ce mémoire, ainsi que Frédéric Cuppens d'avoir accepté d'être mon codirecteur. Un grand merci aussi à Jean-Yves Ouattara pour son temps et ses judicieux conseils.

Je souhaite remercier l'équipe de SentinelC qui m'a permis d'utiliser leur projet comme base pour le mien. Merci à Cédric Veilleux d'avoir pris le temps de m'en expliquer les rouages et de répondre à mes questions. Je remercie Desjardins qui finance SentinelC tout en voulant rendre le code public et accessible gratuitement.

J'aimerais remercier tous les membres de ma famille qui n'ont jamais cessé de croire en moi, ainsi que mes amis qui m'ont changé les idées quand c'était nécessaire.

Finalement, une mention spéciale à ma merveilleuse copine qui m'a encouragé et soutenu pendant les moments difficiles. Sans toi, ce mémoire n'aurait peut-être jamais été achevé.

RÉSUMÉ

La sécurité des routeurs domestiques est en retard par rapport à celle des ordinateurs tout-usage. Le marché concurrentiel incite les manufacturiers à offrir des produits développés rapidement au détriment de leur sécurité. Pourtant, les routeurs occupent une place centrale dans les réseaux domestiques et la plupart sont connectés directement à l'internet, ce qui en fait des cibles de choix pour des acteurs malveillants. Cela est d'autant plus dangereux que la clientèle typique de ces appareils n'a pas beaucoup de connaissances en informatique. De nombreux foyers se retrouvent donc avec un routeur vulnérable sans même le savoir.

La sécurité logicielle des routeurs domestiques peut être améliorée par l'utilisation d'une suite de logiciels développés en suivant des pratiques sécuritaires. Cependant, il est impossible de créer une suite logicielle qui ne contiendra jamais la moindre vulnérabilité. La sécurité renforcée au niveau matériel est donc le parfait complément, permettant de limiter les capacités d'un adversaire en cas de brèche logicielle. Le Trusted Computing est une technologie émergente s'appuyant sur des primitives de sécurité matérielle pour créer un environnement isolé et digne de confiance. Sa récente popularité est due entre autres au produit ARM TrustZone qui le rend accessible sur tous les nouveaux processeurs ARM. Ce mémoire s'intéresse à la possibilité de créer une architecture de routeur domestique sécurisé en profitant des dernières avancées du Trusted Computing. Nos principales contributions sont la revue des technologies les plus sécuritaires et les mieux adaptées aux routeurs, aux niveaux logiciel et matériel, ainsi que le port de plusieurs d'entre elles sur la plateforme ESPRESSObin. Nous avons développé un prototype de routeur domestique qui réussit à protéger un jeton d'authentification même dans le cas où un attaquant parvient à y exécuter du code en tant qu'utilisateur racine. Le jeton est chiffré avec une clé contenue dans un environnement isolé et peut servir à authentifier l'appareil auprès d'un serveur distant.

Le coût en performance de notre solution est minimal au point d'être imperceptible par l'utilisateur. Le code est disponible publiquement et il est organisé en couches Yocto faciles à réutiliser par d'autres développeurs. Nous espérons ainsi rendre les avantages du Trusted Computing plus accessibles à tous les manufacturiers de routeurs afin de mieux protéger les consommateurs.

ABSTRACT

Routers are a central component of every network. However, commercial home routers are currently lacking in security compared to general purpose computers or cloud appliances. The fast-paced market encourages manufacturers to release low cost and rapidly developed models at the deficit of their security. These models make their way into the homes of unaware consumers who put their whole network at risk without knowing it. Routers are a prime target for malicious actors since all the network packets of connected devices pass through them. Moreover, they are generally connected directly to the Internet and their owners might not have any notion in computer science. Therefore, home routers in their default configuration should be more secure than they currently are.

Software security of routers can be improved by hardening exposed services and reducing their number. However, it's impossible to completely remove all software vulnerabilities in such complex and constantly evolving devices. Also, most common isolation techniques like virtualization and containerization don't apply well to routers because of their small computing power and the inherent interdependence between their components. Therefore, secure hardware components can be a valuable complement to software security. When implemented correctly, they can greatly reduce the impact of software vulnerabilities. Trusted Computing is an emergent technology aimed at securing sensitive data and processes through the use of hardware enforced isolation. Its popularization is partly due to the release of ARM TrustZone, which makes Trusted Computing primitives available on all devices equipped with the latest ARM processors. This thesis examines the possibility of creating a secure router architecture using Trusted Computing. Part of our contributions are a survey of secure operating systems based on ARM TrustZone and compatible hardware platforms with similar features to commercial routers. We also created a prototype of a secure home router by using a secure software stack and protecting its authentication token with hardware enforced isolation.

The prototype can successfully protect the token, even when a malicious actor manages to execute arbitrary code as the root user. Moreover, the performance cost is unnoticeable to the user. All the code created is publicly available and organized in Yocto layers which can easily be reused by other developers. Our goal is to make the secure primitives of Trusted Computing more accessible to all router manufacturers.

TABLE DES MATIÈRES

DÉDICACE	iii
REMERCIEMENTS	iv
RÉSUMÉ	v
ABSTRACT	vi
TABLE DES MATIÈRES	vii
LISTE DES TABLEAUX	x
LISTE DES FIGURES	xi
LISTE DES SIGLES ET ABRÉVIATIONS	xii
LISTE DES ANNEXES	xv
CHAPITRE 1 INTRODUCTION	1
1.1 Les routeurs sous attaque	1
1.2 Une cible de choix	1
1.3 Sécurité des routeurs domestiques	2
1.3.1 Sécurité logicielle	4
1.3.2 Sécurité matérielle	6
1.4 Le Trusted Computing	8
1.4.1 Démarrage de confiance	8
1.4.2 Trusted Execution Environment	9
1.5 Objectifs	9
1.6 Plan du mémoire	11
CHAPITRE 2 ÉTAT DE LA SÉCURITÉ DES ROUTEURS	12
2.1 Modèle de menace	12
2.2 État de l'art sur la sécurité logicielle	16
2.3 État de l'art sur la sécurité matérielle	19
2.4 Étude de cas sur l'exploitation d'un routeur	20
2.4.1 Méthodologie	21

2.4.2	Reconnaissance	22
2.4.3	Extraction du micrologiciel	26
2.4.4	Rétro-ingénierie du micrologiciel	30
2.4.5	Résultats et discussion	32
2.4.6	Recommandations de sécurité	38
2.4.7	Conclusion	40
CHAPITRE 3	LE TRUSTED COMPUTING	42
3.1	Racine de confiance	42
3.1.1	Trusted Platform Module	43
3.1.2	Fonction physique inclonable	44
3.1.3	Autres racines de confiance	44
3.2	Architectures matérielles	45
3.2.1	Intel SGX	46
3.2.2	ARM TrustZone	46
3.3	Démarrage sécurisé	47
3.4	Démarrage de confiance	47
3.5	ARM Trusted Firmware	48
3.6	Trusted Execution Environment	49
3.7	Attestation	50
CHAPITRE 4	IMPLÉMENTATION D'UN PROTOTYPE	53
4.1	Élaboration d'une architecture	53
4.2	Choix des technologies	54
4.2.1	SentinelC	55
4.2.2	Trusted Execution Environment	59
4.2.3	Plateforme de développement	63
4.3	Développement sur le ESPRESSObin	64
4.3.1	Yocto	64
4.3.2	Variables d'environnement de U-Boot	66
4.3.3	Récupération par UART	67
4.4	Intégration de OP-TEE	67
4.4.1	Modification du code source	69
4.4.2	Arborescence des périphériques	70
4.4.3	Intégration de meta-arm	71
4.5	Intégration de SentinelC	72
4.6	Application de Confiance	73

4.6.1	Application cliente	75
4.6.2	Application sécurisée	76
4.6.3	Intégration au client SentinelC	76
4.7	Création de recettes Yocto	78
4.8	Protocole d'authentification à distance	78
4.8.1	Attestation	79
4.8.2	Protocoles de défi-réponse	80
4.8.3	Nomenclature	80
4.8.4	Échange de clé initial	81
4.8.5	Détails des requêtes	82
CHAPITRE 5	ÉVALUATION DE LA SOLUTION	85
5.1	Sécurité	85
5.1.1	Sécurité du prototype	85
5.1.2	Sécurité de la solution finale	88
5.1.3	Retour sur l'étude de cas	90
5.2	Performance	90
5.3	Mise en marché	91
5.4	Limites de la solution	92
CHAPITRE 6	CONCLUSION	93
6.1	Synthèse des travaux	93
6.1.1	Retour sur les objectifs	93
6.1.2	Contributions	94
6.2	Améliorations futures	94
RÉFÉRENCES		96
ANNEXES		106

LISTE DES TABLEAUX

Tableau 2.1	Résumé des vulnérabilités découvertes	38
Tableau 4.1	Caractéristiques des Trusted Execution Environment (TEE)	63

LISTE DES FIGURES

Figure 1.1	Attaques à partir d'un routeur compromis	2
Figure 1.2	Influences sur la sécurité d'un routeur domestique	4
Figure 2.1	Composants des routeurs et leur surface d'attaque	13
Figure 2.2	Types d'attaquants	14
Figure 2.3	Survol des étapes de la méthodologie	22
Figure 2.4	Un port visible avec ses broches identifiées	25
Figure 2.5	La badge <i>Attify</i> connectée à la puce par des crochets	28
Figure 4.1	Architecture initiale du routeur sécurisé	55
Figure 5.1	Lecture du jeton d'authentification sur la carte SD	86
Figure 5.2	Le jeton d'authentification n'est pas lisible sur la carte SD du prototype	86
Figure 5.3	Entropie du jeton chiffré	87
Figure 5.4	Récupération du jeton sur l'appareil original	87
Figure 5.5	Tentative de récupération du jeton sur le nouveau système	88

LISTE DES SIGLES ET ABRÉVIATIONS

ABI	Application Binary Interface
API	Application Programming Interface
APT	Advanced Persistent Threat
BBRAM	Battery-Backed Random Access Memory
BSP	Board Support Package
CHAP	Challenge Handshake Authentication Protocol
CVE	Common Vulnerabilities and Exposures
DDoS	Distributed Denial-of-Service
DHCP	Dynamic Host Configuration Protocol
DIP	Dual Inline Package
DNS	Domain Name System
DRM	Digital Rights Management
EL	Exception Level
ELF	Executable and Linkable Format
FACT	Firmware Analysis and Comparison Tool
FDE	Full-Disk Encryption
FIDO	Fast Identity Online
FPGA	Field-Programmable Gate Array
FTP	File Transfer Protocol
GB	Gigabyte
GCC	GNU Compiler Collection
GOT	Global Offset Table
HMAC	Hash-based Message Authentication Codes
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
HSM	Hardware Security Module
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IoT	Internet of Things
I2C	Inter-Integrated Circuit
JSON	JavaScript Object Notation
JTAG	Joint Test Action Group
LAN	Local Area Network

LZMA	Lempel-Ziv-Markov
MITM	Man-in-the-middle
NAS	Network-Attached Storage
NIP	Numéro d'Identification Personnel
NIST	National Institute of Standards and Technology
NVD	National Vulnerability Database
NVRAM	Non-Volatile Random Access Memory
OS	Operating System
PCR	Platform Configuration Register
PHP	Hypertext Preprocessor
PUF	Physical Unclonable Function
RAM	Random Access Memory
RAT	Remote Access Trojan
REST	Representational State Transfer
ROP	Return Oriented Programming
ROM	Read-Only Memory
ROTPK	Root of Trust Public Key
SDK	Software Development Kits
SFTP	SSH File Transfer Protocol
SGX	Software Guard Extension
SoC	System on a chip
SOHO	Small Office, Home Office
SPD	Secure Payload Dispatcher
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
SSH	Secure Shell
TCB	Trusted Computing Base
TCG	Trusted Computing Group
TEE	Trusted Execution Environment
TFTP	Trivial File Transfer Protocol
TLS	Transport Layer Security
TPM	Trusted Platform Module
UART	Universal Asynchronous Receiver Transmitter
UART	Universal Plug and Play
USB	Universal Serial Bus
UUID	Universally Unique Identifier

VLAN	Virtual Local Area Network
VPN	Virtual Private Network
WAN	Wide Area Network

LISTE DES ANNEXES

Annexe A	Correction de code pour ajouter le dispositif OP-TEE à l'arborescence des périphériques	106
----------	--	-----

CHAPITRE 1 INTRODUCTION

1.1 Les routeurs sous attaque

La fréquence et l'intensité des cyber attaques contre les routeurs ont décuplé durant la dernière décennie. La sécurité des routeurs a été négligée dans la fortification des défenses numériques, comme le démontre le grand nombre de vulnérabilités récemment répertoriées [1]. Contrairement aux ordinateurs personnels, la plupart des routeurs sur le marché reçoivent moins d'une mise à jour par année [2]. Il n'est donc pas surprenant de les voir par millions dans des *botnets* comme Mirai [3] ou infectés par des virus comme BotenaGo [4]. Une fois enrôlés, ils participeront ensuite à des attaques distribuées comme des Distributed Denial-of-Service (DDoS), à l'insu de leur propriétaire. Ils peuvent aussi espionner le trafic réseau pour voler les mots de passe des autres utilisateurs du réseau [5]. Malgré leur puissance de calcul souvent modeste, certains acteurs ont aussi utilisé des routeurs vulnérables pour miner des cryptomonnaies [6]. Il est même arrivé que des acteurs sophistiqués, ou Advanced Persistent Threat (APT), développent des outils pour espionner leurs cibles à travers leur routeur [7].

1.2 Une cible de choix

Les routeurs devraient pourtant bénéficier d'une attention accrue, due à leur point central dans les réseaux commerciaux et résidentiels. Non seulement ils sont souvent connectés directement à l'internet et sont allumés en permanence, mais tous les paquets du réseau auquel ils appartiennent passent par ceux-ci. Cette caractéristique fait en sorte qu'un routeur compromis peut être utilisé comme point de pivot pour compromettre les autres appareils qui y sont connectés. Bien qu'il existe des technologies de chiffrement très efficaces pour assurer la protection des données, ce ne sont pas tous les protocoles qui en tirent parti. Ces protocoles sont particulièrement à risque d'être exploités si le routeur est compromis. Par exemple, de nombreux sites web n'utilisent pas de chiffrement Transport Layer Security (TLS). Toutes les données qui leur sont envoyées ou reçues peuvent donc être lues et même modifiées, ce qui peut mener au vol d'informations sensibles. De plus, les téléchargements faits à travers ce canal non sécurisé peuvent être injectés avec des virus de façon totalement imperceptible [8]. De nombreux protocoles comme File Transfer Protocol (FTP) ou Telnet ne sont pas non plus chiffrés et sont vulnérables à des attaques du même type. Toutefois, même les protocoles qui semblent sécuritaires peuvent être attaqués si un acteur malveillant est en contrôle du routeur. Par exemple, les protocoles Secure Shell (SSH) et SSH File Transfer Protocol (SFTP)

peuvent être victimes d’une attaque Man-in-the-middle (MITM), qui donne un accès complet à la communication à l’attaquant. Certaines campagnes ont aussi utilisé des techniques de pollution de Domain Name System (DNS) pour leurrer les utilisateurs du réseau vers de faux sites [9]. Quelques unes de ces attaques sont illustrées à la figure 1.1.

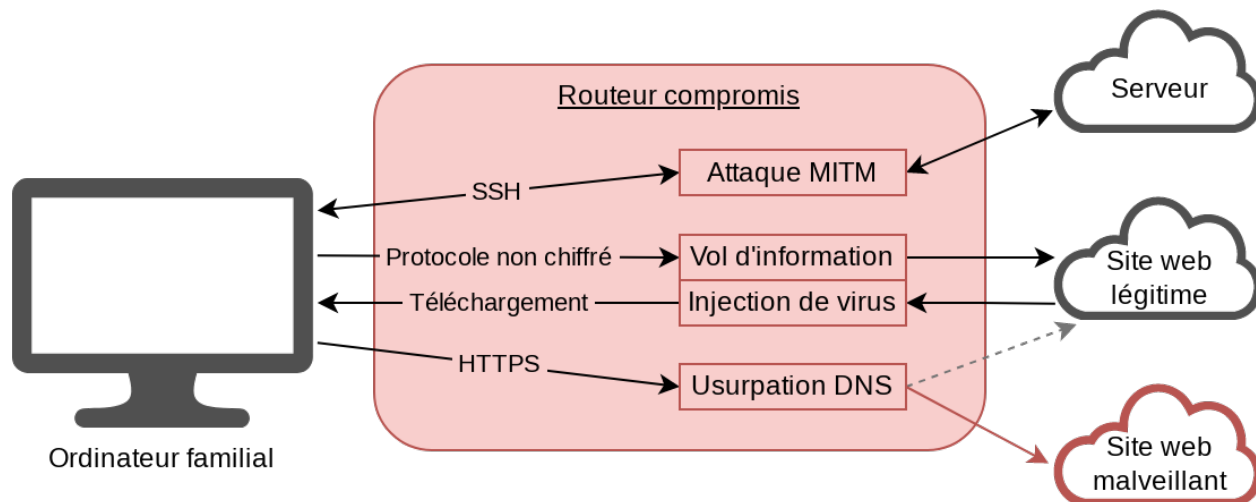


FIGURE 1.1 Attaques à partir d’un routeur compromis

Il va sans dire que les entreprises déploient des efforts considérables pour sécuriser leurs réseaux, car elles ont une forte motivation monétaire à le faire, surtout avec l’augmentation des attaques de rançongiciel [10]. Leurs réseaux sont donc souvent munis de systèmes de détection ou de remédiation d’intrusions, et les routeurs dédiés aux entreprises bénéficient des avancées en matière de performance et de sécurité. Cependant, les routeurs domestiques n’ont pas suivi la cadence et leur sécurité laisse très souvent à désirer. Les routeurs domestiques désignent les routeurs qui sont destinés à être utilisés dans des maisons de particuliers. Ces routeurs sont généralement petits, peu dispendieux et mettent l’emphase sur les capacités Wifi en ayant peu de ports Ethernet. Ils peuvent parfois s’apparenter aux routeurs Small Office, Home Office (SOHO), mais possèdent moins de fonctionnalités. Les routeurs domestiques sont faits pour être faciles à installer et à utiliser, et c’est le type de routeur qui a le plus de lacunes en termes de sécurité. Le terme routeur sera utilisé tout au long de ce mémoire pour désigner les routeurs domestiques.

1.3 Sécurité des routeurs domestiques

Les routeurs domestiques ont plusieurs caractéristiques qui les distinguent des ordinateurs traditionnels. Premièrement, leur matériel est orienté vers la réseautique, ce qui se traduit par plusieurs ports réseau et de puissantes antennes. Deuxièmement, ils sont axés sur une faible

consommation d'énergie et des coûts de fabrication réduits. Contrairement aux ordinateurs tout usage, les routeurs n'ont qu'un seul rôle, qui est d'assurer la bonne circulation des paquets réseau ainsi que les quelques fonctionnalités des routeurs domestiques (contrôle parental, attribution des adresses IP, DNS dynamique, etc.). C'est pourquoi les routeurs ont souvent une puissance de calcul moindre, ainsi qu'une petite capacité de stockage et très peu de périphériques. La plupart des routeurs ont aussi un processeur ARM ou MIPS dû à leur faible coût et basse consommation d'énergie. Ces caractéristiques les apparentent plutôt aux appareils embarqués comme les appareils Internet of Things (IoT).

Sécuriser les routeurs est un grand défi, car leur surface d'attaque est très grande. Par définition, ce sont des appareils qui doivent être exposés pour bien remplir leurs fonctions. Au niveau logiciel, ils occupent une place centrale du réseau et toute vulnérabilité peut être exploitée par une multitude d'appareils connectés. D'un autre côté, les routeurs doivent être proches des utilisateurs, par exemple pour assurer une bonne couverture Wifi, ce qui les rend vulnérables à des attaques physiques.

Une grande partie de la recherche faite pour améliorer la sécurité des réseaux domestiques est dirigée vers la détection et assume que le routeur lui-même n'est pas compromis. Par exemple, [11] propose l'intégration de multiples fonctionnalités de sécurité dans l'appareil de routage principal d'un réseau domestique. Cependant, ces stratégies tombent à plat dès que le routeur est infecté, car les attaquants peuvent ensuite modifier les données recueillies à leur guise et ainsi masquer leur présence.

Le marché extrêmement compétitif est un grand incitatif pour les manufacturiers à créer des routeurs domestiques à faible coût, même si cela vient au détriment de leur sécurité. Les modèles de routeur les plus sécuritaires s'accompagnent toujours d'un prix qui est nettement supérieur à leurs contreparties moins sécurisées. Les entreprises sont souvent prêtes à investir dans une infrastructure sécuritaire, mais les particuliers beaucoup moins. De plus, les consommateurs de routeurs domestiques ne sont pas toujours conscients des risques liés à la sécurité de leur réseau ou outillés pour bien configurer leur appareil. Ces facteurs contribuent encore à ce jour au grand nombre de routeurs domestiques vulnérables à travers le globe. Une étude réalisée en 2009 a trouvé que des milliers de routeurs à travers le monde étaient configurés de manière dangereuse, soit en exposant leur interface web sur l'internet et en ayant encore le mot de passe par défaut [12]. De ces appareils, 41.62% étaient des routeurs domestiques, contre seulement 2.46% de routeurs d'entreprise, ce qui démontre bien l'ampleur du problème. Malheureusement, il n'existe actuellement pas de cadre permettant de créer des appareils à moindre coût ayant néanmoins une sécurité minimale.

Les routeurs domestiques sont des appareils complexes, à la fois sur les plans logiciels et

matériels. De plus, de nombreux acteurs jouent un rôle dans cet objectif, comme le montre la Figure 1.2. Il est donc difficile d'arriver à un produit final sécuritaire, car chacun de ces acteurs peut introduire une faiblesse potentielle, même à son insu.

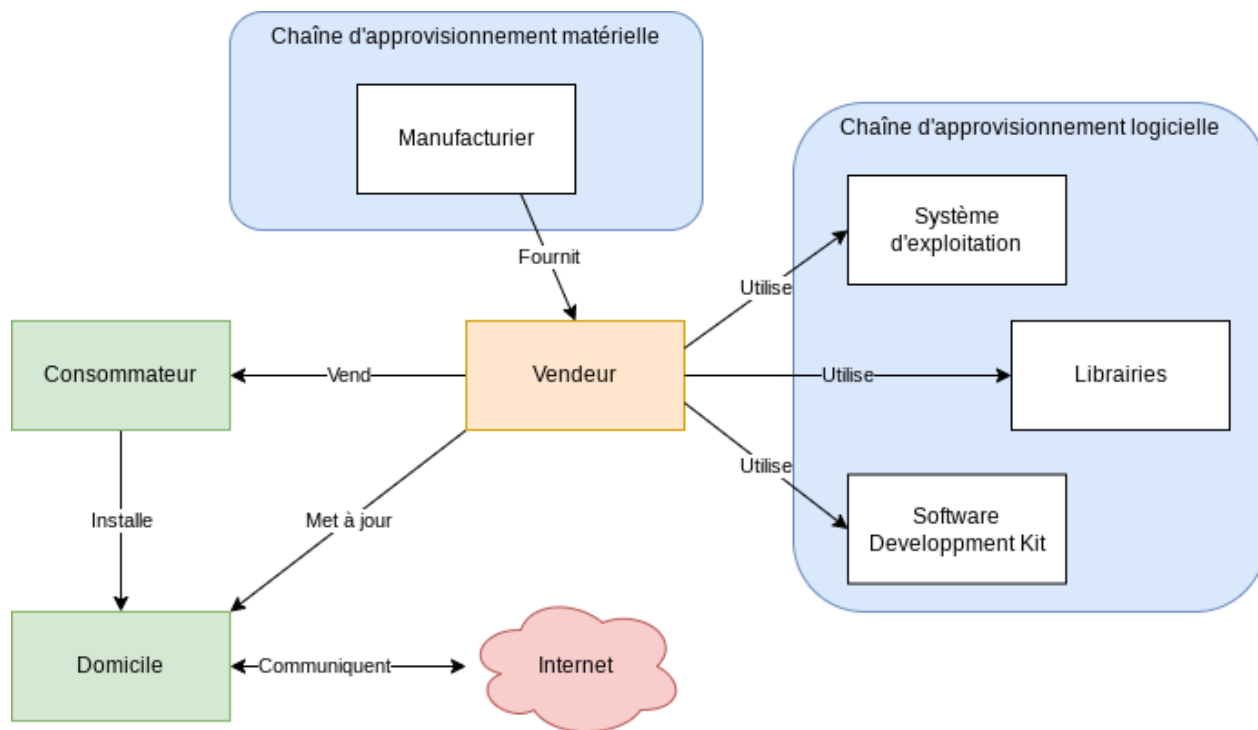


FIGURE 1.2 Influences sur la sécurité d'un routeur domestique

1.3.1 Sécurité logicielle

Lorsqu'on observe les routeurs domestiques sur le marché, on constate que la sécurité logicielle est souvent inadéquate. La majorité des vulnérabilités utilisées par les acteurs malveillants pour prendre le contrôle des routeurs sont trouvées dans les interfaces web permettant la configuration de l'appareil par l'utilisateur. Cependant, d'autres services traditionnellement retrouvés sur des routeurs, comme Universal Plug and Play (UPnP), peuvent être victimes de failles sévères [13]. Parfois, la gravité de la vulnérabilité est amplifiée par la négligence de l'utilisateur qui expose son interface de configuration à l'internet sans être conscient des risques. La vulnérabilité peut rester cachée durant des décennies, laissant des millions d'appareils vulnérables chez les consommateurs [14]. Cependant, il arrive aussi que le vendeur expose lui-même les fonctionnalités vulnérables dans la configuration par défaut, comme dans le cas de certains routeurs Asus en 2014 [15]. D'un point de vue de sécurité, il est impossible de contrôler ce que les utilisateurs font de leur appareil, mais il est primordial que les

manufacturiers prennent les mesures nécessaires pour vendre des produits sécuritaires.

Afin de simplifier le développement du micrologiciel et des applications sur les routeurs, des Software Development Kits (SDK) ont vu le jour. L'utilisation des SDK aide à réduire le coût et la durée de production, mais vient avec ses propres risques. Lorsqu'une vulnérabilité est trouvée dans un SDK, tous les routeurs qui l'utilisent se trouvent dans une situation critique d'un seul coup, ce qui en décuple l'impact [16]. De plus, il y a une montée des attaques par chaîne d'approvisionnement, qui peuvent aussi affecter un grand nombre de systèmes. Elles sont très difficiles à détecter et à prévenir, en particulier lorsqu'elles affectent de grandes entités gouvernementales, comme ce fut le cas pour la faille SolarWinds [17].

En 2020, Weidenbach et vom Dorp [2] ont analysé 127 routeurs domestiques sur le marché pour en attester l'état de sécurité. Ils ont trouvé que tous les routeurs étudiés possédaient au moins une faille de sécurité critique. Certains ont des clés cryptographiques ou des mots de passe fixes directement dans leur micrologiciel, d'autres n'utilisent pas des mesures de protection bien connues sur leurs exécutables et certains n'ont pas reçu de mise à jour depuis plusieurs années. Cela montre bien l'état alarmant de la sécurité logicielle des routeurs domestiques sur le marché.

Il est aussi notoire que certains vendeurs de routeurs peuvent mettre des mois à réparer les failles décelées dans leurs appareils. D'autres ne répondent tout simplement pas quand des chercheurs leur font part de vulnérabilités présentes et ne semblent pas se soucier le moins du monde de la sécurité de leurs produits [18, 19]. Cela est vraisemblablement attribuable à des raisons économiques et permet à des entités malveillantes de continuer à exploiter ces failles au détriment des consommateurs.

Heureusement, il y a une tendance à la hausse des bonnes pratiques de sécurité mises en place par les manufacturiers. Par exemple, la majorité des routeurs vendus aujourd'hui sur le marché n'exposent pas l'interface web de configuration à l'internet en entier, mais seulement au réseau local. Cela n'empêche pas des vulnérabilités de s'y trouver, mais réduit l'impact le cas échéant. Encore mieux, des solutions comme le projet SentinelC tentent de résoudre ce problème en enlevant complètement l'interface de configuration web de leurs routeurs et en la remplaçant par un serveur central depuis lequel toute une flotte d'appareils peut être configurée et surveillée [20]. Aussi, plusieurs routeurs implémentent des mises à jour automatiques du micrologiciel. Or, cela reste peu courant et insuffisant.

1.3.2 Sécurité matérielle

Les opportunités d'attaques sur le plan matériel sont beaucoup moindres que celles sur le plan logiciel par le seul fait qu'un attaquant doit physiquement avoir le routeur en sa possession pour les effectuer. Ces attaques excluent les dénis de service, car il est toujours possible de physiquement détruire ou de tout simplement débrancher le routeur malgré ses mécanismes de sécurité matérielle.

Bien que les routeurs domestiques se retrouvent généralement dans des maisons, la possibilité qu'un attaquant accède à un routeur situé par exemple dans une cour extérieure n'est pas à exclure. Aussi, plusieurs attaques matérielles ont pour objectif principal l'acquisition de données qui peuvent aider à trouver par la suite des failles logicielles qui, elles, seront exploitables à distance. Le but est donc de lire et d'exfiltrer le contenu du système de fichiers, car celui-ci renferme tous les programmes potentiellement vulnérables qui roulent sur l'appareil [21]. Par exemple, lorsque des connecteurs comme Universal Asynchronous Receiver Transmitter (UART) et Joint Test Action Group (JTAG) sont présents sur l'appareil, s'y connecter peut donner accès à des fonctionnalités cachées. Cela peut ouvrir directement une connexion en tant qu'administrateur sur l'appareil ou donner accès à l'interface de ligne de commande du micrologiciel de démarrage, entre autres. Dans le second cas, l'interface propose généralement des commandes pour lire le contenu de la mémoire où réside le système de fichiers du routeur.

Si aucun connecteur n'est disponible sur la carte mère du routeur, une autre technique consiste à se brancher directement sur ses circuits pour en exfiltrer de l'information. La connexion peut être faite sur les branches des puces de la carte, puisqu'ils sont plus faciles d'accès que les circuits imprimés. Cette technique repose sur le principe que toute la mémoire du routeur réside dans une puce et qu'elle doit physiquement se déplacer vers le processeur pour être utilisée. Il est donc possible de lire toutes les données des fichiers utilisés par l'appareil. Si ce n'est pas suffisant, il est aussi possible d'enlever la puce mémoire et de la lire selon ses spécifications. Il est extrêmement difficile de se protéger contre cette attaque. La seule défense possible est le chiffrement de toutes les données sur la puce, mais dans ce cas il faut stocker la clé quelque part en mémoire et le même problème ressurgit. La raison pour laquelle le chiffrement complet du disque, ou Full-Disk Encryption (FDE), est possible sur un ordinateur tout usage est que l'utilisateur est présent à chaque démarrage pour entrer un mot de passe, ce qui est irréaliste pour un routeur. De plus, le coût en calcul et en énergie pour chiffrer et déchiffrer toutes les données à chaque accès est déraisonnable pour un routeur.

Un autre type d'attaque matérielle s'en prend au processus de démarrage des routeurs. Comme sur les ordinateurs à usage général, il y a plusieurs étapes au démarrage avant que

le système d'exploitation principal ne prenne le contrôle. Ces étapes sont rarement sécurisées, car les fonctionnalités du système sont grandement résuées à ces stades du démarrage. Un attaquant qui parvient à prendre le contrôle du routeur peut utiliser le processus de démarrage pour se cacher et assurer sa persistance. Les logiciels malveillants dissimulés dans le micrologiciel de démarrage et le système d'exploitation se nomment *bootkits* et *rootkits*, respectivement. Ils sont extrêmement difficiles à détecter et à détruire dû aux privilèges avec lesquels ils s'exécutent. Un attaquant doit déjà posséder les privilèges administrateur sur un appareil pour y ajouter un tel implant. Cependant, un accès physique peut permettre de contourner cette restriction, car de nombreuses installations des micrologiciels de démarrage, comme *u-boot* [22], permettent d'écraser la mémoire avec de nouvelles données sans aucune vérification ni authentification. Très souvent, le micrologiciel de démarrage permet aussi de démarrer un système d'exploitation de notre choix. Une fois celui-ci démarré, il est alors trivial de lire toutes les informations sensibles de l'appareil, car les restrictions de lecture sont imposées de façon logicielle par le système d'exploitation.

Le démarrage sécurisé, ou *Secure Boot* est une technique qui tente de remédier à ce problème. À chaque étape du processus de démarrage, le programme qui s'exécute vérifie l'intégrité et l'authenticité du programme responsable de la prochaine étape avant de l'exécuter. La vérification est faite en validant la signature numérique du programme avec une clé publique asymétrique. Il est alors impossible pour un attaquant de modifier les programmes de démarrage sans avoir la clé privée servant à générer les signatures. Cependant, pour que cette technique soit robuste, il faut que plusieurs prédicats soient respectés. Tout d'abord, la clé publique et le premier programme de démarrage doivent provenir d'un tiers de confiance et doivent se situer dans une zone de mémoire qui est physiquement en lecture seule. Malheureusement, ces contraintes centralisent la confiance dans les émetteurs des clés publiques et empêchent les particuliers de personnaliser leur séquence de démarrage. Ce n'est pas non plus une solution viable pour les vendeurs de petits appareils embarqués, car ils n'ont pas toujours l'infrastructure pour gérer la distribution de clés cryptographiques. C'est pour ces raisons que les routeurs domestiques n'ont jamais de démarrage sécurisé.

Il est aisé de constater comme il est difficile de se protéger contre les attaques matérielles. Une fois l'appareil entre les mains d'un attaquant, il est presque impossible de garder des données secrètes, que ce soit une clé cryptographique, le code source d'une application ou un jeton d'authentification. La section 2 explore une étude de cas démontrant les capacités d'un accès physique lors de l'évaluation de la sécurité d'un routeur. Par leur nature autonome, les routeurs devront toujours avoir des informations sensibles qui leur servent à s'identifier auprès d'autres partis. Il serait idéal de pouvoir ajouter une couche de sécurité matérielle aux routeurs afin de protéger ces informations. Le Trusted Computing peut combler ces lacunes

grâce aux avancées en matière d'isolation matérielle.

1.4 Le Trusted Computing

Le Trusted Computing est un ensemble de principes et de techniques ayant comme objectif l'établissement d'une primitive de confiance dans un système possiblement compromis. La primitive de confiance peut prendre plusieurs formes, depuis une simple valeur d'une donnée jusqu'à l'exécution complète d'une application avec la certitude qu'elle s'exécute correctement. L'établissement de la confiance se fait de manière incrémentale. Seul un petit élément, appelé racine de confiance, ou *Root of Trust*, doit être digne de confiance au départ dans le système. Cette racine peut ensuite attester de la confiance d'une autre composante plus complexe, qui elle-même peut attester de la confiance d'une autre composante, et ainsi de suite. La racine de confiance peut prendre plusieurs formes. La forme la plus connue est le Trusted Platform Module (TPM), une petite puce dédiée aux opérations cryptographiques sécuritaires ayant été conçue par le Trusted Computing Group (TCG). Malheureusement, très peu d'appareils embarqués et de routeurs possèdent un TPM, principalement à cause de son coût.

1.4.1 Démarrage de confiance

Similairement au démarrage sécurisé, le démarrage de confiance, ou *Trusted Boot*, utilise les principes du Trusted Computing pour établir une chaîne de confiance entre les différentes étapes du processus de démarrage. Cependant, plutôt que de vérifier des signatures électroniques à l'aide d'une clé publique préinstallée sur l'appareil, le démarrage de confiance utilise la racine de confiance pour attester de l'intégrité de tous les programmes dans la chaîne. L'implémentation exacte dépend de la nature de la racine de confiance utilisée, mais les avantages sont les mêmes. Non seulement cette technique empêche la modification des images de démarrage et donc l'installation de *bootkit* et de *rootkits*, mais elle permet aussi de protéger des informations sensibles si un système d'exploitation différent est démarré. En effet, les valeurs résultants de l'attestation de confiance peuvent servir à chiffrer des données importantes, qui seront alors impossibles à déchiffrer si le système n'est pas dans le même état que lors du chiffrement. De plus, le démarrage de confiance permet aux utilisateurs de personnaliser leur séquence de démarrage, car les images n'ont pas besoin d'être signées par des clés cryptographiques.

1.4.2 Trusted Execution Environment

La fonctionnalité la plus puissante du Trusted Computing est la création d'un environnement d'exécution de confiance, ou TEE. Il s'agit d'un milieu où tous les programmes et leurs résultats sont dignes de confiance, même si l'appareil sur lequel ils s'exécutent est indigne de confiance. Un TEE utilise les propriétés des architectures de processeur modernes afin de créer un environnement isolé du reste du système. Cette isolation est construite au niveau matériel de l'appareil, ce qui signifie que même un attaquant qui possède les plus hauts privilèges sur l'appareil ne pourra pas compromettre l'intégrité du TEE. L'implémentation de ces fonctionnalités dans un routeur pourrait permettre d'isoler les programmes et données sensibles du reste des programmes et de créer ainsi un rempart matériel contre les failles logicielles qui sont omniprésentes. Une autre propriété importante d'un TEE est sa capacité à prouver son intégrité à une tierce partie, ce qui est très pertinent dans le contexte d'un routeur [23].

1.5 Objectifs

Ce projet de recherche vise à améliorer la sécurité des routeurs domestiques. Nous avons établi que les routeurs domestiques sur le marché ont de grosses lacunes en matière de sécurité, à la fois logicielle et matérielle. Nous voulons donc proposer une architecture de routeur domestique qui n'a pas de mot de passe fixe ni de matériel cryptographique privé lisible par un utilisateur. Ce routeur devrait minimiser la surface d'attaque logicielle et utiliser des mesures matérielles pour restreindre l'impact en cas de brèche. Il devrait aussi minimiser, voire annuler, l'impact résultant d'une composante infectée par la chaîne d'approvisionnement ou par un fournisseur mal intentionné. Il devrait être facile à installer, à configurer et à utiliser par un consommateur sans expérience technique. Le routeur devrait se mettre à jour automatiquement et avoir un prix abordable pour un consommateur moyen. Finalement, il devrait posséder une forme d'isolation permettant de protéger des données ou des processus sensibles. Cette isolation serait utilisée pour sécuriser un processus d'authentification ou de vérification afin que son propriétaire ait la certitude qu'il est toujours intègre.

Étant donné que tous les routeurs ont des caractéristiques et des rôles un peu différents, ce projet cherche plus particulièrement à protéger une information sensible contre les attaquants. Une information sensible est un petit morceau de données sur laquelle peut être bâtie toute une structure sécuritaire. Par exemple, une clé SSH peut servir à établir un tunnel sécurisé, tant et aussi longtemps que la clé reste secrète auprès des attaquants. De la même manière, un mot de passe peut servir à établir une session sécuritaire auprès d'un service si les attaquants

n'ont aucune manière de le dévoiler. En d'autres mots, une information sensible est la pierre angulaire dont le secret est garant de la sécurité de toute une fonctionnalité.

Ce qui rend la protection d'une information sensible aussi difficile est le contexte et la grande surface d'attaque. Du côté logiciel, il faut assumer que des attaquants peuvent trouver et exploiter une faille qui leur permettra d'élever leurs privilèges au niveau administrateur, leur donnant tous les droits sur l'appareil. Du côté matériel, l'information sensible doit rester secrète même si les attaquants utilisent des attaques physiques pour accéder à des composantes ou modifier leur comportement.

L'objectif de ce projet peut donc se traduire par la recherche d'une réponse à la question suivante :

Est-il possible d'intégrer le Trusted Computing dans l'architecture d'un routeur domestique pour protéger une information sensible contre des attaques logicielles et matérielles ?

La solution de cette question peut être trouvée en atteignant plusieurs objectifs secondaires :

1. Faire un état de la sécurité des routeurs domestiques sur le marché, des attaques les plus courantes contre ceux-ci et des solutions existantes pour en améliorer la sécurité
2. Identifier les informations les plus sensibles et les plus importantes à protéger sur un routeur domestique
3. Déterminer les fonctionnalités du Trusted Computing pouvant aider à protéger ces informations
4. Créer un prototype de routeur intégrant ces fonctionnalités du Trusted Computing afin de protéger une information sensible liée à ses fonctions
5. Tester si la solution protège bel et bien l'information sensible contre les attaques décrites plus tôt
6. Explorer des pistes pour une éventuelle mise en marché

Afin d'atteindre ces objectifs, nous proposons les contributions suivantes :

1. Compte rendu de l'état actuel de la sécurité des routeurs sur le marché
2. Étude de cas sur l'exploitation d'un routeur par l'accès physique
3. Modification des couches Yocto existantes pour supporter OP-TEE sur la plateforme ESPRESSObin
4. Création d'une preuve de concept d'Application de Confiance pour protéger un jeton d'authentification
5. Création d'une couche Yocto intégrant OP-TEE, SentinelC OS et l'Application de Confiance sur la plateforme ESPRESSObin

1.6 Plan du mémoire

Le chapitre 2 explore plus en détail les principes du Trusted Computing et leur implémentation dans les systèmes modernes. Le chapitre 3 décrit l'architecture et les caractéristiques de sécurité du routeur SentinelC et de son système d'exploitation personnalisé SentinelC OS. Puis, le chapitre 4 explique les étapes de l'intégration du Trusted Computing et de SentinelC sur la plateforme de développement choisie, ainsi que l'implémentation de l'application sécurisée. Le chapitre 5 passe en revue la sécurité du prototype, sa performance et son potentiel de mise en marché et pose les limites de la solution. Finalement, le chapitre 6 revient sur les objectifs de recherche et propose des travaux futurs.

CHAPITRE 2 ÉTAT DE LA SÉCURITÉ DES ROUTEURS

2.1 Modèle de menace

Afin de classer les différentes attaques qui peuvent menacer un routeur domestique, il convient de décrire ses composantes et leurs rôles dans une utilisation normale. Tout d’abord, nous caractérisons les routeurs domestiques comme de petits ordinateurs embarqués servant principalement à assurer le bon routage des paquets réseau. Ils sont donc dotés d’au moins 2 périphériques réseau, c’est-à-dire des ports Ethernet ou des antennes Wifi. Puisque ce sont des appareils embarqués, ils n’ont pas les mêmes composantes matérielles qu’un ordinateur standard tout usage. Notamment, leur processeur fait généralement partie d’un système sur puce, ou System on a chip (SoC). Les routeurs standards ont aussi une puce de mémoire vive de type Random Access Memory (RAM) et une puce de mémoire flash servant à entreposer les données persistantes comme le micrologiciel de démarrage, le système d’exploitation et le système de fichiers. Sur la carte mère, on peut souvent trouver des ports sériels comme le UART, qui servent à diagnostiquer des problèmes lors du développement de la carte. Les routeurs ont parfois des ports Universal Serial Bus (USB) qui peuvent servir à héberger un serveur de stockage réseau, ou Network-Attached Storage (NAS). Les principales composantes des routeurs, ainsi que les vulnérabilités pouvant les affecter, sont illustrées sous forme de couches dans la Figure 2.1.

Les composantes des routeurs peuvent être séparées en couches superposées appartenant au côté logiciel ou matériel de l’appareil. La première couche contient les circuits électriques imprimés sur la carte mère, qui incluent parfois des fonctionnalités de débogage comme des connecteurs UART ou JTAG. Ensuite, des puces et des périphériques sont connectés par ces circuits, constituant la deuxième couche. Ces couches forment la partie matérielle d’un routeur, sur laquelle tout le logiciel fonctionne. La première couche logicielle est le micrologiciel de démarrage, qui est en fait constitué de plusieurs stades. Le premier stade de démarrage est un micrologiciel en lecture seule, fourni par le fabricant de l’appareil, qui s’occupe d’initialiser les fonctionnalités de base avant de lire le deuxième stade depuis une mémoire externe et de l’exécuter. Le deuxième stade est plus riche en fonctionnalités et s’occupe de lire et d’exécuter le système d’exploitation. Cette architecture permet d’augmenter la flexibilité du processus de démarrage, mais laisse le deuxième stade vulnérable à des attaques de *bootkits*. Par la suite, le système d’exploitation initialise tous les périphériques et lance les services fournis par l’appareil. Certains services opèrent au niveau binaire, mais d’autres servent à router des applications de plus haut niveau, de la même manière qu’un serveur web

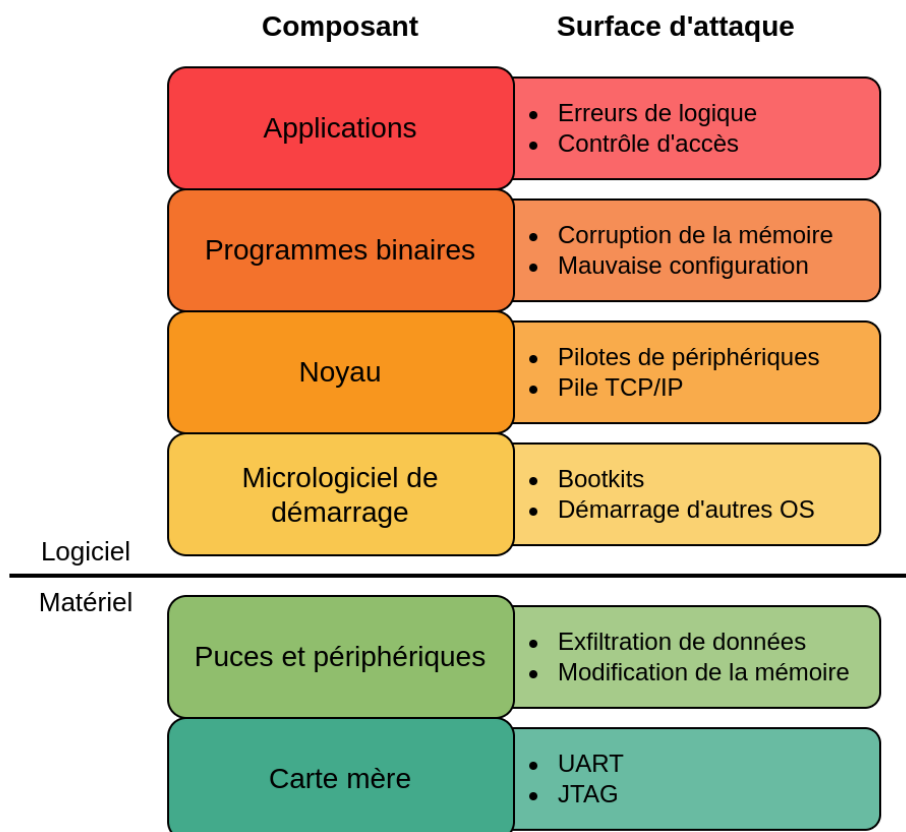


FIGURE 2.1 Composants des routeurs et leur surface d'attaque

est un programme compilé pouvant rouler une application en Hypertext Preprocessor (PHP) avec des fonctionnalités plus riches.

Les routeurs domestiques sont ceux utilisés par les particuliers dans leur maison. Leur utilisateur typique ne possède donc pas beaucoup de connaissances informatiques. C'est pourquoi ils sont conçus pour être faciles d'utilisation et facilement configurables. Très souvent, un foyer ne possède qu'un seul routeur. Ses interfaces réseau sont séparées en 2 zones distinctes : le réseau local, ou Local Area Network (LAN), et le réseau étendu, ou Wide Area Network (WAN). Le réseau local est orienté à l'intérieur du foyer et tous les appareils avec des capacités réseau y sont connectés. Le réseau étendu est normalement connecté à l'internet à travers un modem. Tout port ouvert sur l'interface du réseau étendu sera donc accessible depuis l'internet au grand complet.

Les différents acteurs qui peuvent s'en prendre à des routeurs peuvent être classés en plusieurs catégories selon leurs caractéristiques. En premier lieu, les attaquants à distance sont les plus nombreux. Ces attaquants sont physiquement très loin du routeur et n'y ont accès que par

l'internet. Leur surface d'attaque se réduit donc à l'interface de réseau étendu du routeur. En théorie, il est possible d'attaquer l'intérieur d'un réseau local depuis l'internet, par exemple en installant un Remote Access Trojan (RAT) sur un appareil à l'intérieur du foyer. Cependant, ce cas n'est pas pris en compte, car les attaquants ayant un accès au réseau local font partie d'une autre catégorie. Il y a ensuite les attaquants locaux, qui ont accès au routeur à la fois par son réseau local et son réseau étendu. Concrètement, ce sont des utilisateurs qui sont physiquement proches du routeur et qui sont connectés à son réseau local. Finalement, il y a les attaquants physiques. Ce type d'attaquant possède un accès physique au routeur et peut en profiter pour faciliter son intrusion dans le système. Il peut ouvrir le routeur, connecter des fils aux ports sériels ou directement sur les branches de puces, enlever des puces, etc. Nous présumons toutefois que ce type d'acteur n'a pas des moyens illimités. Par exemple, il ne sera pas en mesure d'espionner ou de modifier les circuits internes des puces en utilisant des outils sophistiqués et coûteux comme des faisceaux d'ions [24]. Les types d'attaquants sont illustrés dans la Figure 2.2.

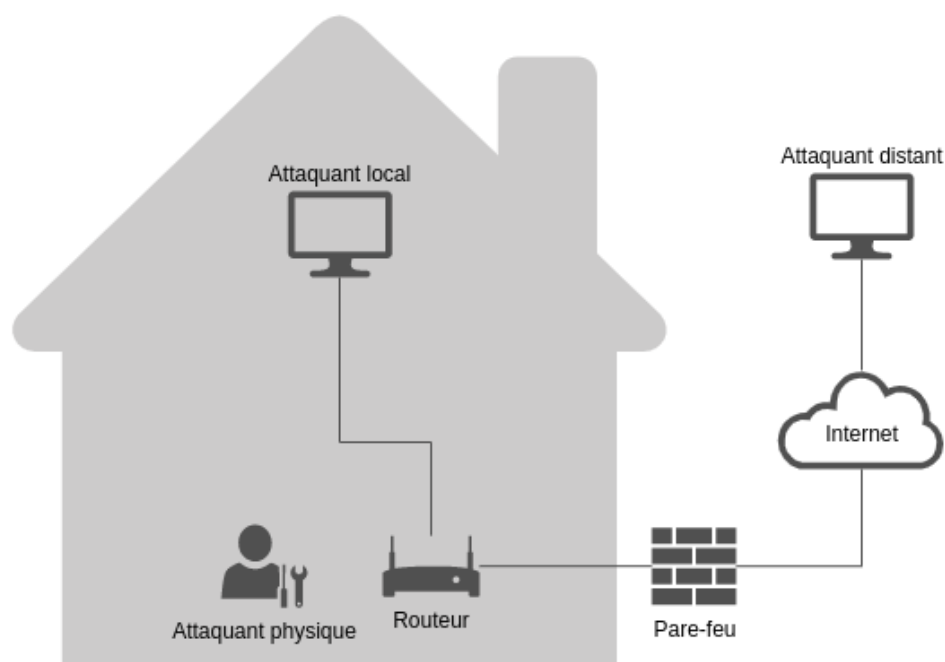


FIGURE 2.2 Types d'attaquants

L'objectif de ce projet est de protéger une donnée sensible qui réside sur l'appareil afin qu'aucun des types d'attaquants ne puisse en connaître le contenu. Lorsqu'on regarde les critères fondamentaux de la sécurité informatique, c'est-à-dire la confidentialité, l'intégrité et la disponibilité [25], on constate que l'aspect confidentialité est le plus visé par ce projet. L'intégrité est elle aussi ciblée, car la donnée sensible qui est protégée peut servir à établir

une base de sécurité pour d'autres composantes, de la même manière qu'un mot de passe peut protéger des fichiers chiffrés. Si la confidentialité du secret est brisée, l'intégrité des données protégées peut être compromise. Toutefois, la disponibilité n'est pas prise en compte, car il est moins dangereux d'avoir un routeur hors service qu'un routeur infecté par un acteur malveillant. Les attaques de déni de services ne font donc pas partie du modèle de menace.

Du point de vue du logiciel, la menace réside dans les services qui sont exposés au réseau. Nous présumons que tout programme qui utilise des données provenant du réseau peut potentiellement contenir une faille aboutissant en l'exécution de code arbitraire dans son contexte par un attaquant. Seuls les services accessibles sur le réseau étendu pourront être exploités par les attaquants à distance, mais ceux qui sont accessibles sur le réseau local pourront tout de même être exploités par les attaquants locaux. Les routeurs qui exposent intentionnellement moins de services auront moins de chances d'être exploités au niveau logiciel. Cependant, même si un routeur n'offre aucun service, il doit minimum assurer le routage des paquets réseau. Cette fonctionnalité centrale peut elle aussi être vulnérable à des failles logicielles qui peuvent résulter en de l'exécution de code avec les privilèges du noyau du système d'exploitation. Le projet AMNESIA :33 met en lumière 33 vulnérabilités trouvées dans des piles logicielles à source ouverte qui ont été utilisées dans des milliers de projets embarqués [26]. La plupart des routeurs n'ont que l'utilisateur racine présent dans leur système d'exploitation, ce qui signifie que toute vulnérabilité qui résulte en de l'exécution de code sur l'appareil se fera directement avec les privilèges racines. Or, même lorsque ce n'est pas le cas, des vulnérabilités logicielles dans diverses composantes pourront permettre à un attaquant d'élever ses privilèges jusqu'à l'utilisateur racine. Nous assumons donc qu'un attaquant sera toujours en mesure d'exécuter du code arbitraire en tant qu'utilisateur racine dans le système d'exploitation du routeur.

Du côté matériel, la menace ne peut provenir que des attaquants physiques. La surface d'attaque varie grandement d'un routeur à un autre, selon le circuit de leur carte mère et les ports qui y sont exposés. Cependant, certaines caractéristiques seront toujours présentes, comme le fait d'avoir une puce mémoire séparée qui contient le micrologiciel de démarrage, le système d'exploitation et le système de fichiers. Une fois de plus, le modèle de menace peut changer selon le type de puces utilisé, les périphériques présents et la facilité d'accès aux circuits d'un point de vue physique. Toutefois, en dernier recours, un attaquant possédant de bons moyens techniques sera toujours en mesure de prélever la puce mémoire de l'appareil et d'y lire le contenu du système de fichiers. Dès lors, nous assumons que les attaquants pourront toujours lire le contenu complet du système de fichier et du système d'exploitation.

Il faut mentionner que les mesures de sécurité relatives au Trusted Computing peuvent elles

aussi être attaquées. Par exemple, une application isolée et sécurisée par un TEE qui possède des erreurs de logique et vérifie mal les valeurs fournies par l'utilisateur peut tout de même être attaquée et compromettre la confidentialité de données sensibles. Puisque nous avons établi qu'un attaquant est en mesure d'exécuter du code avec les privilèges racines sur l'appareil, toute information qui provient de l'extérieur du TEE doit être considérée comme potentiellement malveillante. Toutefois, nous présumons que les primitives du Trusted Computing et du TEE sont robustes et bien implémentées.

2.2 État de l'art sur la sécurité logicielle

La sécurité logicielle des routeurs domestiques est grandement en retard par rapport à celle des logiciels et des ordinateurs tout usage. Küçüksille et al. [27] définissent 3 catégories de vulnérabilités pouvant affecter les routeurs, soit les failles logicielles, les erreurs de conception et les erreurs de configuration. Ils affirment que les erreurs de configuration sont les plus fréquentes bien qu'elles soient les plus faciles à régler, ce qui montre une certaine négligence de la part des vendeurs et des utilisateurs.

En 2020, Weidenbach et vom Dorp ont analysé 127 routeurs domestiques sur le marché pour en attester l'état de sécurité [2]. Ils ont utilisé le Firmware Analysis and Comparison Tool (FACT) [28] pour automatiquement extraire le micrologiciel et l'analyser. Ils ont ensuite évalué la sécurité des micrologiciels selon 5 critères.

1. Nombre de jours depuis la dernière mise à jour
2. Version du système d'exploitation et nombre de vulnérabilités connues
3. Techniques de mitigation d'exploitation employées
4. Quantité de clés cryptographiques privées
5. Nombre de mots de passe fixes

En moyenne, les vendeurs ont publié des mises à jour chaque année et demie, bien que dans le pire cas aucune mise à jour n'a été publiée depuis plus de 5 ans. Les systèmes d'exploitation et leurs versions ont été trouvés grâce à des expressions régulières et soumis dans la base de données de Common Vulnerabilities and Exposures (CVE) du National Institute of Standards and Technology (NIST), la National Vulnerability Database (NVD). Seules les vulnérabilités ayant un score critique ont été retenues. Les routeurs étudiés sont affectés par 53 vulnérabilités critiques en moyenne et même le moins touché est affecté par 21.

Pour ce qui est des techniques de mitigation, ils ont regardé le pourcentage des exécutables du système de fichiers qui utilisent des techniques courantes de mitigation d'exploitation binaire.

- NX : empêche l'exécution de code sur la pile
- PIE : rend aléatoire les adresses des régions en mémoire
- RELRO : protège les pointeurs de fonction des bibliothèques
- Stack canary : prévient les débordements de tampon sur la pile
- FORTIFY_SOURCE : protège toutes les fonctions de la bibliothèque standard C contre les débordements

Toutes ces protections sont connues et largement utilisées depuis de nombreuses années [29]. Les 4 premières sont d'ailleurs activées par défaut dans la plupart des versions de distribution de GNU Compiler Collection (GCC). Ils remarquent que presque tous les vendeurs ont grandement utilisé NX, mais très peu les autres techniques. Il est à noter que certains vendeurs sont vastement plus sécuritaires que d'autres.

En ce qui a trait aux derniers critères d'évaluation, les routeurs évalués ont en moyenne 5 clés privées par appareil. De plus, beaucoup de routeurs ont aussi des mots de passe par défaut très connus (faciles à trouver sur l'internet).

Il existe plusieurs solutions ayant pour objectif la sécurisation des logiciels modernes. Les mesures de protection des exécutables mentionnées plus haut en sont un bon exemple. Elles sont faciles à appliquer ; il s'agit simplement d'ajouter quelques options de compilation. Elles sont donc tout à fait adaptées aux routeurs. De plus, elles sont plutôt efficaces et peuvent empêcher qu'une vulnérabilité de corruption de mémoire puisse être transformée en exécution de code et la restreindre à un déni de service. Malgré tout, ces protections ne sont pas parfaites et il existe de nombreuses méthodes pour les contourner, comme le Return Oriented Programming (ROP) [30].

Une autre approche à la sécurité logicielle est l'encapsulation des logiciels à risque dans un endroit isolé. Par exemple, la virtualisation permet de recréer tout un système de manière logicielle. Cela est très efficace pour contenir une menace, car même si un attaquant arrive à pénétrer le système virtuel, il ne sera pas capable d'accéder au système hôte. Cependant, la virtualisation est très coûteuse sur le plan des ressources et de la latence. La plupart des ordinateurs modernes possèdent des processeurs munis de fonctionnalités de virtualisation assistée par matériel, ce qui leur permet de réduire cette latence. Cependant, ces processeurs sont rarement utilisés dans les appareils embarqués à cause de leur coût élevé. Une alternative à la virtualisation est la conteneurisation. Celle-ci tire avantage des dernières fonctionnalités des systèmes d'exploitation modernes pour isoler un processus ainsi que toutes ses dépendances dans un environnement restreint. Cela présente de nombreux bénéfices, comme une très faible latence, la portabilité, la reproductibilité et la sécurité [31]. En effet, si une application conteneurisée est compromise, la portée de l'attaque est restreinte à l'intérieur du

conteneur ciblé et le système hôte est encore une fois protégé. Dû à sa récente popularisation, sa facilité d'utilisation et sa rapidité, la conteneurisation est plus propice à une utilisation sur des appareils embarqués que la virtualisation. Toutefois, à notre connaissance, il n'existe pas de routeur domestique qui place des processus dans la sécurité d'un conteneur. Cela est probablement dû à un des principes fondamentaux des conteneurs, soit qu'ils servent à isoler des applications individuelles. Idéalement, un seul processus devrait rouler à la fois par conteneur. Cette isolation, qui renforce la sécurité du système, nuit en même temps à la communication entre les applications. Or, pratiquement tous les processus sur un routeur sont interdépendants et interconnectés. Par exemple, l'interface web doit pouvoir changer la configuration des tables de routage et le serveur DNS doit pouvoir consulter la liste des adresses réservées. Ces solutions d'isolation ne sont donc pas adaptées au contexte des routeurs domestiques.

Similairement, il est possible d'installer un Intrusion Detection System (IDS) ou un pare-feu dans le même réseau que le routeur. Des solutions comme OPNsense [32] sont faciles à déployer sur une multitude de plateformes et augmentent instantanément la sécurité du réseau. Similairement, Sharma et al. [11] proposent d'adapter le concept de réseau défini par logiciel aux réseaux domestiques pour en augmenter la sécurité et la modularité. Le système SERENIoT [33], proposé par Corentin Thomasset, met en place des appareils sentinelles dans le réseau domestique pour détecter les anomalies à l'aide d'une blockchain publique. Dorri et al. [34] ont créé une autre solution utilisant une blockchain adaptée aux appareils embarqués pour augmenter la sécurité d'un réseau en déléguant la majorité des calculs aux noeuds ayant le plus de ressources. Dua et al. [35] ont mis en place des règles Snort [36] en collaboration avec un renifleur de paquets sur un routeur domestique pour détecter les attaques sur les appareils connectés. Ils ont démontré que leur approche peut efficacement défendre les appareils contre 10 attaques bien connues tout en utilisant peu de matériel. TrustR [37] est un routeur spécialement conçu pour protéger son réseau contre des attaques au niveau du routage des paquets, comme l'usurpation d'adresse IP ou le déni de service. Il utilise plusieurs métriques pour calculer un niveau de confiance en chaque noeud du réseau et expulser ceux qui ne sont pas dignes de confiance. Cependant, cette solution ne diminue pas les probabilités ni l'impact d'une brèche logicielle sur le routeur. Bien que les pare-feux peuvent bloquer les attaques logicielles visant un routeur et les IDS peuvent les détecter, ces solutions n'augmentent pas en soi la sécurité logicielle du routeur. Elles peuvent donc être considérées comme complémentaires à ce projet, mais pas comme un remplacement. À notre connaissance, il n'existe pas de routeur qui utilise le Trusted Computing pour augmenter sa sécurité, protéger une information sensible et diminuer l'impact en cas de brèche.

Finalement, la sécurité logicielle des routeurs peut être améliorée en suivant des directives de codage sécuritaires lors de l'implémentation des applications qui s'y retrouvent. Malheureu-

sement, l'application de ces directives est à la discrétion des manufacturiers et il n'est pas réaliste d'espérer qu'ils les mettront tous en place. De plus, des erreurs de programmation et des failles de sécurité peuvent toujours exister même si les meilleures directives ont été suivies. L'utilisation de langages de programmation sécuritaires, comme Rust [38], peut aussi aider à réduire les failles de corruption de mémoire, mais des erreurs de logique, de contrôle d'accès et autres, peuvent toujours survenir.

2.3 État de l'art sur la sécurité matérielle

La sécurité matérielle est beaucoup plus difficile à mettre en place, car elle requiert des changements dans des composantes physiques, comme le processeur ou les bus de communication. Or, elle est aussi extrêmement plus robuste que la sécurité logicielle. Toute la sécurité logicielle repose sur des permissions et des contrôles d'accès qui ne sont concrètement que des valeurs en mémoire. Avec des permissions assez élevées, ces valeurs sont modifiables et les mesures de sécurité peuvent être désactivées. La sécurité logicielle est donc par définition imparfaite et contournable. De l'autre côté, les mesures de sécurité matérielle, une fois mises en place, sont pratiquement incontournables à moins de modifier le circuit lui-même. Il est à noter que même si la fonctionnalité matérielle ne peut être contournée, cela ne rend pas le système en entier invulnérable. Elle peut tout de même être mal conçue ou mal utilisée de manière à être exploitable. Un bon exemple est l'attaque Boomerang [39] sur les TEE, discutée plus en détail dans la section 3.6.

Plusieurs projets s'intéressent à la sécurité matérielle des appareils embarqués. SMART [40] établit une racine de confiance dans un appareil embarqué à l'aide d'une architecture hybride entre logiciel et matériel. Un serveur, appelé vérificateur, peut ensuite faire exécuter du code arbitraire sur les appareils SMART en ayant la certitude de son intégrité. Cette architecture permet au vérificateur à la fois de prouver l'identité de l'appareil, de confirmer le contenu d'une partie de sa mémoire et d'être assuré qu'il a bel et bien exécuté le code fourni. Sancus [41] est une autre architecture visant à exécuter du code de manière sécuritaire sur un appareil embarqué distant. Sa particularité est qu'elle permet d'exécuter n'importe quelle application, même des applications tierces non dignes de confiance, de manière sécurisée. Ces applications sont chargées comme des modules et peuvent interagir entre elles ainsi que vérifier leur intégrité respective par le biais d'un appel système dédié. L'isolation du code et des données est basée sur une hiérarchie de clés cryptographiques ainsi que sur la prise en compte du pointeur d'instruction dans la gestion des accès mémoire. Son Trusted Computing Base (TCB) ne comprend que le matériel, ce qui veut dire que la logique d'isolation ne peut être brisée par une erreur logicielle. Son implémentation inclut des modifications au

processeur pour déceler les accès mémoire non autorisés ainsi qu’un compilateur personnalisé pour faire usage des propriétés d’isolation ajoutées. Quelques années plus tard, les auteurs de Sancus ont dévoilé une amélioration de leur architecture, soit Sancus 2.0 [42]. Cette nouvelle version apporte des améliorations comme un meilleur partage des ressources entre les applications et la possibilité de déployer une application de manière chiffrée pour garantir sa confidentialité. L’architecture TrustShadow [43] atteint un objectif similaire en utilisant les capacités de ARM TrustZone. Elle permet de protéger l’intégrité d’une application arbitraire en exécutant à sa place son ombre (shadow), qui en est une réplique située dans le Monde Sécurisé. Puisque le Monde Sécurisé ne possède que des fonctionnalités réduites, les appels système de l’ombre sont relayés au système d’exploitation résidant dans le Monde Normal par le biais d’une structure de données spéciale. Un module dans le noyau reçoit ces requêtes et simule que les appels système sont lancés par l’application elle-même, avant de renvoyer le résultat à son ombre. Puisque le système d’exploitation du Monde Normal n’est pas digne de confiance, toutes les valeurs de retour des appels système sont vérifiées avant d’être utilisées. Cela permet de sécuriser des applications sans leur apporter la moindre modification. Il faut simplement qu’elles soient enregistrées à leur lancement à l’aide d’un appel système spécial.

TrustLite [44] propose une modification minimale de l’architecture des systèmes embarqués pour réussir à isoler des programmes sensibles, appelés *trustlets*, et garantir leur exécution sécuritaire. Cette solution est basée sur une unité de gestion de la mémoire (MMU) qui prend en considération la valeur du pointeur d’instruction (IP) lors d’une demande d’accès à la mémoire. Cela permet aux *trustlets* d’avoir un accès exclusif à certaines plages d’adresses, tandis qu’un programme spécial de lancement agit comme TCB.

Toutes ces solutions sont plutôt dirigées vers des appareils embarqués qui ont des contraintes matérielles encore plus restrictives que les routeurs, comme des capteurs IoT. De plus, la plupart requièrent un processeur modifié, ce qui explique pourquoi leurs prototypes sont faits sur des Field-Programmable Gate Array (FPGA). Ces solutions ne répondent donc pas à notre objectif d’obtenir une architecture reposant sur du matériel déjà existant, peu dispendieux et assez puissant pour subvenir aux fonctionnalités d’un routeur.

2.4 Étude de cas sur l’exploitation d’un routeur

Afin de démontrer la pertinence d’intégrer des mesures de sécurité comme le Trusted Computing dans les routeurs domestiques, nous avons mené une étude de cas sur la sécurité d’un routeur commercial populaire. L’objectif de cette étude est de vérifier quels sont les risques de sécurité présents sur un routeur qui n’a aucune vulnérabilité connue. Elle a aussi pour cible de démontrer les avantages d’avoir un accès physique lors d’un test de sécurité sur un

appareil embarqué.

Le contenu de cette section est majoritairement une traduction de l'article *Security assessment of a commercial router using physical access*, écrit par *Colin Stéphenne*, en cours de parution sur arXiv à l'hiver 2023.

2.4.1 Méthodologie

Les étapes de la méthodologie suivent de manière générale celles du *OWASP Firmware Security Testing Methodology* [21]. L'objectif d'un test de sécurité est de trouver toutes les vulnérabilités présentes dans un système. Il est donc primordial de ne pas arrêter le test lorsqu'une vulnérabilité est trouvée et d'inspecter toutes les composantes systématiquement. Cependant, il est pratiquement impossible de tester toutes les fonctionnalités lors d'un test en boîte noire, car il y a toujours un risque de ne pas trouver une fonctionnalité cachée qui contient une faille de sécurité. L'objectif intermédiaire est donc d'acquérir un accès au système de fichiers pour en extraire le code et les exécutables des services qui roulent sur le routeur. Ce faisant, nous sommes en mesure de transformer le test boîte noire en test boîte blanche et de couvrir toutes les fonctionnalités exposées.

La première étape est de déterminer la surface d'attaque. Sur le plan logiciel, le routeur n'a pas d'interface graphique ni de périphériques d'entrée. La seule manière d'interagir avec lui est donc à travers le réseau, en se connectant aux ports ouverts. Il est donc essentiel de faire un balayage de ports systématique. Les services exposés peuvent avoir des vulnérabilités connues, mais dans le cas contraire il est important de les examiner manuellement pour se familiariser avec les fonctionnalités offertes. Il n'est pas nécessaire de passer beaucoup de temps à essayer d'exploiter les services à ce point-ci, car cette tâche sera grandement facilitée une fois que le code des applications sera obtenu. La deuxième étape est de chercher des accès au système de fichier au niveau matériel. Si des connecteurs UART ou JTAG sont présents sur la carte mère, ils peuvent donner accès à des fonctionnalités très puissantes. La manière d'en tirer profit varie grandement selon les fonctionnalités exposées. S'il n'y a pas de connecteurs, il peut être possible d'exfiltrer le contenu de la mémoire en se connectant aux branches de la puce mémoire et en lisant les signaux qui y circulent. En dernier recours, la puce mémoire peut être enlevée de la plateforme et ses données pourront y être lues directement. À ce point-ci, une des étapes aura permis d'extraire le système de fichiers. La dernière étape est donc d'analyser les programmes qui opèrent sur le routeur afin d'y trouver des vulnérabilités. Les différentes phases de la méthodologie sont résumées dans la Figure 2.3.

Les sous-sections suivantes décrivent l'application de la méthodologie pour effectuer une évaluation de la sécurité d'un routeur commercial. Le routeur choisi pour cette étude de cas

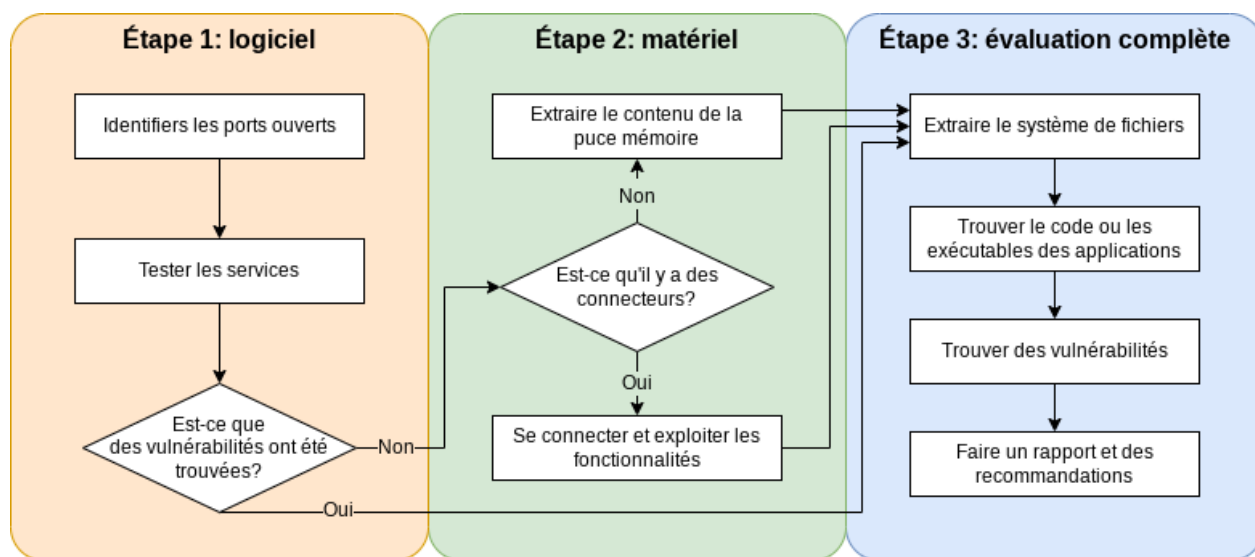


FIGURE 2.3 Survol des étapes de la méthodologie

est un routeur populaire de moyenne gamme. Il a été choisi pour ses bonnes performances à un prix raisonnable, typique d'un routeur qui pourrait être présent dans un domicile normal. Il supporte le wifi 2.4 et 5 GHz et possède plusieurs ports Ethernet. Le test de sécurité a été effectué à la fois sur l'interface locale du routeur et sur l'appareil physique.

2.4.2 Reconnaissance

Un balayage des ports avec l'outil *nmap* [45] révèle quelques services, incluant une interface web sur le port 80. Il est standard pour un routeur de fournir une interface de configuration web, car c'est la manière la plus simple de permettre aux utilisateurs de personnaliser les paramètres de leur réseau. Toutefois, c'est aussi une des cibles les plus prometteuses pour l'exploitation du routeur, car l'interface est souvent faite sur mesure pour chaque modèle et des erreurs de programmation peuvent facilement s'y glisser. De plus, puisque les manufacturiers veulent produire leurs appareils à des prix compétitifs, le code est souvent écrit rapidement et ne suit pas toujours de bonnes pratiques de codage.

Il y a d'autres services exposés par le routeur, comme le serveur DNS *dnsmasq* et le serveur UPnP *miniupnpd*. Cependant, leurs versions n'ont pas de vulnérabilités connues. Tenter d'en trouver est hors de la cible de cette étude, car ce sont des projets à part entière qui sont indépendants du routeur et de son manufacturier.

Le routeur doit être évalué à la fois du côté logiciel et du côté matériel. En premier lieu, l'interface web et ses fonctionnalités doivent être examinées pour y trouver des points d'entrée

potentiels. Ensuite, si aucune vulnérabilité n'est trouvée, la carte mère pourra être enlevée de sa coquille et les composantes physiques pourront être identifiées.

Interface web

L'interface web est assez classique pour un routeur domestique. Le panneau de configuration est protégé par un mot de passe administrateur et il sert à changer les paramètres du réseau. Il n'y a pas de chiffrement TLS en place, ce qui s'explique par la complexité de la distribution des certificats. De plus, le site n'a pas de nom de domaine puisqu'il est accédé directement par son adresse IP. Toutefois, ce n'est pas vraiment un risque de sécurité puisque l'interface ne devrait être accédée que depuis le réseau local. La plupart des pages ont l'extension *.asp*, ce qui suggère que le site utilise *ASP.NET*.

Le site web contient plusieurs sections, qui exposent différents paramètres et permettent d'en changer les valeurs. Les vecteurs d'exploitation les plus fréquents dans ce genre de situation sont les entrées de l'utilisateur, car il arrive que les développeurs assument un certain format pour l'entrée et oublient de vérifier si le format est bien respecté. Dans ce cas, une entrée malformée ne sera pas rejetée par le site, mais risque de le mener dans un état inattendu. Dépendamment de la fonctionnalité affectée, cet état peut être attaqué en créant minutieusement une entrée qui amènera le routeur à exécuter des commandes arbitraires.

Un proxy pour le protocole Hypertext Transfer Protocol (HTTP), comme Burpsuite [46], peut être utilisé pour intercepter les requêtes lancées vers l'interface et évaluer si certaines entrées ont le potentiel d'introduire un état inattendu. Par exemple, si un paramètre n'accepte que des chiffres en entrée, il est peu probable que la valeur soit mal validée. Même si elle l'est, il est peu probable qu'une valeur invalide emmène le site dans un état interne qui pourrait résulter en de l'exécution de code. Au contraire, des entrées sous la forme de chaîne de caractères qui sont passées à un interpréteur de commandes sont beaucoup plus prometteuses, car y introduire des valeurs inattendues signifie que la commande passée à l'interpréteur peut être modifiée, résultant en de l'exécution de code. Certaines fonctionnalités sont plus faciles à implémenter en passant une commande à un interpréteur qu'en codant manuellement la fonctionnalité. Par exemple, les fonctionnalités fournies par les utilitaires qu'on retrouve fréquemment sur les systèmes Linux, comme *ls*, *ping* ou *grep*, peuvent être implémentées par un simple appel à l'interpréteur de commandes. De plus, certaines de ces commandes comprennent des interactions avec le système de fichiers ou les paquets réseau qui peuvent être compliquées à coder manuellement. C'est pourquoi les développeurs peuvent être fortement tentés de faire un appel à l'interpréteur pour ces fonctionnalités. Plus la fonctionnalité est complexe, plus cette tentation est forte. Ces fonctionnalités sont donc très intéressantes à

examiner lors d'une évaluation de sécurité.

La page de diagnostic offre ce genre de fonctions à l'utilisateur. Elle permet d'envoyer des paquets *ping* et *traceroute* à d'autres machines sur le réseau. Puisque les commandes *ping* et *traceroute* sont des utilitaires Linux bien connus, cette fonctionnalité a un grand potentiel d'injection de commandes.

Le serveur web peut lui-même être vulnérable à certaines attaques comme de la corruption de mémoire, des traversées de répertoire, de la contrebande de requête HTTP, etc. Les entêtes des réponses HTTP nous indiquent que le serveur est *GoAhead Web Server*. Toutefois, il n'existe aucune vulnérabilité connue pour cette version. Il est aussi hors de la portée de cette étude de trouver des vulnérabilités dans le projet *GoAhead*, car le but est de mettre l'accent sur un modèle précis de routeur.

Ports sériels

Durant le développement d'une plateforme, il est courant d'utiliser un connecteur sériel pour afficher des informations à des fins de débogage. Les protocoles sériels sont simples d'utilisation, rapides et ne nécessitent pas de composants matériels complexes comme une interface graphique. Ils sont donc parfaits pour interagir avec des petits appareils dans leurs premiers stades de développement.

Les protocoles sériels les plus communs sont Inter-Integrated Circuit (I2C), Serial Peripheral Interface (SPI) et UART. Le protocole SPI utilise 4 fils (sans compter le voltage et la mise à la terre) et peut échanger des données dans les deux sens simultanément. Le protocole I2C utilise également 4 fils (sans compter le voltage et la mise à la terre) et est très rapide, mais ne peut échanger que dans une seule direction à la fois. Le protocole UART est généralement plus lent que les autres options, mais il est le plus simple à implémenter, car il ne requiert que 2 registres à décalage. C'est pourquoi il est le plus populaire pour diagnostiquer des plateformes lors de leur développement. Il n'utilise que 2 fils (sans compter le voltage et la mise à la terre), un pour chaque direction des données, nommés TX et RX pour la transmission et la réception, respectivement.

Le routeur étudié a 4 connecteurs vides au milieu de sa carte mère, comme le montre la Figure 2.4. L'identification du port est grandement facilitée par le fait que les broches sont clairement identifiées. De tous les protocoles sériels, seul UART possède des fils nommés TX, RX, 3.3V (voltage) et GND (mise à la terre).

Afin d'interagir avec l'appareil à travers le port, il faut se connecter à ses broches. Des connecteurs ont donc été soudés dans les trous et connectés à un circuit capable d'interpréter

de l'appareil qui permet de confirmer ou d'infirmer la valeur de *baudrate* utilisée. S'il n'y a toujours aucune donnée reçue, il peut être pertinent de redémarrer l'appareil à chaque essai de fréquence, car nombre d'appareils envoient des données de diagnostic durant leur séquence de démarrage. Lors de cette étude, il fut assez facile de déterminer la bonne valeur de *baudrate* à 57600, car le routeur envoie des informations sur le réseau à intervalles réguliers dans la connexion UART.

Il arrive que des appareils IoT fournissent un accès direct à un interpréteur de commande à quiconque se connecte à leur port sériel [18]. Bien que cela doit être utile lors du développement de la plateforme, c'est une très mauvaise pratique que de laisser cette fonctionnalité en production. Cependant, ce n'est pas le cas du routeur étudié. Même si l'information divulguée à travers le port sériel est intéressante, elle n'apporte rien qui puisse aider à l'exploitation du routeur ou à l'extraction du système de fichiers. De plus, il ne se passe rien quand des données sont envoyées à la connexion, ce qui suggère que le port est peut-être configuré en lecture seule.

Mémoire flash

Parmi les avantages que procure l'accès physique au routeur, un des plus puissants est l'accès à la puce mémoire. La plupart des mesures de protection des données sont implémentées en logiciel et sont impuissantes lorsque l'appareil n'est tout simplement pas allumé. Aussi, les appareils IoT chiffrent rarement les données en mémoire, car cela ajouterait trop de latence à chaque lecture ou écriture de la mémoire. L'extraction manuelle des données de la puce mémoire est une des techniques d'exploitation IoT les plus puissantes, car elle est pratiquement infaillible [50]. Toutefois, elle peut être assez compliquée à exécuter sur certains appareils, dépendamment du modèle de la puce et des circuits de la carte mère. Par exemple, les puces Dual Inline Package (DIP) peuvent se faire accrocher par des crochets spécialisés, alors que les puces montées en surface doivent être dessoudées de la carte et soudées ailleurs pour être lues, ce qui augmente le niveau technique requis et les risques de briser l'appareil.

Le routeur étudié utilise une puce NAND flash 16 Mb GD25Q127CSIG de 8 broches [51]. La fiche technique, qui peut être trouvée en ligne, indique que la puce utilise le protocole SPI pour communiquer.

2.4.3 Extraction du micrologiciel

Puisque nous avons un accès physique au routeur, nous pouvons utiliser des techniques qui seraient impossibles à utiliser à travers le réseau. Nous pouvons modifier le circuit, accéder

aux puces et même souder nos propres composantes.

Il est possible d'identifier les puces grâce au numéro de modèle qui est souvent imprimé sur le dessus. Les appareils embarqués ont généralement au moins une unité centrale de traitement, une puce de mémoire flash pour enregistrer les données persistantes et une puce de mémoire vive, ou RAM.

Cette section décrit 2 méthodes différentes pour extraire le micrologiciel du routeur cible. Le *OWASP Firmware Security Testing Methodology* liste plusieurs stratégies pour obtenir le micrologiciel d'un appareil [21]. Par exemple, il suggère de visiter le site du fabricant ou de chercher sur l'internet au cas où quelqu'un en aurait laissé une copie sur un site de partage. Obtenir le micrologiciel est crucial pour accomplir une évaluation complète de la sécurité, mais ce ne sont pas toutes les méthodes qui fonctionnent pour tous les appareils.

Accrochage de la puce

La première méthode utilisée pour tenter d'extraire le micrologiciel est l'accrochage de fils aux broches de la puce mémoire pour interagir directement avec celle-ci. La carte *Attify* peut être connectée à la puce grâce à de petits crochets pour ensuite traduire le protocole SPI vers le protocole USB, comme le montre la Figure 2.5.

L'utilitaire *flashrom* [52] est capable de communiquer avec le protocole SPI et possède une liste de nombreuses puces et leurs caractéristiques. La mémoire flash de type NAND est structurée en groupes d'octets appelés des pages, qui sont les plus petites unités modifiables et qui sont elles aussi groupées en groupes appelés blocs, qui sont les plus petites unités effaçables [53]. Toutes les puces flash ne peuvent être lues et écrites avec la même logique, car les données à envoyer dépendent de la taille et de l'arrangement des pages et des blocs, des modes SPI supportés et de l'encodage binaire des commandes SPI.

Dans cette étude *flashrom* n'a pas pu extraire le contenu de la puce mémoire. Cela est dû à la manière dont le courant est distribué sur la plateforme. La puce doit être alimentée par sa broche de voltage. Or, cette broche est aussi connectée sur la carte mère à d'autres composantes qui sont alimentées par le même voltage. Cela fait en sorte que ces composantes sont aussi alimentées lorsque du courant est fourni à la puce et que le routeur démarre. Dès les premiers stades du démarrage, le routeur accède à la mémoire flash, ce qui crée de l'interférence avec l'extraction des données et rend toute donnée extraite inutilisable.

Ce problème aurait pu être réglé en dessoudant la puce, ou même juste sa broche de voltage, pour éviter que le reste de la carte mère soit alimenté. Cependant, il a été décidé que cette technique ne serait utilisée qu'en dernier recours pour éviter que cela brise le routeur, que

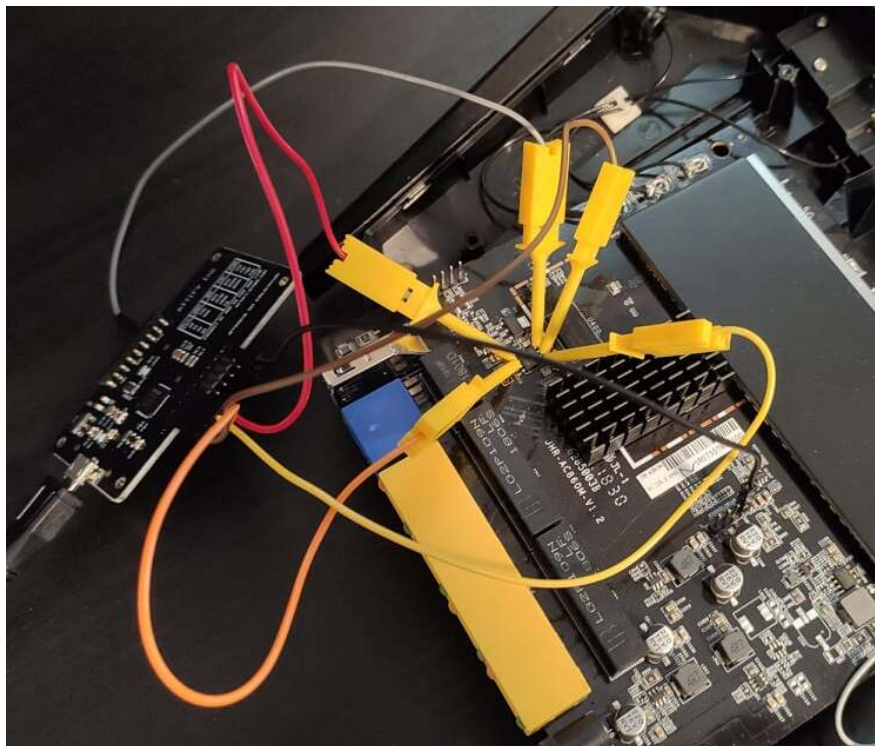


FIGURE 2.5 La badge *Attify* connectée à la puce par des crochets

nous n'avions qu'en un seul exemplaire.

Menu de démarrage

U-Boot est un micrologiciel de démarrage très populaire pour les appareils embarqués parce qu'il est à source ouverte et prend en charge de nombreuses architectures [22]. Par défaut, *U-Boot* expose un menu dans les premières phases du démarrage. Cela est très utile pour le développement, car le menu permet d'écraser le micrologiciel de démarrage avec une nouvelle version ou d'inspecter des parties variées de l'appareil sans avoir besoin d'un système d'exploitation fonctionnel.

Le routeur étudié possède bel et bien un menu de démarrage, mais il n'est visible que durant quelques secondes, c'est pourquoi il est passé inaperçu lors des premières phases de la reconnaissance.

Listing 2.1 menu de démarrage du routeur

Please choose the operation:

- 1: Load system code to SDRAM via TFTP.
- 2: Load system code then write to Flash via TFTP.
- 3: Boot system code via Flash (default).
- 4: Entr boot command line interface.
- 7: Load Boot Loader code then write to Flash via Serial.
- 9: Load Boot Loader code then write to Flash via TFTP.
- a: Load JCG firmware then write to Flash via TFTP.

Ainsi que le montre l'extrait 2.1, le menu de démarrage laisse l'utilisateur choisir entre un démarrage depuis la puce flash ou depuis le réseau à l'aide du protocole Trivial File Transfer Protocol (TFTP). L'option par défaut est automatiquement sélectionnée une fois le délai de quelques secondes écoulé. Or, l'option la plus intéressante pour une étude de sécurité est la quatrième, car elle donne accès à un interpréteur de commandes. Quand le routeur démarre, le choix peut être envoyé à travers la connexion UART. L'hypothèse que le connecteur UART est en lecture seule n'était donc pas véridique. Une fois la quatrième option choisie, l'appareil affiche une interface comprenant quelques commandes, montrées dans l'extrait 2.2.

Listing 2.2 Interpréteur de commandes de *U-Boot*

```
U-Boot U-Boot 1.1.3 (Jul 20 2018 - 10:07:23)
MT7621 # help
?      - alias for 'help'
bootm  - boot application image from memory
cp      - memory copy
erase   - erase SPI FLASH memory
go      - start application at address 'addr'
help    - print online help
loadb   - load binary file over serial line (kermit mode)
md       - memory display
mdio    - Ralink PHY register R/W command !!
mm      - memory modify (auto-incrementing)
nm      - memory modify (constant address)
printenv - print environment variables
reset   - Perform RESET of the CPU
rf       - read/write rf register
saveenv - save environment variables to persistent storage
```

```

setenv  - set environment variables
spi     - spi command
tftpboot - boot image via network using TFTP protocol
version - print monitor version

```

Puisque la fiche technique de la puce flash indique qu'elle communique par SPI, la commande *spi* est particulièrement intéressante, car elle pourrait servir à extraire le micrologiciel. L'extrait 2.3 indique les différentes options disponibles.

Listing 2.3 usage of the spi command

```

MT7621 # help spi
spi spi usage:
    spi id
    spi sr read
    spi sr write <value>
    spi read <addr> <len>
    spi erase <offs> <len>
    spi write <offs> <hex_str_value>

```

La commande prend en paramètre l'opération (lecture ou écriture), l'adresse et la taille et affiche le résultat en hexadécimal. Le processus est lent et la taille maximale pouvant être lue en un seul appel est assez réduite en raison de la petite quantité de mémoire vive disponible. Puisque la puce mémoire contient 16 Mb de données, lire tout son contenu par partie prendrait des heures. Il est donc préférable d'automatiser le tout. Un programme peut interagir avec la session de *screen* et automatiquement envoyer les commandes nécessaires à afficher tout le contenu de la puce et sauvegarder le résultat dans un fichier texte. Ensuite, un autre programme peut lire la sortie hexadécimale et reconstruire le contenu binaire de la mémoire.

2.4.4 Rétro-ingénierie du micrologiciel

Une fois que le contenu de la mémoire est reconstruit, elle doit tout de même être analysée pour y trouver des éléments pertinents. Étant donné que les services qui roulent sur le routeur sont composés de programmes binaires et des fichiers de code, il serait utile d'extraire le contenu du système de fichiers. Il faut noter que les systèmes d'exploitation en temps réel, comme FreeRTOS [54], n'ont pas de système de fichiers. Ils sont donc sensiblement plus compliqués à analyser.

L'outil *binwalk* [55] utilise des signatures binaires pour identifier des fichiers à travers un

amas de données binaires non identifiées. Les signatures sont des octets spécifiques à un type de fichier, souvent présents au début de celui-ci. Ils servent à assurer la validité d'un fichier ou comme séparateur pour des structures de données internes. Par exemple, les fichiers Executable and Linkable Format (ELF) commencent toujours par l'octet 0x7f suivi par la chaîne de caractères *ELF* [56]. En inspectant un amas de données pour trouver ces signatures, *binwalk* est donc capable d'en excaver des fichiers en utilisant des métadonnées pour en déduire la taille.

Lors de cette étude, *binwalk* a été en mesure d'identifier plusieurs composants dans le contenu de la mémoire flash, c'est-à-dire le micrologiciel de démarrage *U-Boot*, une image de noyau Linux et une archive Lempel-Ziv-Markov (LZMA). Une fois l'archive décompressée, il est clair qu'elle ne contient qu'un seul fichier binaire. En utilisant *binwalk* à nouveau, on y trouve une autre archive LZMA, qui contient à son tour une archive CPIO. C'est seulement lorsque celle-ci est décompressée que l'arborescence familière d'un système de fichiers apparaît.

En utilisant les commandes du menu de démarrage pour extraire le contenu de la puce mémoire et en utilisant les signatures de fichiers pour déceler des archives cachées, il a été possible d'extraire tout le système de fichiers du routeur sans jamais avoir à interagir avec son côté logiciel. Cela démontre que l'accès physique à un appareil peut grandement aider lors d'une évaluation de sécurité en transformant un test boîte noire en test boîte blanche.

L'inspection manuelle du système de fichiers révèle que le routeur opère sur BusyBox [57]. Il a été compilé avec l'option d'inclure *telnet*, un utilitaire qui peut être très utile pour gagner un accès au routeur à partir d'une exécution de code. Le processus d'initialisation de l'appareil indique que le binaire *goahead* est mis en route lors du démarrage. La prochaine étape consiste à dénicher les fichiers qui implémentent les fonctionnalités potentiellement vulnérables identifiées lors de la reconnaissance. Ensuite, une évaluation de la sécurité complète peut avoir lieu sur ces fichiers.

Serveur web

Un serveur web est une bonne cible pour trouver des vulnérabilités, car c'est souvent un des composants les plus complexes et les plus personnalisés sur un routeur domestique. Il est aussi accessible depuis le réseau, ce qui rend son exploitation réalisable à distance, bien que la plupart des routeurs modernes ne rendent pas l'interface de configuration accessible sur le réseau étendu par défaut.

L'étape de la reconnaissance a identifié que les pages de l'interface web utilisent l'extension *.asp*. Depuis la racine du système de fichiers, une recherche pour cette extension révèle des

fichiers dans le dossier */etc_ro/web*. La majorité des pages visitées lors de la reconnaissance peuvent être trouvées dans ce dossier. Cependant, d'autres pages sont introuvables dans l'entièreté du système de fichiers, entre autres celles dont l'URL commence par */goform*. Cela est causé par une fonctionnalité de *GoAhead*, qui permet aux développeurs de créer des pages personnalisées en C, appelées *GoForms* (ou *GoActions* dans les versions plus récentes) [58].

La page de diagnostique, qui permet à l'administrateur d'envoyer des requêtes *ping* et *traceroute*, est implémentée dans une page ASP. Cependant, cette page ne fait qu'appeler une autre, soit le *GoForm* */goform/sysTools*. Il est donc essentiel d'investiguer plus en profondeur la manière dont *GoAhead* gère les requêtes vers les *GoForms* afin d'évaluer la sécurité de l'interface web.

GoAhead

GoAhead Web Server est le serveur web le plus utilisé sur les appareils embarqués [59]. Ces principaux attraits sont sa taille compacte, son taux élevé de gestion de requêtes et la modularité des pages web grâce à l'utilisation de CGI et des *GoForms*. Les *GoForms* sont des routines en C qui sont ajoutées dynamiquement comme pages web pour enlever le délai d'appeler des programmes binaires externes. *GoAhead* peut aussi servir des fichiers ASP, donc ASP.NET n'est pas réellement utilisé sur le routeur.

Puisque *GoAhead* est un exécutable compilé, un décompilateur, comme Ghidra [60], est essentiel pour comprendre son fonctionnement interne. Un décompilateur utilise des algorithmes complexes pour recréer des fonctions en C à partir de code binaire compilé. La plupart des décompilateurs offrent aussi les fonctionnalités de créer des graphes de fonctions, identifier les chaînes de caractères, trouver les références à des données, et bien d'autres.

Les multiples fonctions de Ghidra permettent de retrouver la fonction *sysTools*, responsable des fonctionnalités de *ping* et de *traceroute*. Une recherche pour la chaîne *sysTools* mène vers la fonction qui est appelée lorsqu'un utilisateur envoie une requête à l'URL */goform/sysTools*. La fonction est reliée à l'URL dynamiquement lors de l'initialisation du programme, ainsi que toutes les autres pages dont l'URL commence par */goform/*. Une analyse plus poussée de la fonction peut ensuite révéler l'état de la sécurité de cette fonctionnalité.

2.4.5 Résultats et discussion

Cette section énumère les vulnérabilités qui ont été trouvées dans le micrologiciel du routeur. En premier lieu, les fonctions relatives à certains *GoForms* sont analysées. Puis, la sécurité générale de l'appareil est évaluée en interagissant directement avec les fonctionnalités et le

système de fichiers du routeur pendant qu'il est en marche.

Injection de commandes

Le *GoForm sysTools* utilise la fonction *snprintf* pour insérer les paramètres fournis par l'utilisateur dans la commande *ping* ou *traceroute*. La fonction *snprintf* est considérée sécuritaire, car son paramètre de taille aide à prévenir les débordements de tampon. La commande ainsi construite est ensuite passée à un interpréteur de commandes.

Listing 2.4 the decompiled *sysTools* function

```

1 void sysTools(char *request)
2 {
3     char *nest_page;
4     char *tool;
5     char *repeat;
6     char *host;
7     int tool_number;
8     char iter;
9
10    nest_page = websGetVar(request, "nextPage", "adm/systools.asp");
11    tool = websGetVar(request, "tool", "");
12    repeat = websGetVar(request, "repeat", "");
13    host = websGetVar(request, "host", "");
14    tool_number = atoi(tool);
15    if (*repeat == '\0') {
16        if (tool_number == 0) {
17            tool = websGetVar(request, "pingCount", "4");
18            snprintf(&command, 0x400, "%s -c %s %s 1>%s 2>&1", &ping, tool, host, "↵
                /var/systool.log");
19        }
20        else if (tool_number == 1) {
21            snprintf(&command, 0x400, "%s -n -m 4 %s 1>%s 2>&1", "traceroute", ↵
                host, "/var/systool.log");
22        }
23    }
24    else {
25        if (command == '\0') goto abort;
26        doSystem(&command);
27    }
28    if (command != '\0') {
29        if (((((command != ';') && (command != ',') && (command != '|')) &&
30            (((command != '\\' && (command != '\n')) && (command != '`'))))) {
31            tool = &command_second_char;

```

```

32     do {
33         iter = *tool;
34         tool = tool + 1;
35         if (iter == '\\0') goto execute_command;
36     } while (((iter != ';' ) && (iter != ',')) &&
37             ((iter != '|' && (((iter != '\\' && (iter != '\"')) && (↵
                 iter != '`')))))));
38 }
39     snprintf(&command,0x400,"echo InvalidCommand! 1>%s 2>&1","/var/↵
        systool.log");
40 execute_command:
41     doSystem(&command);
42 }
43 abort:
44     websRedirect(request,nest_page);
45     return;
46 }

```

Avant l'exécution de la commande, une liste de caractères interdits y est recherchée, visible aux lignes 29 et 30 ainsi qu'aux lignes 36 et 37 de la fonction. Ces caractères ont une signification spéciale pour l'interpréteur de commandes et peuvent causer des comportements inattendus. La vérification agit comme une mesure de sécurité contre une injection de commandes depuis l'entrée de l'utilisateur. Or, cette liste de caractères est incomplète. La majorité des interpréteurs de commandes, y compris Ash [61], celui utilisé par BusyBox, utilisent l'esperluette (&) pour enchaîner plusieurs commandes l'une à la suite de l'autre et le signe de dollar avec des parenthèses pour exécuter une commande dans un sous-processus. Puisque ces caractères ne sont pas détectés par le code, un attaquant peut exécuter des commandes arbitraires sur le routeur en envoyant des requêtes *ping* ou *traceroute* avec un paramètre *Host* spécialement assemblé. La sortie de la commande est ensuite affichée dans l'interface web. La requête suivante peut donc exfiltrer le contenu du fichier */etc/passwd*.

```
127.0.0.1 && cat /etc/passwd
```

Un accès direct au système du routeur pendant qu'il est en marche peut être très utile pour bien tester sa sécurité. Puisque BusyBox a des fonctionnalités limitées, la manière la plus simple d'y accéder est d'utiliser l'utilitaire *telnetd*.

```
$(telnetd -p 1234 && echo 127.0.0.1)
```

L'entrée ci-dessus ouvrira un serveur *telnet* sur le port 1234 sans perturber le fonctionnement de l'interface web. Une fois lancé, n'importe quelle machine sur le même réseau que le routeur

pourra s'y connecter, sauf qu'un nom d'utilisateur et un mot de passe sont requis pour accéder à l'interpréteur de commandes. Le mot de passe est le même que celui de l'interface web et le nom d'utilisateur peut être trouvé dans le fichier */etc/passwd* dont le contenu a été exfiltré précédemment.

Si les identifiants sont valides, l'utilisateur obtient un accès à l'interpréteur de commandes avec les privilèges racines. Puisque le mot de passe administrateur est nécessaire pour exploiter cette vulnérabilité, il s'agit d'une vulnérabilité d'exécution de code arbitraire authentifié. Dans un scénario plus réaliste, le mécanisme d'authentification de *telnet* peut être circonvenu aisément en utilisant la commande *wget* pour télécharger et exécuter une charge de son choix. Sinon, un mot de passe personnalisé peut être choisi pour *telnet* avec l'option *-password*.

Porte dérobée

Un autre *GoForm* est vulnérable à une injection de commandes. La fonction peut être appelée via l'URL */goform/SystemCommand*.

Listing 2.5 The decompiled *system_command* function

```

1 void system_command(char *request)
2 {
3     char *user_command;
4     int lan_ip;
5
6     user_command = websGetVar(request, "command", "");
7     if ((user_command != (char *)0x0) && (lan_ip = isLanIp(request + 0x30↵
8         ,0), lan_ip != 0)) {
9         if (*user_command == '\0') {
10             snprintf(&command, 0x400, "cat /dev/null > %s", "/var/system_command↵
11                 .log");
12         }
13     }
14     else {
15         snprintf(&command, 0x400, "%s 1>%s 2>&1", user_command, "/var/↵
16             system_command.log");
17     }
18     if (command != '\0') {
19         doSystem(&command);
20         websRedirect(request, "adm/system_command.asp");
21         return;
22     }
23     websRedirect(request, "adm/system_command.asp");
24     return;

```

La routine récupère une chaîne de caractères depuis un paramètre de la requête HTTP et l'exécute directement dans un interpréteur de commandes. Le résultat de la commande devrait être visible sur la page */adm/system_command.asp*, mais cette page n'existe pas dans le système de fichiers, possiblement dans un effort pour cacher cette fonctionnalité. Puisque ce *GoForm* n'a aucune autre utilité que d'exécuter des commandes avec les privilèges racines, il serait plus adapté de parler de porte dérobée que d'injection de commandes.

La raison de la présence d'une telle porte dérobée dans l'interface web du routeur est inconnue. Nous pouvons poser l'hypothèse que les développeurs l'ont placée à cet endroit pour faciliter le débogage et lorsqu'il est venu le temps de la production, ils ont enlevé le point d'accès, c'est-à-dire la page */adm/system_command.asp*, en oubliant d'enlever la fonctionnalité elle-même dans l'exécutable.

Vérification exhaustive

Après avoir trouvé des vulnérabilités d'exécution de code arbitraire authentifiée, il est naturel de vérifier si le mot de passe administrateur peut être exfiltré ou si le mécanisme de connexion peut être circonvenu. Si c'est le cas, plusieurs vulnérabilités pourraient être enchaînées pour obtenir de l'exécution de code arbitraire non authentifiée. La façon de faire la plus systématique est de cumuler tous les noms de fichiers dans le dossier de l'interface web et d'ajouter tous les noms de *GoForms* pour obtenir la liste complète de toutes les URL du serveur web. Puis, à l'aide du fuzzer web *wfuzz* [62], la liste peut être réduite aux chemins qui n'ont pas besoin d'authentification pour être accédés.

Les seuls fichiers accessibles sans authentification sont les images, des fichiers JavaScript inoffensifs et la page de connexion principale. Cependant, il y a quelques *GoForms* qui sont aussi accessibles sans authentification, dont certains qui ont l'extension *.jcg*. La page *cmd.jcg* semble être une autre porte dérobée, car elle exécute les commandes passées via le paramètre HTTP *cmd* dans un interpréteur de commandes. Elle peut sembler extrêmement dangereuse étant donné qu'elle est accessible sans authentification, mais toutes les pages ayant l'extension *.jcg* ont en fait leur propre mécanisme d'authentification. La page *login.jcg* modifie une variable globale lorsqu'un utilisateur lui fournit le bon mot de passe et toutes les autres pages en *.jcg* vérifient cette variable avant d'exécuter quoi que ce soit. Ces pages constituent donc une autre porte dérobée authentifiée.

Les autres *GoForms* accessibles sans authentification ne permettent pas d'exécuter des commandes, mais certains divulguent des informations sur l'état du routeur. Ces pages suivent

toutes le patron *get<item>Ajax* où *item* varie d'une page à l'autre. Les informations divulguées vont du temps depuis le dernier redémarrage à l'adresse MAC en passant par l'adresse IP locale du routeur. Certaines pages montrent des informations plus sensibles, comme la liste des réseaux Wifi visibles par le routeur, ce qui pourrait aider un attaquant à géolocaliser l'appareil.

L'analyse du binaire *GoAhead* et des fichiers du serveur web ont révélé plusieurs vulnérabilités. Cependant, le reste du système contient aussi des failles.

Mot de passe en clair

Le code responsable du mécanisme d'authentification sur l'interface web utilise la commande *nvram_bufget* pour obtenir le bon mot de passe avant de le comparer à celui fourni par l'utilisateur. Aucun chiffrement ni hachage n'est impliqué, ce qui signifie que le mot de passe administrateur est enregistré en clair dans la mémoire vive non volatile, ou Non-Volatile Random Access Memory (NVRAM). L'utilitaire *nvram_bufget* sert à récupérer des valeurs stockées dans la NVRAM. Depuis un interpréteur de commande, le mot de passe administrateur peut donc être récupéré en clair avec la commande *nvram_bufget Password*, ce qui n'est pas sécuritaire.

Identifiants par défaut faibles

Le mot de passe racine par défaut est *admin*, ce qui est très faible, et n'a pas besoin d'être changé pour utiliser le routeur. Il y a bien un avertissement qui conseille de le changer, mais cette étape peut être sautée en cliquant sur le bouton *Setup later*. De plus, le mot de passe par défaut du réseau Wifi est *12345678* et n'a pas besoin d'être changé pour utiliser le réseau. Pire encore, il n'est jamais demandé à l'utilisateur de le changer. Si un utilisateur laisse en place ces identifiants faibles, il devient alors trivial d'utiliser les vulnérabilités d'exécution de code arbitraire authentifié mentionnées plus haut.

Mauvais contrôle d'accès

L'interface web reconnaît les appareils connectés par leur adresse MAC. Les sessions ouvertes n'expirent jamais, ce qui veut dire qu'elles seront valides tant que l'utilisateur ne se déconnecte pas ou que le routeur ne redémarre. La session reste valide même si l'ordinateur qui l'a initiée se déconnecte du réseau. Cela signifie que si l'administrateur légitime du routeur se connecte à l'interface web, un attaquant peut ensuite se faire passer pour lui en copiant son adresse MAC et il sera automatiquement connecté. En utilisant les vulnérabilités d'exécution de

code arbitraire authentifié, il sera en mesure de prendre le contrôle total du routeur même si cela fait des mois que l'administrateur légitime s'est connecté. Cette faille peut aussi être exploitée à distance moyennant une interaction de l'utilisateur. Si l'administrateur légitime est connecté à l'interface web, il suffit qu'il clique sur un lien malveillant reçu par courriel pour que les vulnérabilités d'injection de commandes soient exploitées depuis n'importe où sur l'internet.

2.4.6 Recommandations de sécurité

Bien que la découverte de vulnérabilités soit intéressante, l'objectif de cette étude est d'augmenter la sécurité des appareils embarqués. Dans cette section, nous discutons des moyens qui peuvent être pris pour remédier aux failles trouvées et créer des appareils plus sécuritaires. La Table 2.1 résume les vulnérabilités découvertes, ainsi que les composants affectées et l'impact sur la sécurité. La valeur de l'impact a été calculée en utilisant le calculateur CVSS 3.1 du NIST [63].

TABLEAU 2.1 Résumé des vulnérabilités découvertes

Vulnerability	Components affected	Impact
Injection de commandes dans un <i>GoForm</i>	Serveur web <i>GoAhead</i>	7.2
Porte dérobée dans un <i>GoForm</i>	Serveur web <i>GoAhead</i>	7.2
<i>GoForms</i> potentiellement dangereux	Serveur web <i>GoAhead</i>	6.6
Menu de démarrage disponible sur le port UART	Plateforme physique	5.7
Contrôle d'accès faible	Interface web	5.4
Fuite d'information dans des <i>GoForms</i>	Serveur web <i>GoAhead</i>	5.3
Mot de passe en clair	RAM non volatile	4.9
Identifiants par défaut faibles	Interface web et Wifi	3.0

L'impact le plus élevé vient des vulnérabilités d'exécution de code arbitraire, bien qu'elles ne soient pas critiques (impact de 9.0 ou plus), principalement parce que le mot de passe de l'interface web est requis pour les exploiter.

Détection d'adresses IP

La vulnérabilité d'injection de commandes est causée par une manière incorrecte de vérifier que l'entrée de l'utilisateur est une adresse IP. L'utilisation de liste noire de caractères n'est

jamais recommandée pour filtrer les entrées des utilisateurs contre les injections de commandes. Les listes blanches de caractères sont jugées plus sécuritaires, car elles protègent mieux contre les oublis ou les cas limites. Dans le cas de la fonction *sysTools*, des utilitaires comme *isDigit*, *strtok* ou *sscanf* peuvent être utilisés pour s'assurer que le champ *Host* ne contient que 4 nombres entre 0 et 255 séparés par des points. La validation des entrées d'utilisateurs est un des aspects les plus importants de la sécurité logicielle. Cette vulnérabilité à haut impact aurait pu être évitée en suivant des initiatives comme le *OWASP Secure Coding Practices-Quick Reference Guide* [64]. Ce guide parle de l'importance de filtrer les caractères potentiellement dangereux comme les séquences de contrôle d'interpréteur de commande dans la section *Input validation*.

Suppression des points d'entrée inutiles

Les autres vulnérabilités importantes sont attribuables à des points d'entrée dangereux et inutiles dans l'interface web. Les *GoForms* finissant en *Ajax* révèle toutes sortes d'informations et les *GoForms SystemCommand* et *cmd.jcg* exécutent n'importe quelle commande fournie par un utilisateur connecté. Ces pages ne sont utiles à aucune fonctionnalité du routeur. Ces pages, ainsi que toutes les pages finissant par *.jcg*, devraient donc être complètement enlevées du système.

Si les pages pouvant exécuter des commandes étaient véritablement utilisées par les développeurs pour aider au diagnostic des problèmes, ce n'est pas la façon dont ils auraient dû procéder. Au minimum, ils auraient pu faire en sorte que ces fonctionnalités soient plus faciles à enlever une fois le développement terminé, plutôt que d'avoir à modifier le code du serveur web.

Du côté matériel, l'accès initial au système de fichiers a été acquis grâce au menu de démarrage disponible via le port UART. Même si cela n'a pas permis directement d'exécuter du code arbitraire et que cette fonctionnalité n'est pas liée aux erreurs dans l'interface web, elle pourrait être utilisée pour installer un micrologiciel de démarrage malveillant. C'est donc une vulnérabilité en soi et il serait préférable de mettre le port UART en lecture seule ou de le désactiver complètement lorsque le routeur est prêt pour la production.

Gestion des identifiants et des sessions

La gestion de sessions à partir des adresses MAC est une très mauvaise pratique. N'importe qui sur le réseau local peut balayer les adresses MAC présentes et se faire passer pour elles en utilisant des techniques d'usurpation d'adresse MAC. Il serait plus sécuritaire d'utiliser

des témoins HTTP pour garder la trace des sessions ou de jumeler les 2 stratégies.

De plus, il n'est pas recommandé de laisser l'utilisateur racine rouler un serveur web. Il est conseillé de créer un utilisateur moins privilégié pour lancer les services qui ne sont pas de confiance pour minimiser l'impact dans l'éventualité de leur exploitation.

Les mots de passe par défaut de l'interface web et du Wifi devraient être aléatoires et différents pour chaque appareil. Dans le cas contraire, l'utilisateur devrait minimalement être obligé de changer le mot de passe lors de la première connexion.

Finalement, le mot de passe administrateur ne devrait pas être enregistré en clair. Il devrait être haché avec un sel avant d'être mis dans */etc/passwd* et dans la NVRAM.

2.4.7 Conclusion

Cette étude de cas a démontré le danger potentiel pouvant provenir de l'installation d'un routeur commercial ordinaire sans vulnérabilité connue. Elle a aussi montré le clair avantage d'avoir un accès physique à l'appareil lors d'une évaluation de sécurité. L'accès au matériel a permis d'extraire le contenu de la puce mémoire à travers le menu de démarrage. Plusieurs vulnérabilités ont été trouvées sur le routeur et des manières de les régler ont été proposées.

Les vulnérabilités découvertes sont sérieuses et pourraient exposer des milliers d'utilisateurs à une brèche de sécurité dans leur réseau domestique. Étant donné à quel point certaines failles sont directes et cachées dans un exécutable binaire, il faut garder à l'esprit la possibilité qu'elles soient des portes dérobées mises en place par une entité malveillante. Cela expose un problème majeur de la mondialisation et des chaînes d'approvisionnement. Des manufacturiers étrangers peuvent soit être inattentifs et engager des développeurs inexpérimentés pour sauver des coûts ou véritablement malveillants et associés avec des groupes de piratage. Dans les deux cas, le résultat est que des produits sont vendus à travers le monde avec du code mal révisé qui cache des vulnérabilités et des portes dérobées.

Il est donc très important que les vendeurs d'appareils IoT vérifient la sécurité du code écrit par de tierces parties avant de lancer leurs produits. Toutes les vulnérabilités présentées auraient pu être évitées si le manufacturier avait conduit sa propre évaluation de sécurité. De nombreuses ressources sont disponibles pour aider le développement sécuritaire d'application, comme le standard CERT Secure Coding [65]. Pour aller plus loin, les vendeurs pourraient être forcés d'engager des firmes externes pour évaluer la sécurité de leurs produits avant d'avoir la permission de les vendre.

Cette étude de cas démontre aussi l'importance et la nécessité d'améliorer la sécurité des routeurs domestiques. Il est probable que les logiciels roulant sur les routeurs auront toujours

des failles de sécurité malgré les meilleurs efforts des manufacturiers. C'est pourquoi il est pertinent d'ajouter une couche de sécurité matérielle. Si tous les micrologiciels de routeurs incorporaient du Trusted Computing et un TEE pour protéger les applications et les données sensibles, cela diminuerait énormément l'impact des vulnérabilités présentes dans les applications plus complexes et moins sécurisées.

CHAPITRE 3 LE TRUSTED COMPUTING

Le Trusted Computing, ou informatique de confiance, est une méthodologie qui permet de sécuriser un système de manière à avoir confiance en l'intégrité de ce qui y est exécuté. Certaines implémentations permettent aussi de prouver cette intégrité à une tierce partie, un processus appelé attestation. Bien qu'il soit encore émergent, le Trusted Computing est déjà présent dans de nombreuses applications, de la gestion des droits numériques, ou Digital Rights Management (DRM), au chiffrement de partition BitLocker [66] en passant par l'authentification sécurisée [67]. La sécurité qu'il procure repose sur un petit élément du système qui est digne de confiance et très robuste contre les attaques de toutes sortes. Cet élément est appelé racine de confiance et comporte souvent une composante matérielle étant donnée leur forte résistance aux attaques logicielles. La racine de confiance est utilisée pour vérifier l'intégrité d'un petit nombre de composantes qui sont ensuite jugées dignes de confiance. Celles-ci peuvent à leur tour vérifier l'intégrité de d'autres composantes et ainsi de suite, formant une chaîne de confiance. Les membres de cette chaîne forment la base de confiance du système, ou Trusted Computing Base (TCB). Il est impératif qu'ils soient exempts de faille, car toute la sécurité apportée par le Trusted Computing repose dessus. De plus, ils s'exécutent généralement avec des privilèges élevés sur le système. C'est pourquoi il est préférable de réduire leur nombre et leur taille au maximum, diminuant ainsi la surface d'attaque. Certaines solutions utilisent aussi à cette fin des composantes formellement vérifiées, comme le noyau Linux SeL4 [68].

La validité des vérifications repose sur des primitives cryptographiques comme les fonctions à sens unique et sur l'inviolabilité de la racine de confiance. C'est pourquoi le TCB inclut généralement la racine de confiance, le premier stade du démarrage et le mécanisme de vérification d'intégrité lui-même.

3.1 Racine de confiance

La racine de confiance est la pierre angulaire sur laquelle la sécurité du Trusted Computing repose. Sa propriété fondamentale est d'être la plus résistante possible aux attaques de toutes sortes, surtout celles logicielles. De plus, elle doit pouvoir attester avec certitude de la validité de l'état du système ou d'une autre de ses composantes. La racine de confiance peut prendre plusieurs formes selon les requis et les capacités d'un système. La forme la plus connue est le TPM, créé et popularisé par le TCG.

Il existe 2 types de racines de confiance, soit les racines statiques et les racines dynamiques. Les racines de confiance statiques ne peuvent qu'attester que le système était dans un état donné à un moment précis dans le passé. Par exemple, si le premier stade de démarrage est dans une Read-Only Memory (ROM), celui-ci peut faire office de racine de confiance statique en vérifiant l'intégrité du système avant de le lancer. Suivant le même principe, toute implémentation de démarrage sécurisé est une forme de racine de confiance statique. Cela garantit qu'au démarrage, le système n'était pas compromis, mais ne donne aucune assurance sur l'état du système durant son exécution. Une racine de confiance statique ne protège donc pas contre les attaques qui surviennent après le démarrage ni contre les attaques qui ne modifient pas le code, comme le ROP.

À l'opposé, une racine de confiance dynamique peut attester de l'intégrité de données ou de l'état du système durant son exécution, ce qui permet de contrer des attaques ayant lieu après le démarrage. Les racines de confiance dynamique sont généralement implémentées par la création d'un environnement isolé, comme un TEE. De plus, si une racine de confiance dynamique possède des composantes logicielles, il peut être bénéfique d'en vérifier l'intégrité au démarrage grâce à une racine de confiance statique.

Les racines de confiance peuvent prendre la forme d'une composante matérielle ou être complètement implémentées en logiciel. Les formes matérielles sont généralement plus résistantes aux attaques, mais sont en contrepartie plus coûteuses. Ces composantes matérielles sont appelées des modules de sécurité matériels, ou Hardware Security Module (HSM). Il en existe une grande variété, allant du TPM aux cartes à puces, en passant par les YubiKeys [69]. Les implémentations logicielles sont généralement basées sur des programmes écrits dans une ROM. Cependant, sous cette forme, elles ne peuvent qu'être statiques. Pour être dynamiques, elles doivent au minimum être accompagnées d'un mécanisme sécuritaire de génération de semence.

3.1.1 Trusted Platform Module

Le TPM est une puce spécialisée permettant de générer, emmagasiner et utiliser des clés cryptographiques de manière sécuritaire. Les clés cryptographiques ne peuvent jamais sortir de la puce et peuvent seulement être utilisées à l'intérieur de celle-ci. Elle peut aussi calculer des hachés et les utiliser pour sceller et desceller des clés [70]. Il est ainsi possible de construire toute une arborescence de clés, permettant de segmenter l'accès à des données scellées. L'action de sceller et desceller correspond à cacher des données aux yeux de tous les utilisateurs d'un système et de ne laisser que l'entité qui en est propriétaire les lire. Cela revient généralement à chiffrer les données avec une clé à laquelle seule l'entité propriétaire

a accès.

Les fonctionnalités du TPM incluent aussi la capacité à générer des nombres aléatoires et de construire des hachés incrémentaux dans ses Platform Configuration Register (PCR). Les PCR sont des espaces mémoire sur le TPM qui ont la propriété de ne jamais pouvoir être changés pour une valeur donnée. Ils ne peuvent être modifiés qu'en apposant des hachés, c'est-à-dire en calculant un haché d'une donnée concaténée à la valeur précédente contenue dans le PCR. Ce mécanisme en fait un excellent candidat pour implémenter du démarrage de confiance.

À priori, le TPM est une racine de confiance statique. Cependant, différentes tactiques parviennent à utiliser ses primitives cryptographiques pour construire des environnements sécurisés. Les appareils qui n'ont pas de TPM physique sur leur circuit peuvent aussi bénéficier de ses fonctionnalités grâce aux TPM virtuels (vTPM) qui sont implémentés complètement en logiciel. Ceci est particulièrement pratique pour les environnements infonuagiques où de nombreuses machines virtuelles nécessitent des TPM.

3.1.2 Fonction physique inclonable

Une fonction physique inclonable, ou Physical Unclonable Function (PUF), est une méthode de génération de semence aléatoire qui repose sur les caractéristiques physiques d'une composante matérielle. La PUF doit pouvoir générer toujours la même semence sans pour autant qu'un observateur extérieur puisse la déduire. Les propriétés recherchées sont donc l'unicité, afin que chaque semence soit unique, la robustesse, afin que la même PUF génère toujours la même semence et la résistance aux attaques physiques. Les puces Static Random Access Memory (SRAM) ont naturellement des imperfections qui créent du bruit aléatoire lors de leur mise en marche. Les valeurs initiales des bits peuvent être utilisées pour générer une semence aléatoire [71]. Des composantes comme la SRAM sont des candidats attrayants pour implémenter un PUF, car beaucoup d'appareils en possèdent déjà. Des chercheurs ont toutefois réussi à cloner une PUF sur une puce SRAM afin d'en extraire la semence [72].

3.1.3 Autres racines de confiance

Lorsqu'un mécanisme de démarrage sécurisé est en place, la racine de confiance peut se réduire à une simple clé cryptographique. Il faut alors qu'un mécanisme comme un TEE soit en place pour assurer que seules les applications autorisées y ont accès. La clé peut être mise dans une Battery-Backed Random Access Memory (BBRAM) ou dans un fusible électronique (eFuse), mais cela n'est pas idéal [71]. La BBRAM nécessite une batterie pour

garder son contenu et le fusible n'est pas modifiable. De plus, ces technologies permettent à quiconque ayant un équipement adapté de lire directement la clé sur l'appareil. Finalement, elles ne peuvent faire office de racine de confiance sans une méthode externe de génération de nombres aléatoires. Cela enlève donc beaucoup de flexibilité à ces variantes de racine de confiance.

Certaines implémentations utilisent plutôt un coprocesseur qu'une puce externe comme racine de confiance, ce qui est pratique pour des petits appareils embarqués [73]. Cependant, cela requiert une architecture très spécifique et peu courante.

3.2 Architectures matérielles

Le Trusted Computing est seulement possible grâce aux technologies matérielles et cryptographiques modernes. En effet, les mesures de sécurité purement logicielles sont toujours menacées par des failles, tandis que des mesures soutenues par des architectures matérielles proposent des défenses beaucoup plus difficiles à pénétrer depuis l'environnement logiciel.

La popularisation du Trusted Computing est donc grandement corrélée à la démocratisation de technologies comme le TPM. De plus, certains fabricants de processeurs ont décidé d'intégrer des technologies d'isolation matérielle directement dans leurs processeurs. Cela permet à leurs utilisateurs de bénéficier de la sécurité du Trusted Computing sans avoir à ajouter des composants supplémentaires à leurs produits. Les technologies les plus notables sont Intel Software Guard Extension (SGX) pour l'architecture x86 et ARM TrustZone pour l'architecture ARM. Il est à noter qu'un système d'exploitation ordinaire peut toujours rouler sur ces processeurs sans faire utilisation des propriétés spéciales d'isolation.

Maene et al. [74] décrivent 7 caractéristiques souhaitables pour une architecture matérielle basée sur le Trusted Computing.

- Isolation du code et des données
- Attestation envers un parti externe
- Scellage de code et de données
- Racine de confiance dynamique
- Confidentialité du code
- Résistance aux attaques par canaux dérivés
- Protection de la mémoire, incluant les bus et la mémoire externe

Les architectures décrites dans cette section sont le fruit d'énormément de recherche et de développement et répondent à tous ces critères.

3.2.1 Intel SGX

Intel SGX permet la création de conteneurs isolés appelés enclaves. Ce sont des zones de mémoire privées, accessibles uniquement par un seul processus. Des instructions CPU et mécanismes d'accès à la mémoire spéciaux permettent l'isolation complète des enclaves du reste du système. Cela permet à plusieurs services sensibles de rouler de manière totalement isolée sur le même système, ce qui est bien adapté aux environnements infonuagiques ou plusieurs utilisateurs doivent être séparés [75].

3.2.2 ARM TrustZone

ARM TrustZone utilise des gestionnaires de mémoire pour partitionner la mémoire physique et créer un environnement sécuritaire appelé Monde Sécurisé, ou Secure World (SW). Celui-ci est totalement isolé du Monde Normal, ou Normal World (NW), où un système d'exploitation ordinaire peut résider. Un bit supplémentaire a été ajouté au processeur pour différencier des 2 mondes. Le processeur ne peut donc se trouver que dans un seul monde à la fois et cette séparation est soutenue au niveau matériel.

L'implémentation de TrustZone diffère légèrement entre les processeurs de la famille Cortex-A et ceux de la famille Cortex-M. Les premiers sont destinés à un usage général et sont généralement plus puissants. Leur isolation est un peu plus robuste, car toute communication entre les 2 mondes doit passer par un logiciel superviseur s'exécutant avec des privilèges supérieurs à ceux des 2 mondes. Le transfert d'un monde à l'autre est déclenché par l'appel à des instructions spécialisées du processeur. Les derniers sont plutôt destinés à des appareils embarqués à faible puissance. L'isolation est donc moins robuste par souci de performance. En effet, les applications du Monde Normal peuvent communiquer directement avec celles du Monde Sécurisé et le changement d'un monde à l'autre se fait automatiquement dans l'exécution des routines d'interruption [76].

Ce projet s'intéresse exclusivement à l'architecture Cortex-A, plus adaptée aux routeurs. Les plateformes ARM ont plusieurs niveaux d'exception, ou Exception Level (EL). Concrètement, ils correspondent aux niveaux de privilèges avec lesquels un programme peut s'exécuter. Ils sont numérotés de 0 à 3, 3 étant le plus haut niveau de privilèges. L'architecture ne spécifie pas quel morceau logiciel doit s'exécuter avec quel EL, mais cette séparation est cruciale pour le Trusted Computing. Dans le cas de TrustZone pour Cortex-A, le niveau EL3 est réservé au logiciel superviseur assurant la communication entre les 2 mondes.

Un inconvénient de cette architecture est que tous les processus sensibles se partagent le Monde Sécurisé. Il est donc primordial de garder le nombre et la complexité de ces processus

au minimum. TrustZone est bien adapté aux appareils embarqués, puisqu'il ont généralement peu de processus sensibles et moins de ressources. La vérification des programmes sensible durant leur exécution peut aider à améliorer encore plus le niveau de sécurité [71].

3.3 Démarrage sécurisé

Le démarrage sécurisé, ou Secure Boot, est une méthode de vérification de l'intégrité des différents stades de démarrage. Il repose sur la cryptographie asymétrique. Les images binaires de chaque étape sont signées cryptographiquement avec la clé privée d'un tiers de confiance avant d'être mises sur l'appareil. Lors du démarrage, chaque stade vérifie la signature du stade suivant avant de l'exécuter. La vérification est faite à l'aide de la clé publique du tiers de confiance, qui doit être placée sur l'appareil sur une puce en lecture seule afin d'éviter qu'elle soit modifiée par un attaquant. Cela restreint énormément les développeurs indépendants, car ils doivent faire signer leurs images par un tiers de confiance lors de la production.

3.4 Démarrage de confiance

Le démarrage de confiance, ou Trusted Boot, est une technologie qui permet aussi d'assurer l'intégrité des différents stades de démarrage. Cependant, plutôt que vérifier la signature cryptographique d'un stade avant de l'exécuter, le démarrage de confiance fait un haché du binaire en question, étend le dernier haché avec celui-ci et le compare à une banque de hachés déjà calculés. Si le haché ne correspond pas, le processus de démarrage est arrêté et une alerte est lancée. Le principe d'étendre un haché revient en quelque sorte à concaténer un haché avec les nouvelles valeurs à hacher avant de faire le calcul. De cette manière, non seulement le binaire doit être intouché, mais tous les stades précédents doivent eux aussi être valides pour qu'un certain haché soit correct.

L'objectif est similaire à celui du démarrage sécurisé, mais les fondements en sont différents. La première différence est sur quelle entité repose la confiance. Tandis que la sécurité du démarrage sécurisé repose sur une Autorité de Certification qui doit valider et signer les composantes, le démarrage de confiance donne une plus grande flexibilité en permettant à l'utilisateur de calculer et d'installer lui-même de nouveaux hachés qui seront utilisés lors du prochain démarrage. Cela permet à des développeurs ou des utilisateurs avancés d'installer leur propre version d'un microprogramme ou d'un système d'exploitation tout en continuant de bénéficier de la protection offerte par le démarrage de confiance.

Les 2 systèmes peuvent aussi être utilisés de manière conjointe. Les tout premiers stades du démarrage sont souvent émis par le fabricant du matériel et sont peu prompts à changer.

Disposants aussi de moins de ressources, ils sont donc mieux adaptés au Secure Boot. Les stades plus tardifs peuvent toutefois passer au Trusted Boot pour garder sa flexibilité.

La plupart des implémentations du démarrage de confiance utilisent les PCR d'un TPM pour étendre et stocker les hashés cryptographiques. L'avantage des PCR est qu'ils supportent seulement le calcul d'un hashé par son extension, ce qui empêche un attaquant de réinitialiser la chaîne de confiance en calculant un hashé qui ne tient pas compte des valeurs précédentes.

3.5 ARM Trusted Firmware

Afin d'aider et d'encourager les développeurs à bâtir des systèmes sécuritaires, la compagnie ARM maintient un micrologiciel sécurisé appelé ARM Trusted Firmware [77]. Celui-ci est utilisé par la grande majorité des TEE compatibles avec des plateformes d'architecture ARM. Sur ces plateformes, le processus de démarrage est divisé en plusieurs étapes, nommés BL pour *Boot Loader stage* [78].

- BL1 : Micrologiciel du manufacturier dans une ROM
- BL2 : Micrologiciel de démarrage de confiance
- BL3-1 : Micrologiciel sécurisé aux privilèges EL3
- BL3-2 : Micrologiciel sécurisé aux privilèges EL1 (facultatif)
- BL3-3 : Micrologiciel de démarrage non digne de confiance

Le ARM Trusted Firmware occupe le rôle du BL2. Il s'occupe d'initialiser la plateforme et d'exécuter le BL3-1, qui est l'image du Monde Sécurisé du TEE. Le BL3-3, exécuté plus tard dans la chaîne, est un micrologiciel de démarrage standard, comme U-Boot [22]. Le ARM Trusted Firmware s'assure de bien implémenter le démarrage de confiance [79]. La racine de confiance consiste en une copie de la Root of Trust Public Key (ROTPK), placée dans un registre spécial, et en l'image du premier stade de démarrage, soit le BL1. La chaîne de confiance est construite à l'aide de plusieurs clés cryptographiques. La ROTPK signe l'image du BL2, une clé dite de confiance signe les images BL3-1 et BL3-2, puis une clé non de confiance signe le BL3-3. Les signatures sont calculées sur le hashé de l'image et sont placées avec ceux-ci dans des champs personnalisables de certificats suivant le standard X.509 v3. Pour maintenir la chaîne de confiance, chaque image calcule le hashé de l'image suivante, le compare avec celui emmagasiné et vérifie la signature correspondante. Le processus de démarrage est interrompu si la moindre de ces vérifications échoue.

3.6 Trusted Execution Environment

Les environnements d'exécution de confiance, ou TEE, sont le produit de l'application du Trusted Computing à un système d'exploitation afin de créer un milieu où tous les programmes et leurs résultats sont de confiance. Un TEE utilise les propriétés et les instructions spéciales des architectures de CPU modernes afin de créer un environnement isolé du reste du système. De plus, tous les logiciels qui y roulent sont vérifiés depuis le premier stade de démarrage, car un seul composant infecté pourrait compromettre l'intégrité du TEE. C'est pourquoi un système qui implémente un TEE doit utiliser du démarrage sécurisé et/ou du démarrage de confiance. Il est aussi important de minimiser le nombre et la complexité des logiciels qui en font partie, car une seule faille pourrait en compromettre la sécurité.

Un TEE s'exécute généralement en conjonction avec un autre système non sécurisé. Une des raisons est que pour garantir sa sécurité, le TEE se doit de rester le plus petit et simple possible. Il n'implémente donc pas les fonctionnalités courantes des systèmes tout usage comme la réseautique ou les interfaces utilisateur. La plupart des TEE n'ont même pas de système de fichiers. Toutes ses composantes qui sont si familières sont en réalité extrêmement complexes et encore plus difficiles à sécuriser. C'est pourquoi les TEE font appel à leur contrepartie non sécurisée pour effectuer ce genre d'opérations. Sur ARM TrustZone, le TEE réside dans le Monde Sécurisé tandis que le système d'exploitation non digne de confiance s'exécute dans le Monde Normal.

Des exemples d'implémentations de TEE sont discutés dans la section 4.2.2.

Il ne faut pas penser que les TEE sont à l'abri des vulnérabilités. Bien que la plupart essayent de réduire la taille de leur TCB au maximum, des failles peuvent tout de même s'y glisser. Les mêmes familles de vulnérabilités qui touchent les exécutables du Monde Normal, comme les débordements de tampon ou les doubles libérations, peuvent affecter les Applications de Confiance du Monde Sécurisé [80]. De plus, la complexité inhérente à l'établissement d'un environnement sécurisé peut entraîner des trous sémantiques pouvant être exploités. En 2017, Machiry et al. [39] ont trouvé un tel trou dans le mécanisme de communication entre les 2 Mondes. Les Applications de Confiance ne peuvent pas toujours vérifier les permissions du Monde Normal des régions en mémoire qui leur sont fournies en argument. Cela permet à un programme non privilégié du Monde Normal de demander une opération sur une zone de mémoire privilégiée. Lorsque le contenu écrit en mémoire par l'Application de Confiance peut être contrôlé, par exemple lors d'une opération de déchiffrement, cela permet au programme non privilégié d'élever ses privilèges sur le système d'exploitation du Monde Normal. Les 4 implémentations de TEE qu'ils ont évaluées étaient affectées, mettant à risque des millions

d'appareils. Toutefois, dans le cadre de ce projet, nous assumons que les implémentations de TEE sont adéquatement sécurisées.

Une autre fonctionnalité importante des TEE est la capacité à prouver son intégrité à une tierce partie [23]. Cette fonction s'appelle l'attestation et il existe de nombreuses méthodes pour y arriver. C'est là un grand avantage des TEE par rapport à la simple utilisation d'un HSM. Le HSM peut générer, stocker et utiliser des secrets cryptographiques de manière sécuritaire. Cependant, il n'a aucune notion de l'état du système, puisque c'est une composante externe. Le TEE, au contraire, a une vision privilégiée de l'ensemble du système et est en mesure d'évaluer si les secrets devraient être utilisés. C'est donc une manière de protéger ces secrets contre des attaques logicielles qui auraient permis à des acteurs malveillants de prendre le contrôle du côté non digne de confiance de l'appareil.

3.7 Attestation

L'attestation est un principe par lequel un *Prouveur* donne à un *Vérificateur* une preuve de son état. Plus précisément, il doit prouver que son système est dans un état voulu prédéterminé. Cet état peut inclure plus ou moins de composantes du système. Le protocole d'attestation doit faire en sorte que seul le *Prouveur* peut créer une preuve valide, de sorte qu'un attaquant ne peut la reproduire artificiellement, et que tout changement par rapport à l'état prédéterminé produit un changement dans le résultat de la preuve, la rendant invalide. L'attestation de l'intégrité d'un système peut être soit statique, donc calculée au démarrage, ou dynamique, donc calculée durant son exécution.

Dans [81], Francillon et al. identifient 5 caractéristiques minimales d'un système pour qu'il soit capable d'attester de manière sécuritaire de son état. Ces caractéristiques reposent sur le fait qu'un seul élément de donnée (par exemple, une clé cryptographique) doit rester secret pour assurer que seul le *Prouveur* est en mesure de générer une preuve valide.

- **Accès exclusif** au secret par le *Prouveur*
- **Aucune fuite** d'information permettant de retrouver le secret ne doit se produire pendant ou après que le *Prouveur* génère une preuve
- **Immutabilité** du code du *Prouveur*
- **Ininterruptabilité** de la routine du *Prouveur* qui génère une preuve en fonction de l'état du système
- **Invocation contrôlée** de la routine de génération de preuve à partir du point d'invocation prévu

À partir de ces caractéristiques, ils déduisent des propriétés matérielles minimales qu'un appareil doit posséder pour implémenter un protocole d'attestation sécuritaire. Leur article se

veut une généralisation qui ne correspond pas à un type d'appareil en particulier. Or, un appareil qui possède un processeur ARM doté des capacités de TrustZone répond automatiquement à tous ces critères.

L'attestation n'est pas un problème résolu et beaucoup de recherche est faite pour trouver de meilleures méthodes. Certaines tactiques se basent sur une composante matérielle comme le TPM, alors que d'autres sont implémentées totalement en logiciel et certaines ont même une forme hybride [82].

Ling et al. [83] tirent parti du haut niveau de privilège des processus roulants dans le Monde Sécurisé de ARM TrustZone pour attester de l'état du Monde Normal. Ils proposent une Application Sécurisée qui vérifie que tous les processus du Monde Normal s'exécutent sans avoir été modifiés. Pour ce faire, elle accède à la mémoire du Monde Normal et lit les structures de données du noyau Linux qui y réside. Tous les processus possèdent une structure de type *task_struct* correspondante dans le noyau Linux, qui contient leurs métadonnées. Ces structures sont liées par une liste doublement chaînée qui permet de facilement les itérer. Le symbole *init_task* est situé à une adresse fixe et pointe vers la structure du processus initial. L'Application sécurisée utilise ces principes pour générer un haché de toutes les sections comportant du code de tous les processus en exécution. Ces hachés sont ensuite sécuritairement transmis à un serveur, qui peut les comparer à une liste de hachés calculés à partir des exécutables présents sur l'appareil. Le serveur peut alors détecter si un programme a été modifié ou ajouté sur l'appareil distant. Wehbe et al. [84] proposent une solution similaire, mais qui se veut plus autonome. En effet, leur architecture garde les hachés du code des exécutables dans un module de sécurité externe attaché au processeur. Ce module calcule ensuite un haché du code de chaque application à son lancement afin d'y détecter des modifications. Finalement, une vérification périodique est aussi effectuée afin de contrer les modifications des applications après leur lancement. DRIVE [85] utilise aussi une stratégie similaire, mais sans recourir à des hachés calculés en avance sur l'exécutable. Les auteurs proposent un module dans le noyau Linux qui vérifie périodiquement l'intégrité de tous les processus roulant sur le système. Les mesures prises sont seulement comparées avec les dernières mesures prises pour un processus en particulier. Une autre particularité de DRIVE est d'étendre la vérification à des sections dynamiques de l'exécutable comme la Global Offset Table (GOT). Cela requiert la résolution des symboles afin d'évaluer si le contenu de la GOT est légitime.

Malheureusement, ces techniques ne sont pas parfaites, car un attaquant pourrait potentiellement effacer les traces de ses modifications juste avant la vérification. Wan et al. [86] implémentent une telle attaque, qu'ils nomment TZ-Evasion. Leur technique repose sur l'exécution d'un processus de détection à très haute priorité sur chaque coeur du processeur. Ces

processus envoient un signal régulièrement et lorsqu'ils ne le font pas, il est assumé que ce coeur est utilisé par le Monde Sécurisé. Cela donne alors une chance à TZ-Evasion d'effacer les modifications qu'il a faites au système et de les refaire lorsque tous les coeurs sont revenus dans le Monde Normal. Les auteurs décrivent aussi une défense contre cette technique, qu'ils nomment SATIN. La défense repose principalement sur l'exécution plus fréquente et sur des coeurs aléatoires de la vérification de l'intégrité des processus.

CHAPITRE 4 IMPLÉMENTATION D'UN PROTOTYPE

4.1 Élaboration d'une architecture

L'objectif de ce projet est de créer une architecture de routeur domestique répondant à plusieurs critères. Tout d'abord, la surface d'attaque logicielle doit être minimisée. Puisque la plupart des vulnérabilités affectant les routeurs domestiques exploitent l'interface web de configuration, utiliser une suite logicielle qui n'en possède tout simplement pas est un bon point de départ. De plus, cette suite ne devrait utiliser aucun mot de passe fixe pour toutes ses fonctionnalités et devrait être facile à installer et à utiliser. C'est le cas du système SentinelC, dont le fonctionnement est détaillé à la section 4.2.1. La configuration avec ce système se fait sur un serveur en ligne auquel le routeur se connecte périodiquement pour recevoir des mises à jour. Au premier abord, la configuration sur une interface en ligne plutôt que sur une interface située sur l'appareil peut sembler un simple déplacement du problème. Cependant, cela est réellement plus sécuritaire pour plusieurs raisons. Tout d'abord, si le serveur est compromis par un acteur malveillant, l'emprise de celui-ci sur les routeurs sera limitée aux fonctionnalités en place, alors qu'il aurait un contrôle total s'il avait compromis les appareils directement. Ensuite, l'utilisation d'un serveur central permet de concentrer les efforts de sécurisation de la part des développeurs au lieu d'avoir une multitude d'appareils, d'interfaces et d'architectures différentes à sécuriser. Finalement, puisque notre solution doit avoir des mises à jour automatiques, l'architecture doit dans tous les cas inclure un serveur digne de confiance. L'utilisation de ce serveur pour configurer les appareils n'ajoute donc pas réellement de surface d'attaque.

Le système SentinelC prévoit que le routeur doit s'authentifier auprès du serveur avec un jeton secret partagé par les 2 entités. Dès lors, il est important que ce jeton soit protégé dans le cas où un acteur malveillant réussirait à s'introduire dans le routeur par une autre voie. Le jeton constitue donc une donnée sensible que les propriétés du Trusted Computing pourraient aider à isoler. Ce n'est pas la seule option envisageable, mais certainement une des plus robustes. L'isolation du jeton dans un TEE serait une manière de minimiser l'impact des brèches logicielles et des attaques par chaîne d'approvisionnement. Le jeton pourrait être gardé dans un HSM, ce qui le rendrait impossible à dévoiler. Cependant, cela ne ferait que déplacer le problème, car un HSM ne peut pas différencier une demande d'accès au jeton légitime d'une demande illégitime. Un des avantages de la création d'un environnement isolée par le Trusted Computing est la capacité des processus sécurisés à accéder à la partie non sécurisée du système pour vérifier la légitimité d'une demande d'accès. Afin de créer un

routeur exemplaire, le jeton doit être aléatoire et différent pour chaque installation. De plus, il ne doit en aucun cas sortir de l’environnement isolé. Il doit donc seulement être utilisé par les processus roulant à l’intérieur de celui-ci. L’authentification au serveur doit alors se faire à l’aide d’un protocole plus sophistiqué que le simple envoi du jeton, comme le protocole défi-réponse. Aussi, les processus utilisant le jeton doivent attester de l’intégrité du système avant d’exécuter une requête. Naturellement, le nombre et la complexité des processus s’exécutant dans l’environnement sécurisé doivent être minimisés afin de réduire la surface d’attaque. De plus, il faut que ces applications dépendent le moins possible de bibliothèques provenant de tierces parties pour éviter les brèches par chaîne d’approvisionnement. Finalement, du côté de l’expérience utilisateur, le matériel choisi doit être abordable pour un consommateur moyen. Le routeur doit ne pas demander aucune configuration par son propriétaire ; il doit suffir de le brancher pour pouvoir s’y connecter avec tous ses appareils. Le mot de passe par défaut du Wifi doit être unique et sécuritaire, comme celui que l’utilisateur devra mettre en place pour accéder au serveur de configuration. Une première ébauche de la solution est présentée à la figure 4.1.

Les caractéristiques de cette architecture seront sujettes à des changements à mesure que de nouvelles contraintes seront déterminées. Les détails de l’implémentation du prototype sont exprimés dans les sections qui suivent.

4.2 Choix des technologies

La première étape en vue de créer un prototype de routeur sécurisé est le choix des technologies existantes sur lesquelles le bâtir. Puisque ce projet de recherche vise à créer un prototype de routeur sécurisé à l’aide du Trusted Computing, il nécessite évidemment une plateforme matérielle faisant office de routeur. Il requiert aussi une technologie de Trusted Computing compatible avec la plateforme pour établir un TEE sur lequel reposeront les fondations de la sécurité ajoutée. Toutefois, la plateforme ne peut faire office de routeur sans une couche logicielle pour offrir les fonctionnalités propres à un routeur. La plupart des routeurs disponibles sur le marché ont des couches logicielles propriétaires à source fermée. Il existe des initiatives de micrologiciel de routeur à source ouverte comme DD-WRT [87], Tomato [88] ou OpenWrt [89]. Cependant, elles n’offrent pas de mesures de sécurité additionnelles par rapport aux micrologiciels de routeur standards, dans lesquels de multiples vulnérabilités sont trouvées chaque année. La création d’un routeur sécurisé mérite l’utilisation d’un micrologiciel mettant l’accent sur la sécurité de ses applications. Un bon représentant des routeurs qui donnent une attention particulière à la sécurité est le routeur SentinelC. SentinelC est encore au stade de développement, mais ses spécifications en font un très bon candidat pour l’ajout

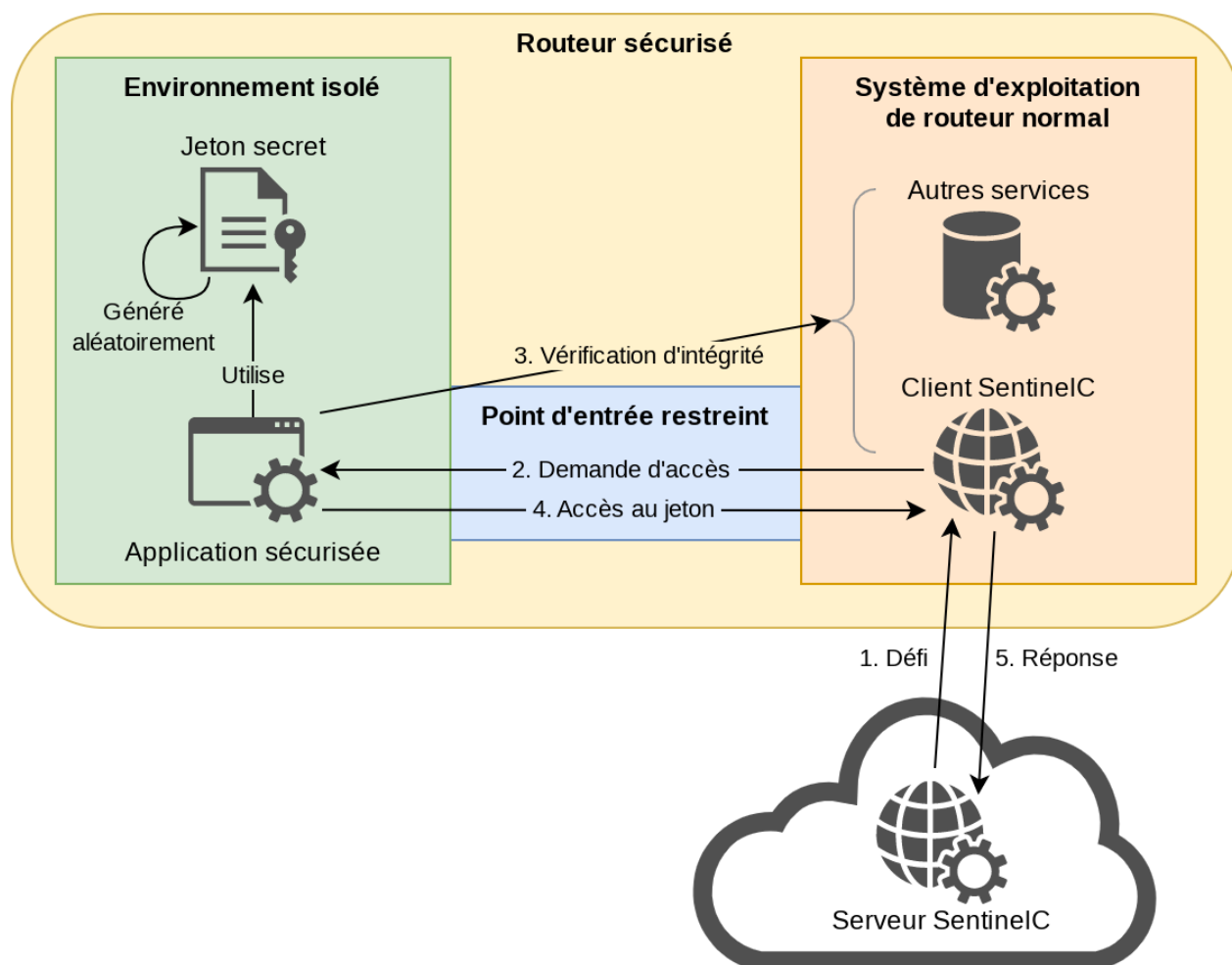


FIGURE 4.1 Architecture initiale du routeur sécurisé

de Trusted Computing. Nous avons établi un partenariat avec leur équipe et c'est pourquoi nous allons utiliser le code de SentinelC pour la création du prototype.

4.2.1 SentinelC

SentinelC est une solution de gestion de routeurs centralisée et sécurisée. Elle s'adresse aux petites et moyennes entreprises, bien qu'elle puisse aussi bien être utilisée par des particuliers. Son objectif est de simplifier la gestion d'une grande flotte d'appareils en rassemblant toutes les configurations individuelles en un seul endroit. La solution comprend un système d'exploitation de routeur personnalisé, appelé SentinelC OS, ainsi qu'un serveur central permettant d'avoir une vue d'ensemble sur les différents réseaux gérés.

Serveur central

Le serveur central (*Cloud Controller*) contient l'interface d'administration et peut être hébergé sur n'importe quel serveur accessible à tous les appareils. Plusieurs fonctionnalités sont offertes par défaut, comme l'ajout automatique de sécurité TLS à la connexion grâce au service *Let's Encrypt*. De plus, le module *Wireguard* du noyau Linux peut être activé pour permettre la connexion à travers un Virtual Private Network (VPN). Le serveur central est composé de quelques éléments, qui sont chacun disponibles sur leur propre sous-domaine.

- `api` : Application Programming Interface (API) pour distribuer les configurations
- `app` : Interface web principale d'administration
- `accounts` : API Representational State Transfer (REST) pour la gestion des utilisateurs
- `portal` : Page de portail captif utilisé dans certaines configurations réseau

Depuis l'interface web, les administrateurs peuvent configurer chaque routeur individuellement ou en groupes personnalisés. Les routeurs sont d'abord séparés par endroit physique, puis en différentes zones. Concrètement, les zones sont des Virtual Local Area Network (VLAN) qui peuvent être attribués aux différents ports des appareils. Chaque zone contient un nombre arbitraire de routeurs et peut avoir sa propre configuration. Le Wifi peut prendre la forme d'un réseau ouvert, d'un réseau privé sécurisé par un mot de passe ou d'un réseau semi-privé avec une authentification sur un portail captif. L'accès aux zones peut être fragmenté selon plusieurs utilisateurs afin d'augmenter la sécurité, de permettre une meilleure modularité et même de déléguer complètement la gestion d'une zone à une tierce partie. Il est aussi possible de lancer des tests diagnostiques sur certaines parties du réseau, par exemple pour en capturer le trafic, faire un balayage de ports ou identifier de potentiels wifi malveillants qui tentent de piéger les utilisateurs. La configuration des appareils prend la forme d'un fichier au format JavaScript Object Notation (JSON) comprenant plusieurs champs prédéterminés. Ce fichier permet de modifier les zones, les canaux VLAN et le type de réseau Wifi. Il est aussi possible d'ajouter une clé publique SSH dans le fichier `authorized_keys` de l'utilisateur root des appareils directement depuis l'interface web, ce qui permet de s'y connecter et facilite le diagnostic de potentielles erreurs.

Système d'exploitation SentinelC

La seconde partie de SentinelC est le système d'exploitation personnalisé, basé sur le noyau Linux, qui est placé sur les appareils de la flotte. L'image binaire est construite à l'aide de Yocto, qui fonctionne en assemblant des couches (*layers*) les unes par-dessus les autres. Les premières couches font directement affaire avec le matériel, ce qui permet aux couches supérieures d'en faire abstraction et de travailler avec un noyau Linux complet. Cela rend

possible la réutilisation de code entre différents projets tant qu'il existe des couches inférieures équivalentes. Les fichiers spécifiques au système d'exploitation de SentinelC sont contenus dans une seule couche. Celle-ci est construite au-dessus de plusieurs couches au code ouvert. Le micrologiciel de démarrage utilisé est U-Boot. Le système utilise Mender pour effectuer ses mises à jour. Mender est contenu dans une couche Yocto et fonctionne en séparant l'espace disque en 4 partitions.

- La partition de démarrage
- La partition système A
- La partition système B
- La partition de stockage persistant

Les 2 partitions système sont identiques, mais une seule est en utilisation à la fois. Elles servent plutôt de sauvegarde l'une de l'autre, afin de minimiser les chances qu'un appareil ne fonctionne plus. Puisque les routeurs peuvent être dispersés dans de nombreux endroits, il faut éviter de devoir aller les réparer sur place, et le fait d'avoir une partition de secours qui permet une connexion SSH peut aider à régler certains problèmes. Lorsque le serveur central indique au routeur qu'il a une mise à jour de micrologiciel, le client Mender la télécharge et l'écrit directement par-dessus la partition système qui n'est pas actuellement en utilisation. Afin de sauver de l'espace en mémoire, la signature de l'image est calculée au fur et à mesure qu'elle est écrite. Finalement, si la signature calculée coïncide avec celle voulue, Mender change dans la configuration de U-Boot le numéro de la partition à démarrer et redémarre l'appareil. Sinon, une erreur est lancée. Ce processus permet d'avoir un temps d'inactivité minimal lors d'une mise à jour du système, soit seulement le temps de redémarrer. Cependant, pour pouvoir l'utiliser, il faut que tout le système soit contenu dans une seule image. Le système d'exploitation SentinelC n'a donc pas de gestionnaire de paquets. Chaque version prend la forme d'une image contenant tous les fichiers et tous les programmes nécessaires à son fonctionnement. Les partitions système sont montées en lecture seule. Cela permet une plus grande stabilité, mais empêche beaucoup d'applications de fonctionner normalement. Par exemple, de nombreux programmes comptent sur le fait qu'ils peuvent mettre leurs données dans les dossiers */var*, */etc* ou */tmp*, et lanceront des erreurs si ce n'est pas le cas. C'est pourquoi un système de fichier *overlayfs* est utilisé par-dessus, ce qui en permet l'écriture. Toutefois, ces modifications sont temporaires et ne résisteront pas un redémarrage. Seule la partition de stockage persistant permet l'écriture de données qui ne seront jamais écrasées par les mises à jour du système. Elle est montée sur le dossier */data*.

Le client SentinelC est un service géré par *systemd*. Les mises à jour fonctionnent sur un principe de scrutation. Le client contacte le serveur central chaque minute pour demander si une nouvelle configuration est disponible. Cependant, à tout moment, le serveur central

peut envoyer un paquet Internet Control Message Protocol (ICMP) spécial qui accélère la prochaine scrutation d'un appareil. Cela permet une application presque instantanée de la configuration. Le client se connecte avec le protocole Hypertext Transfer Protocol Secure (HTTPS) par souci de sécurité et le certificat public du serveur peut être mis directement sur le système de fichier des routeurs comme mesure additionnelle. Chaque appareil possède un jeton unique dont il se sert pour s'authentifier auprès du serveur. Il est simplement transmis à l'API en texte clair à chaque requête. Le jeton est distribué aux appareils lors de la procédure d'enrôlement.

Enrôlement

L'enrôlement des routeurs fonctionne avec le principe du *Trust on First Use*. Cela veut dire que la première connexion au serveur n'est pas sécurisée, mais elle doit être supervisée par un opérateur et sert de base de confiance pour les connexions subséquentes. En premier lieu, le serveur central doit être configuré et accessible depuis un nom de domaine. Ensuite, un opérateur doit allumer le routeur et s'y connecter via un câble Ethernet ou une connexion Wifi. Lorsqu'il n'est pas enrôlé, un routeur avec SentinelC OS lance au démarrage un réseau Wifi publique, un serveur Dynamic Host Configuration Protocol (DHCP) et une interface web pour la configuration. L'opérateur peut s'y connecter et y entrer l'URL du serveur central. Le routeur va se connecter au serveur central, lui fournir son jeton d'authentification et redémarrer. À partir du moment où il est enrôlé, le routeur ne va plus lancer de réseau Wifi publique ou d'interface web au démarrage. Il va simplement aller chercher sa configuration et commencer ses opérations normales.

Sécurité

Le routeur SentinelC ne met intentionnellement pas d'interface web de configuration disponible, car c'est un vecteur d'attaque connu et très populaire. Malgré que le système SentinelC soit en avance en termes de sécurité par rapport aux solutions de routeurs ordinaires trouvés communément dans les foyers et les entreprises, il n'est pas non plus infailible. Une de ses lacunes principales est le manque de sécurité physique des appareils sur le terrain. Dans une situation où un routeur SentinelC doit être déployé dans un environnement non contrôlé où il est accessible physiquement (une école, un bureau, etc.), un attaquant pourrait interagir physiquement avec le routeur pour perturber le système. Par exemple, puisque la séquence de démarrage n'est pas sécurisée, un individu ayant accès au routeur peut utiliser le menu du micrologiciel de démarrage pour extraire la mémoire de l'appareil ou pour installer un nouveau micrologiciel entièrement. Comme indiqué plus haut, toute la confiance que le serveur donne

aux appareils déployés repose sur leur possession du jeton échangé lors de l'enrôlement. Si un attaquant est en mesure de lire ce jeton, il sera capable de faire passer n'importe quel appareil pour le routeur compromis. Dans le cas où un autre appareil se fait passer pour le routeur original auprès du serveur, il pourrait y avoir plusieurs conséquences néfastes. Premièrement, l'appareil original pourra être volé, détruit ou détourné de ses fonctions (pour miner de la cryptomonnaie, par exemple), sans que le détenteur légitime ne s'en rende compte. Deuxièmement, les fonctionnalités de diagnostic fournies par le routeur pourraient être faussées ou activées sans autorisation. Par exemple, un opérateur peut demander à un routeur de lancer un balayage du réseau pour obtenir une liste des appareils connectés. Un attaquant serait alors en mesure de retourner de faux résultats pour camoufler une autre opération sur ce réseau. La seule manière de remédier à ces conséquences est d'inspecter l'appareil sur place, ce qui peut être compliqué s'il y est dans un emplacement distant ou si les attaquants ont ciblé une grande quantité de routeurs. Ces lacunes dans la sécurité de SentinelC sont tout à fait dans la mire du Trusted Computing.

4.2.2 Trusted Execution Environment

Plusieurs projets proposent des implémentations de Trusted Execution Environment. Nous les avons explorés afin de déterminer laquelle convient le mieux au prototype de routeur sécurisé. Toutes les solutions sont compatibles avec ARM TrustZone. Le choix devra être fait en fonction de plusieurs critères, notamment de leur potentielle compatibilité avec SentinelC. La compagnie GlobalPlatform, créatrice du TPM, a aussi créé des spécifications pour aider le développement d'applications utilisant du Trusted Computing. Notamment, leur API client décrit comment les programmes du Monde Normal devraient pouvoir communiquer avec le Monde Sécurisé et leur API interne liste des fonctions qu'un système d'exploitation sécurisé devrait mettre à la disposition des Applications de Confiance. Cela permet aux Applications de Confiance qui respectent cet API de fonctionner dans plusieurs TEE sans être modifiées.

Système d'exploitation ANDIX

ANDIX [90] est un système d'exploitation sécurisé conçu pour augmenter la sécurité des systèmes de contrôle industriels (ICS) grâce aux propriétés d'isolation de ARM TrustZone. Il permet de rouler des Applications de Confiance dans le Monde Sécurisé en laissant un système d'exploitation standard s'exécuter dans le Monde Normal. Le système ANDIX tente de minimiser la taille de son TCB, notamment en laissant le système d'exploitation non sécurisé se charger de certaines tâches complexes comme la gestion du système de fichiers et de la réseautique. Une interface d'appels à distance permet au système d'exploitation sécurisé

de demander à celui non sécurisé de faire des opérations à sa demande. Il peut aussi attribuer des périphériques spécifiquement au Monde Sécurisé, ce qui empêche le Monde Normal d’y accéder. Cela est pratique pour l’implémentation d’entrées et sorties sécurisées. ANDIX suit les spécifications de GlobalPlatform pour l’interface de communication entre les Applications de Confiance et le Monde Normal pour offrir une plus grande compatibilité avec d’autres solutions.

Le système d’exploitation ANDIX a été développé principalement à des fins de recherche. Conséquemment, il a été conçu pour rouler la plateforme de prototypage IMX53QSB ou sur une instance de QEMU [91] modifiée pour supporter les mécanismes de sécurité de TrustZone. Toutes ses composantes sont disponibles à source ouverte, incluant le noyau, le support des Applications de Confiance et les bibliothèques pour l’intégration au Monde Normal.

Genode

Genode [92] est un projet visant à faciliter la création de systèmes sécurisés. Il met l’accent sur la modularité de ses composantes ainsi que le principe de la confiance minimale pour minimiser la surface d’attaque. En effet, Genode met à la disposition des développeurs une multitude de blocs, comme des pilotes de périphérique, des outils de virtualisation et des applications, qu’ils peuvent eux-mêmes assembler selon leurs besoins. Les interactions entre les composantes sont très strictement contrôlées selon des règles définies. L’utilisation d’un micronoyau accentue encore plus l’isolation entre chaque entité du système, comme les gestionnaires de réseau, des processus ou de l’affichage. Une telle architecture permet de restreindre les capacités d’un attaquant si l’un des sous-systèmes venait à être compromis. Genode n’utilise pas les fonctionnalités de ARM TrustZone. Il peut rouler sur des processeurs x86, ARM ou même RISC-V. Ses facultés d’isolations viennent plutôt de son architecture récursive. Chaque composante est enfermée dans une sorte de bac à sable ayant des accès restreints. Une composante peut elle-même avoir d’autres sous-composantes, elles aussi isolées, dont elle peut contrôler les accès.

Sierra TEE

Sierra TEE [93] est un système d’exploitation sécurisé qui permet d’isoler un noyau sécurisé d’un système d’exploitation standard. Il respecte lui aussi les API de GlobalPlatform pour la communication entre les milieux. Sierra TEE peut utiliser les facultés de TrustZone sur les processeurs ARM ou de OmniShield sur les processeurs MIPS pour créer une isolation matérielle. Il peut aussi utiliser de l’émulation pour recréer les mêmes fonctionnalités sur un processeur sans extension de sécurité. Malheureusement, SierraTEE est un logiciel

propriétaire. Il est donc difficile d'en savoir plus sans faire une soumission.

Trusty

Trusty, ou le TEE Fidèle [94], est un système sécurisé à source ouverte créé par les développeurs d'Android. Il est composé d'un noyau sécurisé, un pilote pour permettre à un noyau Linux régulier de communiquer avec le Monde Sécurisé et d'une librairie pour faciliter les interactions avec les Applications de Confiance. Celles-ci s'exécutent toutes dans leur propre bac à sable avec une priorité égale. Le potentiel d'utilisation d'un TEE sur les appareils Android est vaste, allant de la saisie de Numéro d'Identification Personnel (NIP) ou d'empreintes digitales sécurisée à la lecture de contenu protégé par le DRM en passant par la protection de l'appareil contre la réinitialisation forcée. Trusty n'utilise pas l'interface développée par GlobalPlatform et met plutôt à la disposition des développeurs sa propre interface de programmation. Il est possible d'utiliser Trusty sur un processeur Intel en substituant l'isolation fournie par TrustZone par de la virtualisation matérielle [95]. Le Monde Sécurisé et le Monde Normal fonctionnent alors chacun dans une machine virtuelle tandis que l'hyperviseur répond au rôle de moniteur pour les interactions entre celles-ci.

OP-TEE

OP-TEE [96] est une implémentation à source ouverte d'un TEE. Le projet met l'accent sur une petite taille bien adaptée aux systèmes ayant moins de ressources, ainsi qu'une bonne portabilité, tout en fournissant une robuste isolation du Monde Sécurisé. Le coeur de la suite est composé d'un moniteur sécurisé, d'un noyau Linux modifié, d'une librairie pour permettre aux programmes non sécurisés d'interagir avec les Applications de Confiance et d'un service roulant dans le Monde Normal responsable de la communication avec le noyau sécurisé. De plus, l'équipe de OP-TEE met à la disposition des développeurs une suite de tests pour valider les fonctionnalités du TEE, ainsi que des exemples d'Application de Confiance servant de références. Ces détails démontrent de la maturité du projet, qui a été rendu public dès 2014. Cela a généré une forte communauté toujours en quête d'améliorer la solution, ainsi qu'une bonne documentation sur tous ses rouages. OP-TEE supporte officiellement une cinquantaine de plateformes de développement, incluant un bon nombre de plateformes hybrides possédant une partie FPGA, très versatiles pour la recherche et le développement. L'équipe cible aussi des plateformes plus accessibles, comme le Raspberry Pi 3, pour permettre à tous de s'initier aux TEE. Toutes les plateformes supportées ont un processeur ARM Cortex-A, car les capacités d'isolation d'OP-TEE reposent sur les caractéristiques de TrustZone.

Pearl-TEE

Pearl-TEE [97] est une amélioration des architectures courantes de TEE. Dans les implémentations populaires, il y a une sorte de confiance mutuelle implicite entre le noyau sécurisé et toutes les Applications de Confiance, car celles-ci ont théoriquement accès à toutes les zones de mémoire protégées. Cela restreint fortement l'installation d'Applications de Confiance provenant de tierces parties, car elles doivent être dignes de confiance. De plus, les implémentations ont très peu d'Applications de Confiance et celles-ci sont rigoureusement testées. Cette façon de faire va de pair avec la réduction de la taille du TCB, qui se traduit en général par une plus petite surface d'attaque. Or, de multiples applications pourraient bénéficier des fonctionnalités d'un TEE, comme le stockage sécurisé, même si elles ne proviennent pas de sources vérifiées. Pour que l'intégrité du système ne soit pas compromise avec l'installation d'une Application de Confiance vulnérable ou malveillante, la mémoire et l'exécution de chaque Application de Confiance doivent être isolées du reste. Pearl-TEE génère une clé de chiffrement spécifique à chaque Application de Confiance. Cette clé sert ensuite à chiffrer tout le stockage de l'Application, empêchant ainsi les autres d'y accéder. Cependant, les seules données qui peuvent être sécurisées de cette manière sont des données qui ne sont écrites qu'une fois, comme des clés ou autres données qui ne changent jamais. Cela est nécessaire pour éviter les attaques par jeu, mais restreint énormément la solution. La raison sous-jacente est que, comme pour la plupart des TEE, toutes les opérations relatives au système de fichiers sont déléguées au système d'exploitation du Monde Normal. Un processus malveillant pourrait donc enregistrer les données chiffrées à un moment donné et les écrire par-dessus les nouvelles données chiffrées plus tard, annulant ainsi toute modification faite. Pearl-TEE utilise une unité de gestion de mémoire pour isoler les plages d'adresses des Applications de Confiance entre elles. Les TEE populaires isolent déjà l'exécution des différentes Applications de Confiance. Cependant, Pearl-TEE va plus loin en allouant une certaine durée d'exécution à chaque Application pour les empêcher de bloquer le système de manière à créer un déni de service. Pearl-TEE est une solution très avant-gardiste permettant d'installer sans crainte des Applications de Confiance tierces. Elle s'adresse en particulier à Android et aux appareils mobiles, où de nombreuses applications sont installées dynamiquement par l'utilisateur. Leur prototype est une modification d'OP-TEE, qui lui ajoute environ 3% de lignes de code. De ce fait, la plupart de ses fonctionnalités restent inchangées, dont la communication par l'interface de GlobalPlatform.

Comparaison

Toutes les solutions de TEE explorées ont des caractéristiques qui les démarquent. En ce qui concerne ce projet de recherche, les plus importantes sont l'utilisation de TrustZone, puisque la plupart des routeurs n'utilisent pas l'architecture x86, la publication du code source et l'utilisation de l'interface GlobalPlatform pour les communications, afin que la solution développée soit la plus réutilisable possible. Ces caractéristiques sont résumées dans le tableau 4.1.

TABLEAU 4.1 Caractéristiques des TEE

Système	Utilise TrustZone	Source ouverte	API GlobalPlatform
ANDIX OS	Oui	Oui	Oui
Genode	Non	Oui	Oui
SierraTEE	Oui	Non	Oui
Trusty	Oui	Oui	Non
OP-TEE	Oui	Oui	Oui
Pearl TEE	Oui	Non	Oui

Les seuls projets qui répondent à tous les principaux critères sont ANDIX OS et OP-TEE. Notre choix s'est porté sur OP-TEE, car ANDIX OS est plus adressé à la recherche académique qu'à la mise en marché d'un produit. De plus, il ne supporte officiellement qu'une seule plateforme, comparé à une cinquantaine pour OP-TEE. Il y a donc beaucoup plus de chances qu'une installation de OP-TEE soit compatible avec SentinelC. Finalement, la communauté active et la documentation riche de OP-TEE en font un excellent choix pour l'élaboration d'un prototype.

4.2.3 Plateforme de développement

Le choix de la plateforme sur laquelle le prototype sera implémenté repose sur plusieurs facteurs. En premier lieu, certaines contraintes sont imposées par les technologies choisies, comme ARM TrustZone et OP-TEE pour les besoins de Trusted Computing. Le processeur de la plateforme choisie devra donc faire partie de la famille Cortex-A [98]. De plus, une forme de racine de confiance devra être disponible sur la plateforme. Certaines plateformes possèdent des TPM, comme la plateforme ODYSSEY X86J4105864 de seedstudio [99], mais ces options ont été rejetées d'emblée à cause de leur prix élevé. Le TPM n'est pas encore assez démocratisé

pour être abordable pour un vendeur de routeurs commerciaux. De plus, la solution de TEE choisie n'a pas besoin de TPM pour établir une chaîne de confiance sécurisée. Sur le plan logiciel, le choix de Yocto pour assurer le développement oblige à sélectionner une plateforme qui possède des couches de bas niveau, comme le BSP, déjà implémentées et disponibles. En deuxième lieu, plusieurs caractéristiques sont dictées par le contexte de la recherche, c'est-à-dire l'utilité de l'appareil en tant que routeur sur lequel devra fonctionner SentinelC OS. De ce côté, les requis minimaux sont 1GB de RAM et 2GB de mémoire flash (généralement sous la forme d'une carte SD). De plus, il serait souhaitable que la plateforme ait au moins 2 ports Ethernet, bien qu'il serait acceptable que l'un des 2 soit remplacé par un port USB et un adaptateur. Finalement, certains critères sont désirables sans toutefois être essentiels. Notamment, une communauté active de développeurs peut aider à la résolution rapide des problèmes potentiels. Aussi, un bas prix est désirable pour faciliter la popularisation de la solution. Les développeurs de SentinelC ont porté leur projet sur seulement quelques plateformes. De celles-ci, seuls le Raspberry Pi 3 et le Marvell ESPRESSObin possèdent un processeur ARM. La liste des plateformes de développement officiellement supportées par OP-TEE est disponible dans sa documentation [100]. Or, la documentation est aussi très explicite sur le fait que le port de OP-TEE sur le Raspberry Pi 3 n'est pas réellement sécuritaire [101]. La raison est que son micrologiciel de démarrage est propriétaire, ce qui le rend impossible à ajouter dans la chaîne de confiance du démarrage sécurisé. La plateforme ESPRESSObin répond toutefois à tous les critères voulus. Elle possède un processeur ARM Cortex A53, 3 ports Ethernet, 2GB de RAM et un emplacement de carte SD pour la mémoire flash, le tout pour un prix raisonnable. La communauté est active et plusieurs couches Yocto sont déjà disponibles, certaines créées par les développeurs de SentinelC.

4.3 Développement sur le ESPRESSObin

L'équipe de SentinelC avait déjà rassemblé les ressources nécessaires pour la compilation d'un système minimal sur le ESPRESSObin. Eux-mêmes se sont basés sur le travail fait par l'équipe de TeamRedLabs sur les couches de base de la plateforme. Ce système comprend un micrologiciel de démarrage, un noyau Linux et une installation de Mender. Ces ressources prennent la forme de couches et d'options de compilation sur Yocto. Cela en fait donc une excellente base sur laquelle construire un système pour le ESPRESSObin.

4.3.1 Yocto

Yocto est un projet ouvert visant la réutilisation de code en encapsulant les différentes composantes d'un système sous la forme de couches. Ces couches doivent être le plus indépendantes

possible les unes des autres afin de permettre une bonne modularité. Par exemple, si un système utilise un certain programme, il est naturel que l'installation de ce programme soit totalement indépendante de l'installation du micrologiciel de démarrage utilisé par le système. Ces composantes doivent donc se retrouver dans des couches différentes. Ainsi, il sera plus facile de porter l'utilisation du programme sur une autre plateforme, car il suffira d'interchanger la couche qui gère l'intégration matérielle pour celle de la nouvelle plateforme tout en gardant les couches supérieures. La couche qui gère l'intégration au matériel s'appelle la couche Board Support Package (BSP). Yocto permet donc de porter un logiciel sur de nombreuses plateformes beaucoup plus facilement. Chaque couche est composée de plusieurs recettes, qui servent chacune à compiler et installer une composante de la couche.

Une autre particularité de Yocto est que certaines recettes sont utilisées pour compiler les outils de compilation du système. Une des plus grandes difficultés lors du développement d'un projet embarqué est la réplication d'une compilation réussie d'une machine à l'autre. Chaque développeur possède un environnement de développement différent, ce qui peut engendrer des erreurs de compilation très difficiles à résoudre. Yocto permet la réplication des conditions de compilation en intégrant des recettes qui servent à compiler des versions précises des compilateurs qui sont ensuite utilisés pour compiler les programmes du système. Par exemple, si un programme doit être compilé avec GCC, des versions différentes de GCC présentes sur les machines de différents développeurs peuvent donner des résultats différents. Une cible Yocto aura donc une recette pour d'abord compiler une version précise de GCC qui sera ensuite utilisée pour compiler le programme. Cela permet une réplique presque parfaite des conditions de compilation et réduit drastiquement les risques d'erreurs.

Les couches comprennent plusieurs recettes et il peut arriver qu'on ne veuille pas toutes les exécuter. Aussi, plusieurs couches peuvent avoir des versions différentes d'une même recette. Lorsque plusieurs recettes construisent la même cible, la recette la plus récente est utilisée. Les recettes peuvent être personnalisées via le fichier *local.conf*. Ce fichier permet de changer la valeur de variables utilisées dans les recettes. Cela permet la personnalisation des couches pour un système en particulier. Pour partager un système complet utilisant Yocto, il faut donc partager à la fois les couches utilisées et le fichier *local.conf*. Lors de la compilation, Yocto construit toutes les recettes dont les cibles sont spécifiées dans le fichier *local.conf*. Pour modifier une recette, il est recommandé d'utiliser des fichiers *.bbappend*, qui sont concaténés au fichier de recette.

Afin de rendre la compilation encore plus reproductible et prévisible, nous avons utilisé un conteneur *docker* spécialement créé pour construire des logiciels avec Yocto. Les couches utilisées sont ajoutées avec la commande *bitbake-layers add-layer*. La compilation de toutes

les recettes pour la cible *core-image-minimal* est lancée avec la commande *bitbake core-image-minimal*. Après un moment, toutes les composantes sont compilées et les images sont disponibles dans un dossier. L'image du micrologiciel de démarrage peut être mise sur la plateforme à l'aide d'une clé USB. Les images contenant le système d'exploitation, les partitions de Mender et le système de fichiers sont regroupées en un seul fichier qui peut être mis directement sur la carte SD du ESPRESSObin. Malheureusement, les ressources fournies par l'équipe de SentinelC sont faites pour la version 5 de la plateforme ESPRESSObin, alors que la version actuelle est la 7. Elles ont dû être modifiées avant de pouvoir être utilisées. Entre autres, le système de fichiers résultant était plus grand que la taille permise normalement par Mender. Il a fallu augmenter la taille maximale en modifiant une variable dans le fichier *local.conf*.

```
MENDER_BOOT_PART_SIZE_MB = "32"
```

De plus, il faut ajuster le paramètre *DDR_TOPOLOGY* pour profiter des 2 GB de mémoire vive de la plateforme. La compilation se termine alors avec succès. Le micrologiciel de démarrage U-Boot est mis sur une clé USB pour remplacer celui qui est préalablement en place. Cependant, l'image générée n'est pas signée, ce qui brise la chaîne de confiance du processus de démarrage. Le micrologiciel en place refuse donc d'installer une image qui n'est pas signée. Ce problème est réglé en appelant la cible *arm-trusted-firmware* de Yocto, qui s'occupe de signer les images de démarrage avec une clé privée de développement. Il est alors possible d'installer le nouveau micrologiciel. L'appareil ESPRESSObin n'arrive toutefois pas à démarrer, car l'adresse physique où l'image du noyau Linux est mise en mémoire par le micrologiciel de démarrage appartient à une zone réservée à OP-TEE. Cela résulte en une violation des droits d'accès et une interruption du démarrage. C'est le micrologiciel de démarrage qui lit l'image du noyau de la carte SD et la place en mémoire physique. Dans le cas de U-Boot, les adresses sont déterminées par des variables d'environnement qui sont présentes dans son image, mais qui peuvent aussi être changées dynamiquement lors du démarrage.

4.3.2 Variables d'environnement de U-Boot

Le micrologiciel U-Boot emmagasine ses variables d'environnement sous la forme de chaînes de caractères ASCII séparées par des caractères nuls. Lors du démarrage, il tente en premier lieu de lire les variables fournies par l'utilisateur depuis le périphérique de mémoire externe, soit la carte SD. Si la lecture échoue, U-Boot utilise des variables d'environnement par défaut, situées dans sa propre image et paramétrées lors de sa compilation [102]. Les variables d'environnement qui régissent les adresses en mémoire des images du noyau et de l'arbre

des périphériques (Device Tree) sont *kernel_addr_r* et *fdt_addr_r*, respectivement. Il est possible de modifier ces variables au démarrage en accédant au menu de démarrage de U-Boot. La commande *printenv* liste toutes les variables d’environnement. La variable *bootcmd* contient la commande lancée lors du démarrage. Il est possible d’en modifier seulement certaines parties en lançant chaque instruction manuellement. Cependant, il est préférable de modifier les variables dans le code pour ne pas avoir à interrompre le processus de démarrage à chaque lancement. Les variables d’adresse sont déjà modifiées par un correctif du code de U-Boot fait par l’équipe de TeamRedLabs. Il suffit donc de modifier leur correctif en y mettant des adresses qui ne sont pas dans l’espace réservé à OP-TEE pour permettre à U-Boot de démarrer le système sans erreur. Après l’application du correctif, les adresses dans les variables d’environnement par défaut dans l’image U-Boot sont maintenant valides. Il faut donc mettre à jour le micrologiciel de démarrage sur le ESPRESSObin pour que les changements prennent effet. Puisque U-Boot n’arrive pas à lire les variables d’environnement sur la carte SD, ce sont celles par défaut qui sont utilisées.

4.3.3 Récupération par UART

La modification du micrologiciel de démarrage à multiples reprises pour ajuster certaines options ne vient pas sans risque. Étant donné que le remplacement du micrologiciel se fait à partir du menu de démarrage fourni par celui-ci, si un micrologiciel fautif est installé, il est alors impossible de le remplacer. Toutefois, les concepteurs du ESPRESSObin ont prévu cette éventualité et ont intégré une méthode de remplacement du micrologiciel de démarrage en cas d’urgence. Cette technique, appelée récupération par UART, utilise des particularités du circuit de la plateforme. Pour l’utiliser, il faut d’abord changer la position de 3 connecteurs physiques. Il est ensuite possible d’utiliser un logiciel spécialisé pour remplacer le micrologiciel de démarrage sans passer par le menu de démarrage [103]. Une fois le changement des adresses appliqué aux variables d’environnement par défaut de U-Boot, le système arrive à démarrer et l’intégration des différentes composantes peut commencer.

4.4 Intégration de OP-TEE

Le projet OP-TEE est composé de plusieurs parties qui jouent chacune un rôle dans le système final. Premièrement, le système d’exploitation OP-TEE est le système d’exploitation sécurisé qui roule dans le Monde Sécurisé de TrustZone. Il s’occupe de gérer le stockage sécurisé et d’exécuter les Applications de Confiance. Deuxièmement, le client OP-TEE est une suite d’outils et de bibliothèques permettant au système d’exploitation non sécurisé d’interagir avec les Applications de Confiance. Troisièmement, les exemples sont des Applications de

Confiances simples créées afin de démontrer les capacités de base de OP-TEE. Elles peuvent servir à confirmer que ces capacités fonctionnent et comme inspiration lors de la création de notre propre Application de Confiance. Finalement, la suite de test *xtest* permet de vérifier si une installation de OP-TEE fonctionne bien grâce à plusieurs centaines de tests sur ses différentes fonctionnalités. La suite doit elle aussi être installée dans le système d'exploitation non sécurisé. Le système d'exploitation OP-TEE requiert plusieurs morceaux de code externes pour être installé sur la plateforme ESPRESSObin. Le répertoire *mv-ddr-marvell* contient le code nécessaire à la gestion des périphériques de la plateforme, le répertoire *atf-marvell* contient une version du ARM Trusted Firmware compatible avec le ESPRESSObin et le répertoire *A3700-utils-marvell* contient des utilitaires pour signer les images générées. Ces répertoires sont gérés par le fabricant de la plateforme, car le code qui s'y trouve est fortement dépendant de son côté matériel. Une installation minimale de OP-TEE comprend donc le micrologiciel de démarrage U-Boot, le système d'exploitation OP-TEE et les répertoires mentionnés ci-haut. Toutefois, une installation minimale n'est pas pratique pour un projet de développement, car la suite de test permet de s'assurer que tout fonctionne réellement et le client OP-TEE est nécessaire pour lancer des Applications de confiance depuis le système d'exploitation non sécurisé. La méthodologie employée pour cette phase du développement est de diviser pour mieux régner. Chaque composante du système final est analysée et construite à part pour s'assurer d'une bonne compréhension de son fonctionnement avant de l'intégrer au projet final. Il est donc pertinent de construire une installation minimale de OP-TEE avant de la rendre complète. Les différents répertoires de code sont compilés en faisant bien attention de prendre la même version pour tous. Cela est facilité par la fonctionnalité d'étiquetage de version (tags) du protocole Git. Les commandes pour la compilation sont prises presque telles quelles depuis la documentation de chaque répertoire. Les seuls changements sont les chemins vers les fichiers.

L'installation minimale de OP-TEE arrive à démarrer sans erreur. Or, cela ne veut pas dire que les fonctionnalités de OP-TEE sont toutes fonctionnelles, car le client et la suite de test ne sont pas installés pour le confirmer. L'intégration de ces composantes est à la fois essentielle et beaucoup plus complexe, car elle doit s'immiscer dans la construction du système de fichiers. En effet, une fois compilés, les exécutables résultants doivent se retrouver au bon endroit pour pouvoir être utilisés. La documentation officielle de OP-TEE réfère à l'utilisation de Makefiles fournis par les développeurs. Ces Makefiles sont très faciles à utiliser, car ils permettent la construction du système en entier avec à peine quelques commandes. Toutefois, ils sont d'autant plus difficiles à comprendre et à modifier pour les adapter à de nouvelles plateformes. Même si le processeur Marvell ARMADA 3700 utilisé est officiellement supporté par OP-TEE, il n'existe pas de Makefile pour la plateforme ESPRESSObin. De plus,

les Makefiles fournis utilisent Buildroot pour construire leur système, qui est un concurrent à Yocto. Les commandes utilisées ne sont donc pas compatibles avec la méthode de compilation choisie pour ce projet. Afin de porter la compilation de toutes les composantes de OP-TEE sur Yocto, chaque élément doit être transféré un à un vers le projet Yocto en s'assurant que tout fonctionne à chaque étape.

Le client OP-TEE n'est composé que d'une librairie et d'un exécutable, le *tee-suppllicant*. À priori, il est donc possible de les installer manuellement sans les intégrer dans le processus de compilation de Yocto. Cela constitue un bon premier test de l'installation du système d'exploitation OP-TEE. Le transfert sur la plateforme peut se faire avec une clé USB ou sur le réseau. Pour un transfert sur le réseau, l'option la plus simple est d'utiliser l'utilitaire *netcat* se trouvant par défaut sur l'installation de BusyBox. Malheureusement, la version de *netcat* installée n'a pas l'option d'écouter sur un port pour recevoir une connexion. Cette option doit être ajoutée manuellement en modifiant le fichier *defconfig* de la recette de BusyBox pour activer l'option *CONFIG_NC_SERVER*.

Malheureusement, le lancement du client résulte en une erreur qui mentionne qu'il n'y a pas de dispositif OP-TEE sur le système. En effet, le client utilise les fichiers spéciaux */dev/tee0* et */dev/teepriv0* pour communiquer avec le noyau Linux non sécurisé afin qu'il transmette ses requêtes au noyau OP-TEE. L'absence de ces fichiers sur le système indique que le noyau non sécurisé n'est pas conscient de la présence d'un noyau sécurisé sur la machine. Cela n'est pas surprenant puisque les recettes de Yocto utilisent un noyau Linux ordinaire. L'organisation à but non lucratif Linaro, créatrice de OP-TEE, maintient un noyau Linux spécialement adapté pour la collaboration avec le noyau sécurisé OP-TEE. Il est donc nécessaire de remplacer le noyau ordinaire par celui-ci dans les recettes Yocto.

4.4.1 Modification du code source

Initialement, tous les morceaux de code utilisés dans la compilation avec Yocto sont pris directement de répertoires de code sur l'internet. Or, il peut être utile de modifier le code source d'une composante afin de mieux la déboguer ou d'ajouter une fonctionnalité. Pour ce faire, il faut indiquer à Yocto de prendre une version locale du code plutôt que de la prendre en ligne. Il faut premièrement s'assurer que le dossier où se trouve notre code est bien accessible par les programmes à l'intérieur du conteneur docker utilisé pour la compilation. Ensuite, il faut modifier les recettes visées pour qu'elles utilisent le code local. La variable *SRC_URI* détermine la source du code utilisé par une recette. Plusieurs protocoles sont supportés, dont le protocole *file ://* pour les fichiers locaux. Cette modification change aussi le chemin vers lequel les sources sont copiées. Il faut donc aussi changer la variable *S*, qui indique le chemin

à utiliser pour les commandes de compilation. Pour chaque répertoire de code, il faut faire attention de bien utiliser une version qui est compatible avec les autres composantes. Yocto permet de préciser une étiquette de version spécifique directement dans la variable *SRC_URI*.

Une fois le nouveau noyau Linux mis en place, l'appareil ne parvient plus à démarrer, car il ne reconnaît plus la carte SD sous le bon nom. Les messages de démarrage indiquent que le noyau attend de voir la partition */dev/mmcblk0p2*, mais la carte SD est reconnue sous le nom *mmcblk1*.

4.4.2 Arborescence des périphériques

L'attribution des noms aux périphériques est déterminée par un fichier d'arborescence des périphériques. Il indique les protocoles et les options qui doivent être utilisées pour interagir avec chaque composante de la plateforme, incluant les périphériques de mémoire. La plateforme ESPRESSObin possède 2 périphériques de mémoire externe. Le noyau de Linaro utilise une arborescence différente qui inverse l'index de ces 2 périphériques, c'est pourquoi le noyau attend le mauvais dispositif. Il suffit donc de créer une correction de code pour inverser de nouveau l'index des périphériques de mémoire externe dans l'arborescence du noyau de Linaro. Il faut aussi ajouter dans l'arborescence le dispositif *optee* en spécifiant sa méthode d'accès et la plage d'adresse à laquelle il se trouve. La correction de code en question est montrée à l'annexe A. Il faut ensuite s'assurer que cette correction est appliquée en l'ajoutant à la variable *SRC_URI*. Cette modification de l'arborescence des périphériques permet au noyau de Linaro de démarrer avec succès, mais les dispositifs */dev/tee0* et */dev/teepriv0* sont toujours absents. La manière dont le noyau OP-TEE se rend disponible auprès du noyau non sécurisé est en se faisant passer pour un périphérique. Il doit donc lui aussi être ajouté au fichier d'arborescence des périphériques. Le micrologiciel de démarrage doit lui aussi être conscient de la présence de OP-TEE, car il est impératif que le noyau sécurisé soit lancé avant le noyau non sécurisé dans le processus de démarrage. Il faut donc indiquer au ARM Trusted Firmware la présence d'un Secure Payload Dispatcher (SPD) lors de la compilation. Il faut aussi lui indiquer le chemin vers l'image du SPD, dans ce cas-ci le fichier *tee-pager_v2.bin*, afin qu'il l'utilise comme stade BL32 lors du démarrage. Le fichier est copié dans le dossier de sortie des images, dont le nom est dans la variable *DEPLOY_DIR_IMAGE*. Il suffit donc d'ajouter les 2 lignes suivantes aux options de compilation de *arm-trusted-firmware*.

```
SPD=opteed
```

```
BL32="${DEPLOY_DIR_IMAGE}/optee/tee-pager_v2.bin"
```

Finalement, pour que le noyau non sécurisé soit conscient de l'existence du noyau OP-TEE,

il faut lui ajouter les options `CONFIG_TEE` et `CONFIG_OPTEE`. Après toutes ces modifications, les dispositifs de OP-TEE sont finalement visibles dans le dossier `/dev` et le client `tee-supplciant` peut être lancé sans erreur. Cependant, seule la suite de test peut réellement confirmer que tout fonctionne correctement. Puisque l'intégration complète de OP-TEE est presque finalisée, il est plus judicieux à ce point de transférer toute la compilation vers Yocto. La compilation manuelle de plusieurs composantes permet de mieux comprendre et de faire des modifications et des tests plus facilement, mais l'objectif final est d'avoir un système qui se construit automatiquement en quelques commandes grâce à Yocto. Cela simplifie aussi l'ajout de la suite de test, car elle contient beaucoup de fichiers, ce qui la rend fastidieuse à transférer manuellement vers l'appareil.

4.4.3 Intégration de meta-arm

Il existe déjà des couches Yocto qui contiennent des recettes pour construire OP-TEE et ses composantes. La plus notoire est dans le répertoire *meta-arm*, qui regroupe des couches d'usage général pour les plateformes ARM. Ce répertoire a été créé par Linaro et s'appelait auparavant *meta-linaro*. Ces recettes ne sont pas du tout dirigées vers la plateforme ESPRESSObin, mais la modularité inhérente des couches Yocto facilite le transfert d'une plateforme à une autre. Le répertoire *meta-arm* est énorme et contient des dizaines de recettes. Les couches qui contiennent les recettes relatives à OP-TEE sont *meta-arm* et *meta-arm-toolchain*. Elles peuvent être ajoutées au projet avec le programme *bitbake-layers*. Finalement, il faut ajouter les cibles voulues dans le fichier *local.conf* pour qu'elles soient réellement construites.

```
IMAGE_INSTALL_append = " optee-os optee-client optee-test optee-examples"
```

Malheureusement, ces recettes ne sont toujours pas construites, car la plateforme ESPRESSObin ne fait pas partie de leur liste de plateformes supportées. Il faut donc ajouter manuellement la machine *espressobin-v7* à la variable `COMPATIBLE_MACHINE`. Similairement, la recette du système d'exploitation OP-TEE échoue lors de la compilation, car il n'existe pas de configuration du nom de *espressobin-v7* dans les dossiers du répertoire. La variable `OPTEEMACHINE` est utilisée pour déduire le chemin de la configuration et prend par défaut la valeur de la variable `MACHINE`, soit *espressobin-v7*. Il existe cependant des fichiers de configuration pour le processeur Marvell ARMADA 3700 dans le dossier *marvell-armada3700*. Il faut donc ajuster la variable `OPTEEMACHINE` en conséquence dans le fichier de configuration *local.conf*. Ces ajustements permettent la construction du système complet incluant toutes les composantes de OP-TEE. De plus, tous les tests conduits par le programme *xtest* réussissent, ce qui signifie que l'installation est réellement réussie et que toutes les fonction-

nalités sont utilisables. L'intégration de OP-TEE au système permet l'ajout d'une couche de sécurité matérielle au routeur. Cependant, à ce stade l'appareil n'a aucune fonctionnalité nécessaire à un routeur. Il faut ajouter le code de SentinelC au système pour profiter de toutes ses capacités et de la sécurité logicielle qui lui est propre.

4.5 Intégration de SentinelC

L'équipe de SentinelC a déjà mis en place des couches Yocto pour la construction du micrologiciel d'un routeur SentinelC. Cependant, les seules plateformes supportées pour le moment sont le Raspberry Pi et les appareils possédant un processeur Intel 64 bits. Heureusement, la modularité de Yocto facilite la réutilisation du code sur d'autres plateformes. L'intégration du système d'exploitation SentinelC sur une plateforme repose sur 3 éléments. Premièrement, le système utilise Mender pour gérer les mises à jour du micrologiciel. Cela change la disposition des partitions et le processus de création des images. Cette partie est déjà partiellement intégrée dans le prototype, car les premières couches Yocto sur lesquelles il est construit avaient pour objectif l'intégration de Mender sur le ESPRESSObin. Deuxièmement, les fonctionnalités du routeur SentinelC requièrent des options de configuration particulières du noyau Linux. Étant donné qu'il est essentiel d'utiliser le noyau Linux fourni par Linaro pour faire fonctionner OP-TEE, il faut transférer ces options de configuration d'un noyau à l'autre. Troisièmement, il faut ajouter les programmes et outils relatifs aux fonctionnalités du routeur SentinelC dans le système de fichiers. Cette partie est gérée par une couche Yocto qui télécharge et construit la partie *daemon* du projet SentinelC. Les couches nécessaires à la construction du système sont ajoutées avec la commande *bitbake-layers*. L'étape qui suit est de fusionner les fichiers *local.conf* de notre prototype et de SentinelC en choisissant bien les options à garder et celles à écarter. Par exemple, les couches Yocto de SentinelC utilisent un noyau Linux standard stable. Ce choix se reflète par l'option suivante.

```
PREFERRED_PROVIDER_virtual/kernel = "linux-stable"
```

Or, ce noyau n'est pas compatible avec OP-TEE. La recette qui utilise le noyau de Linaro est dans le fichier *linux-esspressobin_5.10.100.bb*. Il faut donc changer l'option du noyau préféré pour *linux-esspressobin* dans le fichier *local.conf* pour l'utiliser. De plus, le client SentinelC utilise des fonctionnalités du noyau Linux qui doivent être activées à la compilation, comme les *nftables* et le système de fichiers *overlayfs*. Ces options peuvent être ajoutées directement dans le fichier *defconfig* de la recette du noyau de Linaro. Une paire de clés RSA doit aussi être générée avec *openssl* en suivant les instructions de la documentation Mender afin de signer les images de mise à jour. Le système d'exploitation SentinelC utilise un système de fichiers

de type *overlayfs*. Ce système de fichiers rend tous les fichiers en lecture seule par défaut et les fichiers modifiés sont stockés dans une couche séparée. Cette configuration nécessite des paramètres de démarrage spécifiques, qu'il faut ajouter aux variables d'environnement de U-Boot.

```
rootrw=/dev/mmcblk1p4 init=/sbin/init-overlay
```

Le paramètre *rootrw* indique la partition principale à monter en lecture et écriture tandis que le paramètre *init* spécifie le programme qui doit être démarré pour lancer le système [104]. Ces options sont déjà définies dans une correction de code créée par l'équipe de SentinelC, mais elle est désactivée, car la syntaxe est conflictuelle avec celle d'une autre correction de code. Pour préserver la modularité des corrections de code, il est préférable d'en faire une nouvelle qui respecte la syntaxe. Cela peut être accompli en faisant les modifications nécessaires dans le répertoire de code et en les extrayant avec la commande *git format-patch*. À ce stade, toutes les composantes ont été intégrées au système de compilation Yocto pour la plateforme ESPRESSObin. L'appareil arrive à démarrer et à exécuter les opérations régulières d'un système SentinelC tout en possédant les capacités propres à OP-TEE. Il faut ensuite créer une Application de Confiance pour faire profiter le routeur SentinelC des avantages de OP-TEE en termes de sécurité.

4.6 Application de Confiance

Les Applications de Confiance (Trusted Applications) sont des programmes qui roulent de manière isolée dans le Monde Sécurisé. Concrètement, le processeur est dans le niveau d'exécution 3 lorsqu'il exécute les instructions du programme. L'Application a donc théoriquement accès à toutes les zones de mémoire de l'appareil. Chaque Application de Confiance possède 2 exécutables. Le premier est l'application à proprement parler et s'exécute dans le Monde Sécurisé. Or, tous les échanges de données entre le Monde Sécurisé et le Monde Normal utilisent des structures de données et une méthode d'appel, ou Application Binary Interface (ABI), bien précises. C'est pourquoi chaque application doit aussi comporter un exécutable situé dans le Monde Normal qui agit comme un client et qui expose les fonctionnalités de l'Application de Confiance d'une manière facile à utiliser. Les fonctionnalités propres au TEE, comme l'accès au stockage sécurisé ou la génération de données aléatoires, sont disponibles à travers les fonctions d'un API. Les exécutables du Monde Sécurisé et du Monde Normal ont chacun leur API, soit l'API TEE et l'API TEEC respectivement. Pour communiquer avec l'Application, le client dans le Monde Normal doit d'abord initialiser un contexte qui le lie au TEE. Ensuite, il doit ouvrir une session avec une Application de Confiance donnée.

Les Applications de Confiance sont chacune identifiées par un Universally Unique Identifier (UUID) sans signification particulière. Même l'exécutable du Monde Sécurisé est stocké dans le système de fichiers du système d'exploitation non sécurisé, dans un fichier portant le nom du UUID avec l'extension *.ta*. Toutefois, cet exécutable est chiffré avec une clé contenue dans le TEE. Il est donc impossible de la modifier sans avoir accès à cette clé. Lorsque le client ouvre une session, l'Application est déchiffrée et lancée. Puis, le client peut faire autant d'appels à des fonctions de l'Application qu'il souhaite. Finalement, il doit conclure en fermant la session et en détruisant le contexte. L'Application de Confiance n'a qu'une seule fonction faisant office de point d'entrée. Si elle désire offrir plusieurs fonctionnalités, elle peut prendre un entier en argument et faire correspondre chaque valeur à une fonction spécifique. Le client peut ensuite passer le numéro de la fonction comme premier argument et tous les autres ensuite. Il est à noter qu'une Application ne peut avoir que 4 arguments au maximum et qu'ils doivent tous être soit un entier sur 64 bits ou un tampon de taille fixe. Il faut donc tout d'abord déterminer les fonctionnalités que l'Application de Confiance doit exposer pour ensuite faire une liste des fonctions à implémenter.

Afin de sécuriser le jeton d'authentification utilisé par le client SentinelC, il ne suffit pas de le stocker dans le TEE. En effet, pour pouvoir prouver son identité au serveur, le client doit à un moment donné sortir le jeton du TEE pour l'envoyer dans ses requêtes HTTP. À ce moment, un adversaire pouvant exécuter des commandes arbitraires en tant qu'utilisateur racine sur l'appareil est en mesure de lire le jeton, par exemple en extrayant la mémoire du client SentinelC. La seule manière de véritablement protéger le jeton est de créer un protocole spécialisé entre l'Application de Confiance, le client et le serveur afin que le routeur puisse prouver son identité sans que le jeton ne sorte du TEE. Bien que des primitives cryptographiques comme le Hash-based Message Authentication Codes (HMAC) ou des protocoles comme Kerberos, ayant des propriétés similaires à celles voulues, laissent présager qu'une solution à ce problème existe, il n'y a pas de protocole qui réponde exactement à ce scénario. De plus, il est connu que d'inventer son propre protocole cryptographique comporte de grands risques, car des failles peuvent facilement s'y glisser. Dès lors, puisque ce projet de recherche vise avant tout l'élaboration d'un prototype de routeur sécurisé à l'aide du Trusted Computing, nous avons décidé de nous en tenir à une preuve de concept simple. L'Application de Confiance implémentée augmente légèrement la sécurité du jeton, car il est impossible de le lire directement dans le système de fichiers de l'appareil. Or, son objectif est aussi de servir de base pour de futures recherches qui pourront se concentrer sur l'élaboration d'un protocole sécuritaire sans devoir se soucier de l'intégration du Trusted Computing et du système d'exploitation SentinelC sur la plateforme ESPRESSObin. La section 4.8 propose tout de même un protocole d'authentification, incluant l'attestation de l'état de l'appareil auprès du serveur, qui ne

permet pas à un adversaire d'exfiltrer le jeton même s'il possède les privilèges racines sur le système d'exploitation non sécurisé.

Une Application de Confiance minimaliste faisant office de preuve de concept a pour seul objectif d'enregistrer le jeton d'authentification dans le stockage sécurisé fourni par le TEE afin qu'il ne soit plus présent dans le système de fichiers. Elle n'a donc besoin d'exposer que 2 fonctions.

- `register_token` : enregistre le jeton en mémoire sécurisée
- `get_token` : lit et retourne le jeton de la mémoire sécurisée

Ces fonctions suffisent à intégrer l'utilisation du Trusted Computing dans la gestion de l'authentification du client SentinelC auprès du serveur. Concrètement, la sécurité du jeton n'est pas particulièrement améliorée lorsque l'appareil est en fonction, car n'importe quel utilisateur peut appeler la fonction `get_token` pour l'obtenir. Cependant, le jeton ne sera pas lisible dans le système de fichier quand l'appareil est éteint. De plus, il est préférable pour une preuve de concept de ne pas imposer une direction pour les développements futurs. Cette implémentation ne requiert qu'un léger ajustement du code du client SentinelC et aucune modification du code du serveur. Cela laisse une plus grande amplitude lors de la création d'un système véritablement sécuritaire dédié à être mis sur le marché.

4.6.1 Application cliente

Le jeton d'authentification est composé de 40 caractères alphanumériques aléatoires. Il ne peut donc pas être transmis sur 64 bits et doit être transmis dans un tampon mémoire. L'API TEEC prévoit 2 types de tampons pouvant être transférés du Monde Normal au Monde Sécurisé. Une structure de type *TEEC_RegisteredMemoryReference* contient un pointeur vers un tampon qui est partagé par les 2 mondes. Un tampon partagé peut être créé en appelant la fonction *TEEC_RegisterSharedMemory* sur un tampon existant ou en allouant un tampon partagé directement avec la fonction *TEEC_AllocateSharedMemory*. Cependant, il existe une autre méthode pour partager des données dans les cas simples. La structure de données *TEEC_TempMemoryReference* peut aussi être passée à un appel de fonction d'une Application de Confiance, mais elle n'a pas à pointer vers un tampon qui est partagé. En effet, la fonction se charge de partager automatiquement le tampon vers lequel l'argument pointe. Il faut bien sûr informer l'Application de Confiance que la région n'est que temporairement partagée en spécifiant le type de paramètre *TEEC_MEMREF_TEMP_<type>*. Le type est *INPUT* pour envoyer des données vers le Monde Sécurité et *OUTPUT* pour en recevoir. Le rôle de l'application cliente est d'exposer les fonctionnalités de l'Application de Confiance sur la ligne de commande. Elle doit donc lire les arguments et déterminer l'opération à effectuer.

Ensuite, elle initialise le contexte de connexion au TEE et ouvre une session en fournissant le UUID de l'Application de Confiance. Le tampon contenant le jeton est initialisé et rempli dans le cas de l'enregistrement du jeton. Puis, la fonction *TEEC_InvokeCommand* est utilisée pour appeler l'Application de Confiance avec les paramètres spécifiés. Il ne faut pas oublier que le premier argument est l'index de la fonction à appeler. Le client doit finalement gérer les erreurs s'il y en a et retourner le jeton lu à l'utilisateur si applicable.

4.6.2 Application sécurisée

L'Application de Confiance ne peut avoir qu'un seul point d'entrée. Cette fonction doit lire le premier paramètre pour déterminer laquelle des fonctionnalités est réellement appelée. Dans le cas de l'enregistrement du jeton, un tampon est alloué avec la fonction *TEE_Malloc*. Il est très important de transférer le jeton du tampon partagé vers un tampon qui n'est pas partagé avant de faire la moindre opération dessus pour éviter des situations de compétition (*race conditions*). En effet, l'Application n'a aucune garantie que le client ne modifiera pas les données du tampon partagé après l'avoir transmis. L'Application peut ensuite enregistrer le jeton dans la mémoire sécurisée avec la fonction *TEE_CreatePersistentObject*. Cette fonction prend en paramètre les attributs de l'objet à créer. Pour une preuve de concept, l'objet est créé avec des attributs permettant sa lecture, son écriture et sa suppression. Toutefois, il est très important qu'une réelle application ne mette que l'attribut de lecture. De cette façon, le jeton ne pourra être écrit qu'une seule fois et tout programme voulant le modifier pour empêcher le client SentinelC de fonctionner correctement échouera. Lors de la récupération du jeton, l'application cliente doit fournir un tampon partagé. L'Application de Confiance doit confirmer l'existence du jeton dans la mémoire sécurisée à l'aide de la fonction *TEE_GetObjectInfo1*. S'il est présent, son contenu est lu avec la fonction *TEE_ReadObjectData* et il est retourné au Monde Normal par le tampon partagé. Lors de la compilation, il est essentiel de lier la trousse de développement d'Applications de Confiance (TA DevKit) pour bénéficier des fonctions de l'API GlobalPlatform. Encore une fois, le répertoire d'exemples d'Applications de Confiance de OP-TEE est un bon guide et démontre sa grande utilité pour faciliter l'accès au Trusted Computing pour les développeurs.

4.6.3 Intégration au client SentinelC

Une fois l'Application de Confiance mise en place sur le système, il faut modifier le code du client SentinelC pour qu'il utilise le jeton en mémoire sécurisée plutôt que de le lire dans un fichier. Dans l'implémentation actuelle, le jeton est lu une seule fois au démarrage du client. Puis, il est inséré dans les entêtes des requêtes au serveur qui nécessitent d'être authentifié. Il

faut donc changer la manière dont ces requêtes remplissent leurs entêtes. Il serait très simple de lire le jeton au démarrage du client SentinelC et de le garder en mémoire tout au long de son exécution. Cependant, cela ne serait pas représentatif d'un scénario réel, car dans le cas où un véritable protocole sécurisé est utilisé, l'Application de Confiance devra être appelée à chaque requête. En effet, cela est nécessaire pour que l'Application de Confiance atteste de l'état du système avant d'exécuter la requête. Il est donc préférable d'encapsuler la récupération du jeton dans une fonction qui doit être appelée avant d'envoyer toute requête devant être authentifiée. L'appel à l'Application de Confiance est trivial grâce au module *subprocess* de Python. L'utilisation de la mémoire sécurisée est ainsi intégrée au client SentinelC avec un minimum de changements à son code. Dans l'implémentation originale, le jeton est créé par le programme */sbin/init-overlay.sh*. Ce programme est exécuté en premier sur le système à cause du paramètre de démarrage *init* passé au noyau Linux par U-Boot. Il se charge de générer un jeton aléatoire et de le mettre dans le fichier */data/api_token* si celui-ci est inexistant. Le code responsable de cette tâche est affiché dans le Listing 4.1.

Listing 4.1 La création originale du jeton

```

1 if [[ ! -e $ROOT_RWMOUNT/chinook/api_token ]]; then
2     api_token="$(random_string 40)"
3     log "Generating api_token... $api_token"
4     init_config_file 077 $ROOT_RWMOUNT/chinook api_token "$api_token"
5 fi

```

Il faut donc remplacer ce morceau de logique par un appel à l'exécutable du Monde Normal de l'Application de Confiance. Le jeton ne doit être généré que s'il n'est pas déjà existant, ce qui peut être vérifié par un appel à la fonction *get_token*.

Listing 4.2 La création modifiée du jeton

```

1 if [[ ! $(sentinelc-agent get_token) ]]; then
2     api_token="$(random_string 40)"
3     log "Generating api_token... $api_token"
4     sentinelc-agent register_token "$api_token"
5 fi

```

Grâce à cette modification, le jeton est automatiquement généré et mis dans la mémoire sécurisée de l'appareil lors de son premier démarrage. Celui-ci est ensuite prêt à être enrôlé auprès d'un serveur SentinelC.

4.7 Création de recettes Yocto

Les dernières sections ont décrit comment OP-TEE et SentinelC ont été adaptés à la plateforme ESPRESSObin. Le prototype est fonctionnel, mais il n'est pas facilement reproductible, car il consiste en de nombreuses modifications aux recettes Yocto provenant des répertoires de *meta-arm*, TeamRedLabs et SentinelC. Nous avons donc décidé de suivre la philosophie de Yocto, qui veut que tout projet soit un ensemble de couches itératives et modulaires. Nous avons créé plusieurs répertoires de code pour organiser ces modifications et rendre la reproduction de notre prototype la plus facile possible. Le but de cette démarche est d'encourager de futures recherches pour améliorer le prototype jusqu'à une éventuelle mise en marché. Le répertoire *sentinelc-agent* contient tout le code pour compiler l'Application de Confiance de notre prototype. Le répertoire *meta-sentinelc* contient toutes les modifications aux recettes déjà existantes que nous avons utilisées. Il a fallu extraire ces changements et les regrouper dans de nouvelles recettes. Par exemple, au lieu de modifier la recette *linux-espressobin* pour qu'elle utilise le noyau Linux de Linaro, nous avons créé une recette *linux-optee* à part entière. Il suffit ensuite de régler la variable *PREFERRED_PROVIDER_virtual/kernel* à *linux-optee* dans le fichier *local.conf* pour que ce soit ce noyau qui soit utilisé dans le système. Puisque beaucoup de configuration a aussi lieu dans le fichier *local.conf*, il convient de le fournir aux développeurs. C'est pourquoi nous avons créé un répertoire de code principal qui sert à rapatrier tous les autres répertoires et qui contient aussi le fichier *local.conf* utilisé lors de la compilation. Les répertoires qui contiennent des modifications qui sont impossibles à convertir en recette Yocto ont dû être clonés pour que les modifications soient accessibles directement par le protocole git. Les répertoires utilisés sont organisés dans un fichier *manifest.xml* dans le format dicté par l'outil repo [105]. Cela permet à n'importe qui de télécharger tous les répertoires requis et de bien les nommer en une seule commande. La puissance des outils repo et Yocto permet de télécharger et de compiler toutes les images de notre prototype à partir de rien en quelques commandes seulement (et en plusieurs heures). Au moment de l'écriture de ces phrases, la plupart de ces répertoires sont disponibles gratuitement sur GitHub. Cela est nécessaire pour que les recettes Yocto puissent les télécharger automatiquement. L'objectif est de tous les rendre disponibles gratuitement quand l'équipe de SentinelC sera prête à rendre public son projet.

4.8 Protocole d'authentification à distance

Ce projet de recherche a pour objectif l'intégration du Trusted Computing dans l'architecture d'un routeur. La création d'un protocole d'authentification à distance entre le client et le

serveur SentinelC est donc hors de sa portée. Aucun protocole existant ne répond exactement à la situation visée, c'est-à-dire mettant en jeu 4 partis dont 2 peuvent être complètement compromis. De plus, créer un protocole sécurisé demande une expertise particulière, car les primitives de cryptographie peuvent être sujettes à des attaques inusitées si mal employées. C'est pourquoi l'Application de Confiance créée à la section 4.6 n'est qu'une preuve de concept. Malheureusement, l'intégration du Trusted Computing dans le système SentinelC n'augmente pas significativement la sécurité de la solution si le protocole d'authentification repose sur l'échange d'un jeton non chiffré. La seule manière de réellement sécuriser le jeton est de le protéger non seulement dans son stockage, mais aussi dans son utilisation. Ce défi est laissé à de futures recherches. Néanmoins, nous avons élaboré un squelette détaillant les étapes générales d'un protocole sécurisé. L'objectif d'une telle démarche est d'une part la démonstration que les principes du Trusted Computing peuvent réellement protéger le jeton contre des adversaires ayant un accès racine sur l'appareil. D'autre part, ce squelette peut servir de ligne directrice pour l'éventuelle implémentation d'un protocole concret.

4.8.1 Attestation

Dans l'optique d'obtenir la meilleure sécurité possible pour le routeur, même un protocole d'authentification à distance n'est pas suffisant pour protéger son intégrité contre des attaques sophistiquées. Le protocole empêche quiconque de lire le jeton, car celui-ci réside dans la sécurité du TEE. Cependant, l'Application de Confiance n'a aucune manière de savoir quel programme fait appel à ses fonctions. Un adversaire pourrait donc programmer son propre client utilisant l'API TEEC pour faire authentifier des requêtes malveillantes auprès de l'Application de Confiance sans que celle-ci puisse faire la différence avec un appel légitime. De plus, le serveur SentinelC a seulement la certitude de communiquer avec le bon appareil, mais n'a aucune garantie de l'intégrité de celui-ci. Cette faille n'est pas aussi critique que le vol du jeton, car pour l'exploiter, toutes les requêtes doivent passer par le véritable routeur sans pouvoir dupliquer le jeton sur une autre machine. Toutefois, un protocole optimal devrait avoir une manière d'attester de l'intégrité du client SentinelC ou de l'état du système d'exploitation non sécurisé en général. Une étape d'attestation de l'état du routeur est donc intégrée au protocole d'authentification proposé. Cependant, la méthode d'attestation idéale n'est pas discutée ici et est laissée à des travaux futurs. Plusieurs manières d'attester de l'état d'un appareil embarqué sont discutées à la section 3.7.

4.8.2 Protocoles de défi-réponse

Les protocoles de défi-réponse permettent de prouver l'identité d'un client par la résolution d'un défi donné par le serveur. Ce type de protocole repose sur un secret qui est partagé entre le client et le serveur. Dans sa forme la plus simple, le défi est d'envoyer un mot de passe et le client doit répondre avec le bon mot de passe. Cependant, dans ce scénario, quiconque arrive à écouter la conversation entre le client et le serveur peut voler le mot de passe et plus tard se faire passer pour le client sans que le serveur puisse le détecter. C'est le cas du protocole d'authentification basique du protocole HTTP [106]. Il est une des premières méthodes d'authentification à avoir été ajouté au protocole HTTP, mais puisque le trafic internet passe par plusieurs noeuds avant de rejoindre sa destination, tous ces noeuds peuvent lire le mot de passe. C'est pourquoi cette méthode est comme non sécuritaire sans l'utilisation d'une couche supplémentaire de chiffrement comme TLS.

Un protocole de défi-réponse est approprié pour l'authentification des routeurs SentinelC pour plusieurs raisons. Premièrement, l'enrôlement des appareils prévoit déjà une phase durant laquelle le client et le serveur s'échangent des informations sous la supervision d'un opérateur. C'est un moment idéal pour échanger une valeur secrète qui pourra servir de base au défi, ce qui facilite l'implantation du protocole dans le flot d'utilisation de la solution SentinelC. Deuxièmement, la création dynamique des défis par le serveur augmente significativement la protection contre certaines attaques comme les attaques par rejeu. En effet, dans l'éventualité où un adversaire intercepte une réponse, il ne pourra pas l'utiliser pour authentifier de futures requêtes, car le défi sera différent. Il existe plusieurs protocoles défi-réponse bien établis comme le protocole Fast Identity Online (FIDO) [107] et le protocole Challenge Handshake Authentication Protocol (CHAP) [108], utilisé dans l'authentification de certains réseaux sans-fil. Cependant, ces protocoles ne conviennent pas tout à fait à ce projet, notamment parce qu'ils requièrent une interaction de l'utilisateur. La description de cet exemple de protocole a aussi comme objectif de démontrer son intégration avec le client et le serveur SentinelC.

4.8.3 Nomenclature

Il est important de bien identifier les termes utilisés pour que toutes les étapes soient claires. Il y a 4 entités impliquées dans le protocole.

- Serveur : serveur SentinelC dans le nuage
- Client : client SentinelC roulant sur le routeur
- Application de Confiance : exécutable sécurisé roulant dans le TEE du routeur
- Application hôte : exécutable non sécurisé exposant les points d'entrée de l'Application

de Confiance

Un *nonce* est une valeur qui est destinée à n'être utilisée qu'une seule fois. Les termes *snonce* et *cnonce* désignent un *nonce* qui a été généré par le serveur ou le client, respectivement.

4.8.4 Échange de clé initial

L'échange de clé initial repose sur le principe de la confiance au premier usage, ou *Trust on First Use*. La raison est qu'un attaquant disposant de privilèges racines sur la machine pourra toujours lire et modifier les messages échangés de manière à corrompre l'échange de clé. Par exemple, si une clé symétrique est choisie, l'attaquant sera en mesure de la lire s'il est bien positionné lors de l'échange. D'un autre côté, si une clé asymétrique est utilisée, l'attaquant pourra se placer en position de MITM pour que tous les messages utilisent ses clés. Dans les 2 cas, il sera en mesure de compromettre l'intégrité et la confidentialité des données transmises par la suite. C'est pourquoi le *Trust on First Use* suppose que l'appareil n'est pas compromis au moment de l'échange initial.

Une manière simple de sécuriser l'échange de clés est de générer une paire de clés asymétriques maîtresse sur le serveur SentinelC avant même de distribuer les appareils. La clé publique est mise directement dans l'Application de Confiance sur le routeur et la clé privée est utilisée pour signer cryptographiquement la clé que le serveur envoie à l'appareil. De cette manière, l'Application de Confiance est en mesure de vérifier que la clé reçue provient bel et bien d'un serveur SentinelC.

Selon ce protocole, l'échange de clé se produit lors d'un appel au chemin */register* sur l'API du serveur. À ce moment, le serveur génère une paire de clés asymétriques et signe la clé publique avec la clé privée maîtresse. Cette étape pourrait être sautée si tous les routeurs utilisaient directement la clé publique maîtresse, contenue dans leur micrologiciel, pour chiffrer des données. Cependant, il est plus sécuritaire et plus modulaire d'avoir une paire de clés pour chaque routeur enregistré sur le serveur. De cette manière, le serveur peut facilement révoquer ou invalider une paire qui serait compromise ou dont l'appareil est hors d'usage. Le serveur redonne donc une clé publique, ainsi que la signature qui l'authentifie, au client. Le client relaie ces données à l'Application de Confiance, qui vérifie la validité de la signature. Si elle est bonne, la clé publique du serveur est mise en mémoire sécurisée. Ensuite, l'Application de Confiance génère une clé secrète symétrique, la place aussi en mémoire sécurisée et la chiffre avec la clé publique. La clé chiffrée est alors retournée au serveur, qui la déchiffre et la garde en mémoire. Le routeur et le serveur partagent maintenant une clé symétrique secrète que nul autre ne connaît. Avec l'utilisation du protocole, la clé symétrique remplace le jeton d'authentification dans sa fonction de valeur secrète partagée avec le serveur. Il est à

noter qu'un attaquant en contrôle de l'appareil peut utiliser la clé publique du serveur pour lui envoyer sa propre clé symétrique et déchiffrer de futures messages. Cependant, comme cet échange à lieu à la mise en marche initiale du routeur, il est peu probable qu'il soit déjà infecté. De plus, l'opérateur qui se charge de l'enrôlement pourrait détecter un comportement inhabituel. Néanmoins, cela montre que l'échange repose sur le principe du *Trust on First Use*.

4.8.5 Détails des requêtes

Lors d'une requête authentifiée au serveur, plusieurs étapes devront être suivies. Cependant, le protocole diffère légèrement dans le cas où le client veut envoyer des données au serveur (requêtes POST, UPDATE et PATCH) et celui où le client veut prendre de l'information du serveur (requêtes GET). Si l'API du serveur respecte les principes du REST, il ne devrait pas y avoir de cas où le client envoie et reçoit de l'information en même temps en une seule requête.

Les routeurs sur le terrain ont déjà été enrôlés préalablement par un opérateur. Ils disposent donc dans leur mémoire sécurisée d'une clé symétrique partagée avec le serveur ainsi que de leur identifiant, sous la forme d'une chaîne de caractères. Le serveur dispose pour sa part d'une structure de donnée quelconque dans laquelle chaque routeur est associé avec un identifiant et une clé symétrique unique. De plus, il possède une structure de données associative, comme une *hashmap*, contenant les sessions actives. Chaque entrée associe un *snonce* avec l'estampille temporelle du moment où il a été créé, l'identifiant du client de la session et le *cnonce* qu'il a fourni.

Envoi de données

Plusieurs fonctionnalités du système SentinelC requièrent que le client envoie des données au serveur. Par exemple, si l'administrateur demande de faire un balayage du réseau pour avoir une liste des appareils connectés. Le client doit alors se connecter au serveur pour lui soumettre la liste. Il est important que la requête soit authentifiée à l'aide du protocole sécurisé, car autrement un adversaire pourrait envoyer une liste incomplète, par exemple pour dissimuler ses propres appareils. Les étapes qui doivent être entreprises sont les suivantes.

1. Le client demande à l'Application de Confiance de générer un *cnonce* par le biais de l'application hôte
2. L'Application de Confiance génère le *cnonce* et le place en mémoire sécurisée
3. L'Application de Confiance chiffre le *cnonce* et l'identifiant du client à l'aide de la clé

- partagée et retourne au client le bloc résultant
4. Le client envoie une requête au chemin */nonce* du serveur avec le bloc chiffré
 5. Le serveur nettoie la liste de sessions selon leur estampille temporelle et une durée de vie prédéterminée (quelques minutes au plus)
 6. Le serveur vérifie que le nombre maximal de sessions n'est pas déjà atteint. Si c'est le cas, il retourne un code 503 au client
 7. Le serveur déchiffre le bloc à l'aide de la clé partagée
 8. Le serveur crée un *snonce* et ajoute une session en incluant l'identifiant du client, le *cnonce* et une estampille temporelle
 9. Le serveur chiffre l'identifiant, le *cnonce* et le *snonce* avec la clé partagée en un bloc de données
 10. Si tout s'est bien déroulé, le serveur renvoie un code 200 au client avec le bloc chiffré
 11. Le client passe à l'Application de Confiance, par le biais de l'application hôte, les données à envoyer au serveur ainsi que le bloc chiffré
 12. L'Application de Confiance peut à se moment attester de l'état de l'appareil et refuser de coopérer s'il est compromis
 13. L'Application de Confiance déchiffre le bloc de données et vérifie la validité du *cnonce* et de l'identifiant
 14. L'Application de Confiance chiffre les données, les nonces et l'identifiant avec la clé partagée
 15. L'Application de Confiance supprime le *cnonce* de la mémoire sécurisée
 16. L'Application de Confiance renvoie les données chiffrées au client par le biais de l'application hôte
 17. Le client envoie une requête au chemin voulu sur le serveur avec les données chiffrées
 18. Le serveur utilise la clé partagée pour déchiffrer les données
 19. Le serveur vérifie que le *snonce* est bien dans la liste, que l'identifiant correspond bien à celui du client et que l'estampille temporelle est encore dans le délai acceptable
 20. S'il y a une erreur de déchiffrement, que le *snonce* est invalide ou que l'estampille est trop vieille, le serveur renvoie un code 400
 21. Si tout est valide, le serveur exécute la requête authentifiée
 22. Le serveur supprime la session avec le *snonce* correspondant de la structure de données

Réception de données

Similairement, certaines fonctionnalités demandent au routeur SentinelC de prendre des données du serveur en s'authentifiant. Un exemple courant est le changement de la configuration réseau du routeur. Il est essentiel que le protocole soit rapide, car par défaut les routeurs SentinelC demandent chaque minute au serveur si leur configuration a changé. Les étapes qui doivent être entreprises sont les suivantes.

1. Le client demande à l'Application de Confiance de générer un *cnonce* par le biais de l'application hôte
2. L'Application de Confiance génère le *cnonce* et le place en mémoire sécurisée
3. L'Application de Confiance chiffre le *cnonce* et l'identifiant du client à l'aide de la clé partagée et retourne au client le bloc résultant
4. Le client envoie une requête au chemin voulu sur le serveur avec les données chiffrées
5. Le serveur déchiffre l'identifiant et le *cnonce* à l'aide de la clé partagée
6. Le serveur exécute la requête (qui ne doit pas changer son état)
7. Le serveur chiffre les données résultantes de la requête, ainsi que l'identifiant du client et le *cnonce*, avec la clé partagée
8. Si tout s'est bien déroulé, le serveur renvoie un code 200 au client avec le bloc chiffré
9. Le client passe les données chiffrées à l'Application de Confiance par le biais de l'application hôte
10. L'Application de Confiance peut à ce moment attester de l'état de l'appareil et refuser de coopérer s'il est compromis
11. L'Application de Confiance déchiffre les données avec la clé partagée
12. L'Application de Confiance vérifie la validité de l'identifiant et du *cnonce* et cesse son exécution en cas d'erreur
13. L'Application de Confiance supprime le *cnonce* de la mémoire sécurisée
14. L'Application de Confiance retourne les données déchiffrées au client par le biais de l'application hôte

Ce protocole permet à un routeur SentinelC d'envoyer ou de recevoir des données de la part du serveur associé de manière sécurisée. Il permet au routeur de s'authentifier sans que le jeton d'authentification (dans ce cas-ci la clé symétrique) ne sorte de la sécurité du TEE. De plus, toutes les données envoyées ou reçues du serveur sont chiffrées et illisibles pour un parti externe.

CHAPITRE 5 ÉVALUATION DE LA SOLUTION

Le prototype construit dans la section 4 répond à une partie des objectifs de recherche. Il est pertinent d'évaluer son efficacité sur plusieurs fronts, soit le niveau de sécurité qu'il ajoute à un routeur domestique, l'impact sur la performance générale de l'appareil et la viabilité d'un tel routeur sur le marché. Il est important par la suite d'inspecter les limites de la solution, afin de trouver des pistes pour des travaux futurs.

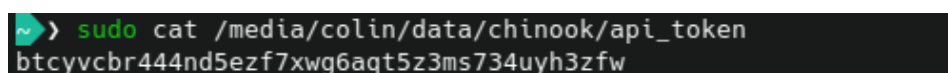
5.1 Sécurité

Afin d'évaluer la sécurité de la solution de manière objective, il faut différencier la sécurité du prototype de la sécurité de la solution finale. La solution finale est l'architecture imaginée au début de l'implémentation et détaillée à la section 4.1. Elle comprend tout d'abord le contenu du prototype, soit OP-TEE et SentinelC sur la plateforme ESPRESSObin. Toutefois, la solution finale comprend aussi un protocole sécurisé entre le client et le serveur SentinelC, ainsi qu'une manière efficace pour l'Application de Confiance d'attester de l'état de la plateforme avant de signer les requêtes. La sécurité du prototype n'est pas complète sans les additions de la solution finale. Il est donc normal que le prototype ait certaines lacunes en matière de sécurité, mais il ne faut pas perdre de vue l'objectif final lors de son évaluation.

5.1.1 Sécurité du prototype

Le Trusted Computing apporte de la sécurité au niveau matériel en isolant un noyau sécurisé et des Applications de Confiance. La suite de tests fournie par OP-TEE et installée sur le prototype nous permet de confirmer que ses fonctionnalités et ses primitives de sécurité sont réellement en place. Sur le prototype, OP-TEE présente 2 avantages notables en matière de sécurité. En premier lieu, il aide à protéger le jeton d'authentification du client SentinelC. Lorsque le jeton est stocké dans la mémoire sécurisée, aucun processus du Monde Normal ne peut y accéder directement. Cela est déjà une bonne amélioration par rapport à l'enregistrement du jeton en texte dans un fichier lisible par tous les utilisateurs. Évidemment, lorsque l'appareil est en marche, la situation est similaire, car n'importe quel utilisateur peut demander le jeton. Toutefois, le deuxième avantage de l'installation de OP-TEE intervient lorsque l'appareil est éteint. Quand c'est le cas, le jeton est tout simplement impossible à extraire. En effet, tous les objets dans la mémoire sécurisée sont chiffrés avec une clé unique. Chaque objet a sa propre clé, qui est dérivée d'une clé unique à son Application de Confiance, elle

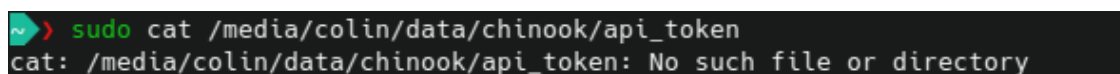
aussi dérivée d'une clé unique à l'appareil. L'objet ainsi chiffré est placé dans le système de fichiers du Monde Normal. Il est donc accessible à tous, mais seule l'Application de Confiance peut en lire le contenu. Puisque la clé unique à l'appareil est générée à partir de valeurs qui ne sont accessibles que lorsque celui-ci est en marche, il est impossible de la retrouver s'il est éteint. Cela est une nette amélioration par rapport à l'implémentation originale, car seul un accès au système de fichier était requis pour récupérer le jeton. La figure 5.1 démontre avec quelle facilité le détenteur de la carte SD peut y lire le jeton après avoir monté la partition de données persistentes sur le chemin `/media/colin/data`.



```
~> sudo cat /media/colin/data/chinook/api_token
btcyvcbr444nd5ezf7xwg6agt5z3ms734uyh3zfw
```

FIGURE 5.1 Lecture du jeton d'authentification sur la carte SD

Cependant, grâce au Trusted Computing, la récupération du jeton sur le prototype est liée à la plateforme matérielle à laquelle il appartient. Un attaquant qui vole la carte SD du prototype en laissant derrière lui le routeur n'aura donc aucune chance de déchiffrer le jeton qui s'y trouve. La figure 5.2 montre que le fichier `api_token` n'est plus présent dans le système de fichiers du prototype.



```
~> sudo cat /media/colin/data/chinook/api_token
cat: /media/colin/data/chinook/api_token: No such file or directory
```

FIGURE 5.2 Le jeton d'authentification n'est pas lisible sur la carte SD du prototype

En effet, sur le prototype, le jeton est contenu dans un fichier chiffré situé dans le dossier `/data/tee`. Le fichier est nommé par un simple chiffre pour ne pas divulguer d'information sur son contenu. Afin de confirmer que le jeton est réellement chiffré, nous pouvons calculer son entropie avec l'utilitaire `ent` [109]. Le fichier contient plusieurs en-têtes ; le contenu de l'objet sécurisé ne commence donc qu'à partir de l'octet 0x3000, ou 12288. La figure 5.3 montre que l'entropie par octet de l'objet vaut presque 8, ce qui démontre qu'il est bel et bien chiffré.

Le jeton ne peut donc pas être extrait du prototype s'il est éteint et il ne peut pas être lu de la mémoire sans l'appareil physique auquel il appartient. Toutefois, il ne faut pas qu'il soit lisible non plus par le même appareil si le système d'exploitation a changé, car cela pourrait consister la base d'une attaque par rejeu ou par rétrogradation. En effet, si un acteur malveillant parvient à voler la carte SD d'un routeur, il sera en mesure de sauvegarder le fichier `/data/tee/2` même s'il ne peut pas en lire le contenu. Lorsque les

```

~> sudo ls -l /media/colin/data/tee/2
-rw----- 1 root root 16384 Dec 31 1969 /media/colin/data/tee/2
~> sudo dd if=/media/colin/data/tee/2 bs=1 skip=12288 | ent
4096+0 records in
4096+0 records out
4096 bytes (4.1 kB, 4.0 KiB) copied, 0.00669834 s, 611 kB/s
Entropy = 7.956399 bits per byte.

Optimum compression would reduce the size
of this 4096 byte file by 0 percent.

Chi square distribution for 4096 samples is 246.75, and randomly
would exceed this value 63.30 percent of the times.

Arithmetic mean value of data bytes is 128.7083 (127.5 = random).
Monte Carlo value for Pi is 3.085043988 (error 1.80 percent).
Serial correlation coefficient is 0.006174 (totally uncorrelated = 0.0).

```

FIGURE 5.3 Entropie du jeton chiffré

administrateurs légitimes de l'appareil auront constaté le vol, ils réinitialiseront le routeur avant de le redéployer. À ce moment, si ce même acteur parvient à compromettre le même appareil, il pourra écraser le nouveau fichier `/data/tee/2` avec l'ancien qu'il a sauvegardé. Si ce nouveau fichier est accepté par le système, le jeton est alors rétrogradé à une valeur précédente potentiellement non sécuritaire. La figure 5.4 montre la récupération du jeton sur le système où il a été créé et enregistré.

```

root@sentinelc-hsbxnt:~# md5sum /data/tee/2
899860a654e7cb0235f4a407dac294d9 /data/tee/2
root@sentinelc-hsbxnt:~# echo $(sentinelc-agent get_token)
emykqpgpku3c4nms59v98vcqdshg7mvnzn22u944

```

FIGURE 5.4 Récupération du jeton sur l'appareil original

Pour simuler ce scénario, nous avons sauvegardé le fichier `/data/tee/2` et nous avons réinstallé le système en entier. Puis, nous avons écrasé le nouveau fichier `/data/tee/2` avec l'ancien. La figure 5.5 montre que l'ancien fichier ne peut être déchiffré par l'Application de Confiance du nouveau système.

Le haché du fichier confirme qu'il s'agit bien du même sur les 2 captures. Le nom d'hôte est cependant différent, ce qui indique que le système a été réinstallé. Cette propriété est attribuable à la manière dont OP-TEE gère la mémoire sécurisée. En effet, certaines données relatives aux chiffrements des objets restent emmagasinées dans le noyau sécurisé. Ces données sont détruites lors de la réinstallation du système, ce qui empêche de telles attaques.

```

root@sentinelc-hxepke:~# md5sum /data/tee/2
899860a654e7cb0235f4a407dac294d9  /data/tee/2
root@sentinelc-hxepke:~# echo $(sentinelc-agent get_token)
E/TC:? 0
E/TC:? 0 TA panicked with code 0xffff000f
E/LD: Status of TA b64b4786-db0a-4519-911b-f80ecee76ea0
E/LD: arch: aarch64
E/LD: region 0: va 0x40004000 pa 0x04800000 size 0x002000 flags rw-s (ldelf)
E/LD: region 1: va 0x40006000 pa 0x04802000 size 0x009000 flags r-xs (ldelf)
E/LD: region 2: va 0x4000f000 pa 0x0480b000 size 0x001000 flags rw-s (ldelf)
E/LD: region 3: va 0x40010000 pa 0x0480c000 size 0x004000 flags rw-s (ldelf)
E/LD: region 4: va 0x40014000 pa 0x04810000 size 0x001000 flags r--s
E/LD: region 5: va 0x40015000 pa 0x04828000 size 0x001000 flags rw-s (stack)
E/LD: region 6: va 0x40016000 pa 0x05001000 size 0x001000 flags rw-- (param)
E/LD: region 7: va 0x4001a000 pa 0x00001000 size 0x00b000 flags r-xs [0]
E/LD: region 8: va 0x40025000 pa 0x0000c000 size 0x00c000 flags rw-s [0]
E/LD: [0] b64b4786-db0a-4519-911b-f80ecee76ea0 @ 0x4001a000
E/LD: Call stack:
E/LD: 0x4001b088
E/LD: 0x4001a150
E/LD: 0x4001bd24
E/LD: 0x4001a1e0
Command GET_TOKEN failed: 0xffff3024 / 3

```

FIGURE 5.5 Tentative de récupération du jeton sur le nouveau système

Sur notre prototype, un attaquant qui réussit à démarrer un système d'exploitation personnalisé de son choix peut théoriquement récupérer les valeurs uniques de l'appareil et s'en servir pour générer les clés de chiffrement utilisées par OP-TEE pour protéger le jeton. Cependant, OP-TEE met en place un démarrage sécurisé pour éviter ce genre d'attaque. Dès le premier stade du démarrage, chaque image vérifie l'intégrité de la prochaine image avant de l'exécuter. Actuellement, le prototype n'écrit pas le premier stade de démarrage dans une ROM afin simplifier le développement, mais cela devra être le cas en production. À ce moment, il sera impossible de démarrer un système qui n'est pas signé par la clé détenue par le fabricant. Cela préviendra quiconque de démarrer un système d'exploitation de son choix pour déchiffrer le jeton. Il est donc sensé d'affirmer qu'outre l'accès par la preuve de concept d'Application de Confiance, il est impossible de lire le jeton d'authentification de SentinelC sur le prototype.

5.1.2 Sécurité de la solution finale

Le prototype actuel laisse n'importe quel processus demander le jeton, car l'Application de Confiance n'a aucun mécanisme pour différencier les processus du Monde Normal entre eux. C'est pourquoi une méthode d'attestation de l'état de l'appareil est nécessaire pour parfaire la sécurité de la solution. Différentes méthodes d'attestation sont discutées dans la section 3.7.

Elles ont toutes leurs points forts et leurs points faibles. Éventuellement, de futures recherches devront déterminer quelle méthode est la plus adaptée à ce routeur sécurisé. Idéalement, l'Application de Confiance devra être en mesure d'attester soit que le processus faisant appel à ses fonctions est légitime, soit que l'appareil en entier est intègre. Cela est réalisable grâce à l'implémentation de TrustZone et de OP-TEE, qui permet aux Applications de Confiance de lire toutes les plages d'adresses du Monde Normal. Plusieurs méthodes potentielles sont décrites à la section 3.7.

Dans la solution finale, l'Application de Confiance devra changer les attributs avec lesquels elle crée l'objet contenant le jeton dans la mémoire sécurisée. Il devra rester lisible, mais ne devra plus être modifiable ni effaçable. Il sera alors impossible de le changer après le premier appel à la fonction *register_token*. La fonction *get_token* devra être remplacée par des fonctions qui serviront à authentifier des données à envoyer au serveur et vérifier les données reçues de celui-ci. À ce point, aucune fonction de l'Application de Confiance ne donnera plus le jeton et celui-ci ne sera utilisable qu'à l'intérieur du Monde Sécurisé.

Une fois que l'Application de Confiance aura un mécanisme d'attestation, il sera possible de l'intégrer dans un protocole d'authentification sécurisé entre le client et le serveur SentinelC. La solution finale, grâce au stockage sécurisé, au mécanisme d'attestation et au protocole d'authentification, réussit à protéger le jeton de tout accès, même lors de son utilisation pour prouver l'identité du routeur auprès du serveur central. Le jeton est protégé quand le routeur est éteint grâce au démarrage sécurisé et à la mémoire sécurisée. Lorsque l'appareil est en marche, le jeton ne peut pas sortir du Monde Sécurisé et n'est lisible que par une seule Application de Confiance. Les données envoyées au serveur sont authentifiées avec des primitives cryptographiques qui ne révèlent en aucun cas le contenu de la clé utilisée. De ce fait, même un attaquant possédant des privilèges racines dans le Monde Normal ne pourra en aucun cas récupérer le contenu du jeton.

Même si le prototype n'a qu'une seule Application de Confiance, l'installation de OP-TEE est une pierre d'assise de sécurité sur laquelle de nombreux autres programmes pourraient s'appuyer dans le futur. La mémoire sécurisée pourrait servir à stocker d'autres secrets, par exemple des clés de chiffrement pour les sessions Wifi. Il faut noter que la sécurité ajoutée par le Monde Sécurisé est conditionnelle à ce qu'il n'y ait pas de faille dans les Applications de Confiance. Il est donc recommandé de maintenir le nombre et la taille des Applications de Confiance relativement petit.

5.1.3 Retour sur l'étude de cas

La section 2.4 décrit comment l'accès physique à un routeur peut faciliter son exploitation. Il est pertinent d'évaluer comment la solution finale de routeur sécurisé se défend face aux techniques qui y sont utilisées.

La majorité des vulnérabilités trouvées sur le routeur de l'étude cas, incluant les plus critiques, sont logées dans le code de son interface web. Sur cet aspect, l'utilisation de SentinelC pour assurer les fonctions de gestion et de configuration est une mesure de sécurité en soi, car son architecture ne comprend tout simplement pas d'interface web. Une bonne partie des failles logicielles sont ainsi évitées. Sur le routeur étudié, le mot de passe administrateur était aussi disponible en clair sur la NVRAM, ce qui n'est pas le cas dans notre solution. En effet, le mot de passe y est haché avec un sel comme dans un système Linux moderne standard.

Les autres failles trouvées à la section 2.4 sont toutes liées au côté matériel du routeur. Le port UART exposé sur la carte mère a permis d'accéder au menu de démarrage et d'extraire le contenu de la mémoire. Il est difficile de juger notre prototype sur cet aspect, car étant encore au stade du développement, il est normal que le menu de démarrage soit facilement accessible avec un simple câble USB. Son accès pourra être désactivé lors de la mise en marché, mais la nature de la carte ESPRESSObin rend très difficile le blocage de toutes les avenues d'exploitation matérielle. Par exemple, les fonctionnalités de récupération par UART permettront toujours à un utilisateur expérimenté de remplacer le micrologiciel de démarrage par un autre, personnalisé, lui permettant d'extraire le contenu de la mémoire. Toutefois, dans les faits, l'extraction de la mémoire n'est pas une fin en soi. Ce n'est qu'une façon de transformer un test boîte noire en test boîte blanche pour faciliter la découverte de vulnérabilités logicielles. Il n'est donc pas essentiel de l'empêcher complètement et il peut même être bénéfique de supposer qu'il sera toujours possible de le faire, afin de ne pas négliger la sécurité des logiciels qui roulent sur l'appareil.

5.2 Performance

Il est essentiel que les modifications du système d'exploitation et du protocole d'authentification ne nuisent pas au fonctionnement du routeur SentinelC. Le démarrage sécurisé ajoute très peu de latence à chaque lancement et n'affecte pas du tout la performance de l'appareil une fois qu'il est en marche. De la même manière, le temps de génération et d'enregistrement du jeton dans la mémoire sécurisée est négligeable et se produit de toute manière à un moment où le routeur n'est pas encore opérationnel. Au contraire, la récupération du jeton par le client SentinelC se doit d'être extrêmement rapide, car ce processus est effec-

tué par défaut chaque minute. Nous avons exécuté la commande *sentinelc-agent get_token* 1000 fois pour obtenir le temps d'accès moyen au jeton et avons obtenu un résultat de 41.4 millisecondes. Ce temps d'accès moyen est suffisamment rapide pour ne pas impacter le fonctionnement du routeur même si la procédure est appelée régulièrement. Il faut cependant noter que la solution finale proposée inclut une phase d'attestation de l'état du routeur à chaque demande d'utilisation du jeton par un processus. En fonction de la méthode utilisée, cette phase pourrait facilement être très chronophage. Par exemple, certaines des méthodes existantes explorées dans la section 3.7 nécessitent de parcourir la liste de tous les processus en cours d'exécution pour inspecter une partie de leur mémoire. Les travaux futurs devront donc prendre en compte la vitesse d'exécution lors du choix de la méthode d'attestation.

5.3 Mise en marché

Un des objectifs de ce projet est d'avoir un prototype qui a les bonnes caractéristiques pour pouvoir éventuellement être mis sur le marché. De ce côté, le choix de la plateforme ESPRESSObin est très avantageux. Avec son bas prix, ses 3 ports Ethernet, ses 2 ports USB et sa petite taille, le ESPRESSObin est le parfait candidat pour la création d'un routeur commercial. Son seul défaut est l'absence de carte Wifi, qui est essentielle dans un contexte de réseautique domestique. La plupart des foyers ont des ordinateurs, téléphones et autres appareils qui ont besoin d'une connexion sans fil à l'internet. Ce défaut pourrait être corrigé par l'ajout d'une carte Wifi USB.

Ainsi que mentionné plus haut, la carte ESPRESSObin est par définition une carte de développement. Dans l'éventualité où le prototype est mis en marché, il serait intéressant d'évaluer une possible modification de la plateforme. Il serait possible de réduire le coût de production et d'augmenter la sécurité matérielle en enlevant certaines composantes superflues comme le circuit de récupération par UART.

Un obstacle possible à la mise en marché de l'architecture proposée est la difficulté d'implémentation de la solution. La courbe d'apprentissage de Yocto est très abrupte et peut poser problème pour de nouveaux développeurs. Malgré tout, Yocto est un outil très fréquemment utilisé dans le domaine des appareils embarqués et beaucoup de développeurs en sont déjà familiers. La compilation du système à partir de rien prend une dizaine d'heures et au-dessus d'une soixantaine de Gigabyte (GB). Cette longue durée pourrait décourager de nouveaux développeurs à essayer le projet. Malheureusement, la nature même de Yocto tend vers une longue compilation et une grande utilisation du disque, car tous les outils de compilation doivent eux-mêmes être téléchargés et compilés. Cependant, la solution comporte énormément de recettes qui ne sont pas nécessaires. Cela est dû à la fusion de plusieurs projets Yocto

comprenant beaucoup de recettes parfois redondantes ou superflues. Un travail d'émondage des recettes non nécessaires pourrait améliorer les performances de la compilation et simplifier la compréhension du projet par de nouveaux développeurs.

5.4 Limites de la solution

La solution dans son état actuel possède plusieurs limites. Certaines sont inhérentes à l'état de prototype, comme le fait que le démarrage sécurisé utilise une clé de chiffrement de développement, qui n'est pas écrite dans la mémoire en lecture seule de l'appareil. La clé ne peut donc pas constituer une réelle racine de confiance, ce qui met en jeu toute la sécurité apportée par l'utilisation du Trusted Computing. Toutefois, il sera aisé de régler ce problème lors de la production, car la plateforme ESPRESSObin contient des fusibles microscopiques (eFuse) qui ne sont modifiables qu'une seule fois. La méthodologie pour y écrire une clé servant au démarrage sécurisé est disponible dans la documentation du fabricant [110]. La plateforme de développement ESPRESSObin possède aussi des fonctionnalités qui nuisent à la sécurité de la solution, comme la récupération par UART. De plus, le code de SentinelC ne supporte pas très bien les commutateurs réseau intégrés comme celui utilisé par la plateforme ESPRESSObin. Pour donner un accès à l'internet au prototype, il faut exécuter quelques commandes manuellement par le port UART, ce qui nuit grandement à la facilité d'utilisation.

Une des limites du côté de la sécurité vient avec l'architecture de TrustZone pour les processeurs Cortex-A. Chaque Application de Confiance s'exécute avec des privilèges élevés et peut compromettre la sécurité de tout le Monde Sécurisé en cas de brèche. Il serait donc préférable que plusieurs personnes fassent la vérification du code de toutes les Applications de Confiance devant être installées sur l'appareil.

Finalement, il arrive parfois que le système d'exploitation du prototype rencontre une erreur fatale, forçant le routeur à redémarrer. Les traces indiquent qu'il s'agit surtout de fautes de pages. Il peut être très difficile de comprendre et de réparer de telles erreurs dans un système aussi complexe. Il serait important dans le futur que le prototype ait une plus grande stabilité. Toutefois, il est possible que le problème provienne d'une composante externe, comme l'implémentation du noyau Linux par Linaro, auquel cas sa remédiation est hors de la portée de ce projet.

CHAPITRE 6 CONCLUSION

Ce chapitre conclut le mémoire en faisant une synthèse des travaux réalisés. Il passe en revue les objectifs atteints pour ensuite répondre à la question principale de recherche. Les contributions apportées sont énumérées et différentes améliorations sont proposées pour amener le projet encore plus loin avec de futures recherches.

6.1 Synthèse des travaux

6.1.1 Retour sur les objectifs

La question principale de recherche a été divisée en objectifs secondaires permettant sa résolution. Voici comment ils ont été atteints :

1. Faire un état de la sécurité des routeurs domestiques sur le marché, des attaques les plus courantes contre ceux-ci et des solutions existantes pour en améliorer la sécurité. Le chapitre 2 commence par séparer les différents attaquants en 3 types distincts. Il fait ensuite un état de la sécurité des routeurs sur le marché à la fois sur le plan logiciel et matériel. La sous-section 2.4 donne un exemple concret d'une méthodologie d'exploitation d'un routeur et du faible niveau de sécurité de certains modèles. Les sections 2.2 et 2.3 font la revue des dernières avancées en matière de sécurité logicielle et matérielle, respectivement, pour les routeurs et autres appareils embarqués. Le chapitre 3 examine aussi des solutions basées sur le Trusted Computing.
2. Identifier les informations les plus sensibles et les plus importantes à protéger sur un routeur domestique. La section 4.2.1 décrit le fonctionnement du routeur SentinelC et identifie le jeton d'authentification comme une information sensible cruciale à son niveau de sécurité.
3. Déterminer les fonctionnalités du Trusted Computing pouvant aider à protéger ces informations. Le chapitre 3 expose en détail les principes du Trusted Computing et ses différents éléments. Plus loin, la section 4.2.2 énumère plusieurs implémentations d'environnement d'exécution de confiance et identifie OP-TEE comme étant le mieux adapté pour ce projet.
4. Créer un prototype de routeur intégrant ces fonctionnalités du Trusted Computing afin de protéger une information sensible liée à ses fonctions. Le chapitre 4 passe en revue toutes les étapes de l'implémentation d'un prototype, incluant le choix des technologies utilisées. Le prototype comprend une installation de OP-TEE et des logiciels

de SentinelC sur une plateforme répondant aux exigences d'un routeur. Dans la section 4.6.3, le client SentinelC est modifié pour tirer profit de la sécurité offerte par l'installation de OP-TEE.

5. Tester si la solution protège bel et bien l'information sensible contre les attaques décrites précédemment. La section 5.1 évalue le niveau de sécurité du prototype et de la solution finale proposée. La section 5.1.3 revient sur les failles trouvées dans l'étude de cas et détermine si le prototype leur est vulnérable.
6. Explorer des pistes pour une éventuelle mise en marché. Les caractéristiques du prototype pouvant affecter une potentielle mise en marché sont abordées dans la section 5.3.

Toutes ces étapes nous permettent de statuer que oui, *il est possible d'intégrer le Trusted Computing dans l'architecture d'un routeur pour protéger une information sensible contre des attaques logicielles et matérielles.*

6.1.2 Contributions

Ce faisant, ce projet apporte les contributions suivantes :

1. Compte rendu de l'état actuel de la sécurité des routeurs sur le marché
2. Étude de cas sur l'exploitation d'un routeur par l'accès physique
3. Modification des couches Yocto existantes pour supporter OP-TEE sur la plateforme ESPRESSObin
4. Création d'une preuve de concept d'Application de Confiance pour protéger un jeton d'authentification
5. Création d'une couche Yocto intégrant OP-TEE, SentinelC OS et l'Application de Confiance sur la plateforme ESPRESSObin

Tout le code mentionné dans les contributions est ouvert et disponible sur GitHub.

6.2 Améliorations futures

Ce mémoire décrit l'implémentation d'un prototype, qui, par définition, doit être amélioré pour en arriver à une solution complète. En premier lieu, il serait souhaitable que le jeton d'authentification soit généré à l'intérieur de l'Application de Confiance au lieu de lui être fourni par le biais du programme *sentinelc-agent*. Pour aller plus loin, la principale amélioration laissée à des travaux futurs est la création et l'implémentation d'un protocole d'authentification sécurisé entre le routeur et le serveur SentinelC. Cela permettra de véritablement concrétiser l'apport du Trusted Computing en matière de sécurité pour le routeur SentinelC.

De manière complémentaire, il faudra déterminer une méthode d’attestation de l’état du système par l’Application de Confiance responsable de l’utilisation du jeton d’authentification. En particulier, il serait souhaitable que l’Application de Confiance puisse confirmer que le processus du Monde Normal qui lui fait une demande est bien le client SentinelC. Autrement, n’importe quel processus est en mesure de faire utilisation du jeton. Il est toutefois important de noter que cela reste une amélioration par rapport à la situation originale, car même si n’importe quel processus peut faire utilisation du jeton, toutes les demandes d’utilisations doivent passer par l’appareil sur lequel il a été généré, car le jeton ne peut sortir de la mémoire sécurisée.

En deuxième lieu, certaines améliorations pourraient faciliter l’utilisation et la reprise du prototype par d’autres développeurs. Par exemple, de nombreuses recettes Yocto sont compilées inutilement lors de la première compilation, ce qui fait perdre une bonne quantité de temps et d’espace de stockage aux développeurs. De plus, il serait souhaitable de régler les problèmes de compatibilité de SentinelC avec les cartes réseau du ESPRESSObin et les occasionnelles fautes de page dans le noyau.

Finalement, il serait intéressant d’explorer l’intégration du Trusted Computing dans d’autres fonctionnalités d’un routeur afin de tirer profit au maximum de son installation. Éventuellement, il serait remarquable d’intégrer le Trusted Computing dans des projets de micrologiciels de routeurs libres pour faire profiter le plus de gens possible de la sécurité que cette technologie peut apporter. Il serait aussi intéressant de porter le projet sur une plateforme possédant un TPM à mesure que cette technologie se démocratise et que son prix baisse.

RÉFÉRENCES

- [1] E. G. Record-breaking number of router security flaws discovered in the last few years. [En ligne]. Disponible : <https://atlasvpn.com/blog/record-breaking-number-of-router-security-flaws-discovered-in-the-last-few-years>
- [2] P. Weidenbach et J. vom Dorp, “Home router security report 2020,” Fraunhofer Institute for Communication, Information Processing and Ergonomics FKIE, Rapport technique, June 2020.
- [3] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas et Y. Zhou, “Understanding the mirai botnet,” dans *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC : USENIX Association, août 2017, p. 1093–1110. [En ligne]. Disponible : <https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis>
- [4] AT&T. (2021) At&t alien labs finds new golang malware (botenago) targeting millions of routers and iot devices with more than 30 exploits. [En ligne]. Disponible : <https://cybersecurity.att.com/blogs/labs-research/att-alien-labs-finds-new-golang-malwarebotenago-targeting-millions-of-routers-and-iot-devices-with>
- [5] (2018, mai) New VPNFilter malware targets at least 500K networking devices worldwide. [En ligne]. Disponible : <https://blog.talosintelligence.com/vpnfilter/>
- [6] Over 200,000 MikroTik Routers Compromised in Cryptojacking Campaign. [En ligne]. Disponible : <https://www.trendmicro.com/vinfo/ru/security/news/cybercrime-and-digital-threats/over-200-000-mikrotik-routers-compromised-in-cryptojacking-campaign>
- [7] R. Brandom. (2017, juin) The CIA has lots of ways to hack your router. [En ligne]. Disponible : <https://www.theverge.com/2017/6/15/15812216/cherryblossom-cia-router-hack-surveillance-dlink-linksys-belkin>
- [8] C. S. Writer. (2014, oct.) Tor exit node hacked to inject malicious code in downloads. [En ligne]. Disponible : <https://techmonitor.ai/technology/cybersecurity/tor-exit-node-hacked-to-inject-malicious-code-in-downloads-291014-4420718>
- [9] D. Goodin. (2022, juin) A wide range of routers are under attack by new, unusually sophisticated malware. [En

- ligne]. Disponible : <https://arstechnica.com/information-technology/2022/06/a-wide-range-of-routers-are-under-attack-by-new-unusually-sophisticated-malware/>
- [10] S. M. Kerner, “Ransomware Trends, Statistics and Facts in 2023.” [En ligne]. Disponible : <https://www.techtarget.com/searchsecurity/feature/Ransomware-trends-statistics-and-facts>
- [11] P. K. Sharma, J. H. Park, Y.-S. Jeong et J. H. Park, “Shsec : Sdn based secure smart home network architecture for internet of things,” *Mobile Networks and Applications*, vol. 24, n^o. 3, p. 913–924, Jun 2019. [En ligne]. Disponible : <https://doi.org/10.1007/s11036-018-1147-3>
- [12] A. Cui, Y. Song, P. V. Prabhu et S. J. Stolfo, “Brave new world : Pervasive insecurity of embedded network devices,” dans *Recent Advances in Intrusion Detection*, E. Kirda, S. Jha et D. Balzarotti, édit. Berlin, Heidelberg : Springer Berlin Heidelberg, 2009, p. 378–380.
- [13] 277,000 routers exposed to Eternal Silence attacks via UPnP. [En ligne]. Disponible : <https://www.bleepingcomputer.com/news/security/277-000-routers-exposed-to-eternal-silence-attacks-via-upnp/>
- [14] Actively exploited bug bypasses authentication on millions of routers. [En ligne]. Disponible : <https://www.bleepingcomputer.com/news/security/actively-exploited-bug-bypasses-authentication-on-millions-of-routers/>
- [15] D. Goodin. (2014, févr.) Dear Asus router user : You’ve been pwned, thanks to easily exploited flaw. [En ligne]. Disponible : <https://arstechnica.com/information-technology/2014/02/dear-asus-router-user-youve-been-pwned-thanks-to-easily-exploited-flaw/>
- [16] Attackers Actively Exploiting Realtek SDK Flaws. [En ligne]. Disponible : <https://threatpost.com/attackers-exploiting-realtek/168856/>
- [17] Lucian Constantin. (2020) Solarwinds attack explained : And why it was so hard to detect. [En ligne]. Disponible : <https://www.csoonline.com/article/3601508/solarwinds-supply-chain-attack-explained-why-organizations-were-not-prepared.html>
- [18] James Clee. (2020) Multiple vulnerabilities in wavlink router leads to unauthenticated rce – cve-2020-10971 and cve-2020-10972. [En ligne]. Disponible : <https://james-clee.com/2020/04/18/multiple-wavlink-vulnerabilities/>
- [19] ——. (2020) More information disclosure in wavlink devices : Cve-2020-10973, cve-2020-10974, and cve-2020-12266. [En ligne]. Disponible : <https://james-clee.com/2020/04/23/more-information-disclosure-in-wavlink-devices/>
- [20] What is the SentinelC project ? | SentinelC. [En ligne]. Disponible : <https://www.sentinelc.com/docs/>

- [21] OWASP Firmware Security Testing Methodology. [En ligne]. Disponible : <https://scriptingxss.gitbook.io/firmware-security-testing-methodology/>
- [22] u-boot. (2022) "das u-boot" source tree. [En ligne]. Disponible : <https://github.com/u-boot/u-boot>
- [23] M. Sabt, M. Achemlal et A. Bouabdallah, "Trusted execution environment : What it is, and what it is not," dans *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 1, 2015, p. 57–64. [En ligne]. Disponible : <https://ieeexplore.ieee.org/document/7345265>
- [24] D. Goodin. (2010) Ex-Army man cracks popular security chip. [En ligne]. Disponible : https://www.theregister.com/2010/02/17/infineon_tpm_crack/
- [25] J. J. Cranford. Three Pillars of Infosec : Confidentiality, Integrity and Availability. [En ligne]. Disponible : <https://www.cybereason.com/blog/three-pillars-of-infosec-confidentiality-integrity-and-availability>
- [26] D. d. Santos. (2020, déc.) AMNESIA :33 – Forescout Research Labs Finds 33 New Vulnerabilities in Open Source TCP/IP Stacks. [En ligne]. Disponible : <https://www.forescout.com/blog/amnesia33-forescout-research-labs-finds-33-new-vulnerabilities-in-open-source-tcp-ip-stacks/>
- [27] E. U. Küçüksille, M. A. Yalçinkaya et S. Ganai, "Developing a penetration test methodology in ensuring router security and testing it in a virtual laboratory," dans *Proceedings of the 8th International Conference on Security of Information and Networks*, ser. SIN '15. New York, NY, USA : Association for Computing Machinery, 2015, p. 189–195. [En ligne]. Disponible : <https://doi.org/10.1145/2799979.2799989>
- [28] F. FKIE. About — FACT documentation. [En ligne]. Disponible : https://fkie-cad.github.io/FACT_core/index.html
- [29] F. Weimer. (2018, mars) Recommended compiler and linker flags for GCC. Section : C++. [En ligne]. Disponible : <https://developers.redhat.com/blog/2018/03/21/compiler-and-linker-flags-gcc>
- [30] N. Ammari, "Defeating Code-Reuse Attacks with Binary Instrumentation," Mémoire de maîtrise, École Polytechnique de Montréal, 2018. [En ligne]. Disponible : <https://publications.polymtl.ca/3243/>
- [31] Qu'est-ce que la conteneurisation ? – La conteneurisation expliquée – AWS. [En ligne]. Disponible : <https://aws.amazon.com/fr/what-is/containerization/>
- [32] OPNsense® a true open source security platform and more. [En ligne]. Disponible : <https://opnsense.org/>

- [33] C. Thomasset, “Sereniot : politiques de sécurité collaboratives pour maisons connectées,” Mémoire de maîtrise, Polytechnique Montréal, July 2020. [En ligne]. Disponible : <https://publications.polymtl.ca/5322/>
- [34] A. Dorri, S. S. Kanhere, R. Jurdak et P. Gauravaram, “LSB : A Lightweight Scalable Blockchain for IoT security and anonymity,” *Journal of Parallel and Distributed Computing*, vol. 134, p. 180–197, 2019. [En ligne]. Disponible : <https://www.sciencedirect.com/science/article/pii/S0743731518307688>
- [35] A. Dua, V. Tyagi, N. Patel et B. Mehtre, “Iisr : A secure router for iot networks,” dans *2019 4th International Conference on Information Systems and Computer Networks (ISCON)*, 2019, p. 636–643.
- [36] Snort - Network Intrusion Detection & Prevention System. [En ligne]. Disponible : <https://www.snort.org/>
- [37] S. Tan, X. Li et Q. Dong, “Trustr : an integrated router security framework for protecting computer networks,” *IEEE Communications Letters*, vol. 20, n^o. 2, p. 376 – 9, 2016/02/.
- [38] Rust Programming Language. [En ligne]. Disponible : <https://www.rust-lang.org/>
- [39] A. Machiry, E. Gustafson, C. Spensky, C. Salls, N. Stephens, R. Wang, A. Bianchi, Y. R. Choe, C. Kruegel et G. Vigna, “Boomerang : Exploiting the semantic gap in trusted execution environments,” dans *Proceedings of the Network and Distributed System Security Symposium*, 2017.
- [40] K. Eldefrawy, D. Perito et G. Tsudik, “Smart : Secure and minimal architecture for (establishing a dynamic) root of trust,” 01 2012.
- [41] J. Noorman, P. Agten, W. Daniels, R. Strackx, A. V. Herrewewege, C. Huygens, B. Preneel, I. Verbauwhede et F. Piessens, “Sancus : Low-cost trustworthy extensible networked devices with a zero-software trusted computing base,” dans *22nd USENIX Security Symposium (USENIX Security 13)*. Washington, D.C. : USENIX Association, août 2013, p. 479–498. [En ligne]. Disponible : <https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/noorman>
- [42] J. Noorman, J. V. Bulck, J. T. Mühlberg, F. Piessens, P. Maene, B. Preneel, I. Verbauwhede, J. Götzfried, T. Müller et F. Freiling, “Sancus 2.0 : A low-cost security architecture for iot devices,” *ACM Trans. Priv. Secur.*, vol. 20, n^o. 3, jul 2017. [En ligne]. Disponible : <https://doi.org/10.1145/3079763>
- [43] L. Guan, P. Liu, X. Xing, X. Ge, S. Zhang, M. Yu et T. Jaeger, “Trustshadow : Secure execution of unmodified applications with arm trustzone,” dans *Proceedings of the*

- 15th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '17. New York, NY, USA : Association for Computing Machinery, 2017, p. 488–501. [En ligne]. Disponible : <https://doi.org/10.1145/3081333.3081349>
- [44] P. Koeberl, S. Schulz, V. Varadharajan et A.-R. Sadeghi, “Trustlite : A security architecture for tiny embedded devices,” 04 2014.
 - [45] Gordon Lyon. (2020) Nmap : the network mapper - free security scanner. [En ligne]. Disponible : <https://nmap.org>
 - [46] PortSwigger. (2022) Burp suite - application security testing software. [En ligne]. Disponible : <https://portswigger.net/burp>
 - [47] Attify Team. (2016) Introduction to attify badge : Hacking iot hardware. [En ligne]. Disponible : <https://blog.attify.com/hack-iot-device/>
 - [48] BSD. (2022) screen(1) - linux man page. [En ligne]. Disponible : <https://linux.die.net/man/1/screen>
 - [49] Philippe Lucidarme. (2021) Most common baud rates table. [En ligne]. Disponible : <https://lucidar.me/en/serialib/most-used-baud-rates-table/>
 - [50] Deral Heiland. (2022) Lessons in iot hacking : How to dead-bug a bga flash memory chip. [En ligne]. Disponible : <https://www.rapid7.com/blog/post/2022/04/07/lessons-in-iot-hacking-how-to-dead-bug-a-bga-flash-memory-chip/>
 - [51] GigaDevice. (2020) Gd25q127c datasheet. [En ligne]. Disponible : <https://www.gigadevice.com/datasheet/gd25q127c/>
 - [52] flashrom. (2022) flashrom. [En ligne]. Disponible : <https://www.flashrom.org/Flashrom>
 - [53] Micron Technology. (2006) Nand flash 101 : An introduction to nand flash and how to design it into your next product. [En ligne]. Disponible : <https://user.eng.umd.edu/~blj/CS-590.26/micron-tn2919.pdf>
 - [54] FreeRTOS. (2022) Freertos - market leading rtos (real time operating system) for embedded systems with internet of things extensions. [En ligne]. Disponible : <https://www.freertos.org/>
 - [55] ReFirmLabs. (2021) binwalk. [En ligne]. Disponible : <https://github.com/ReFirmLabs/binwalk>
 - [56] Frank Hofmann. (2019) Understanding the elf file format. [En ligne]. Disponible : https://linuxhint.com/understanding_elf_file_format/
 - [57] Erik Andersen. (2021) Busybox. [En ligne]. Disponible : <https://busybox.net/>
 - [58] GoAhead. (2016) Goactions. [En ligne]. Disponible : <https://www.embedthis.com/goahead/doc/users/goactions.html>

- [59] ——. (2022) Embedded web server - goahead | embedded web servers | embedthis software. [En ligne]. Disponible : <https://www.embedthis.com/goahead/doc/>
- [60] National Security Agency. (2022) Ghidra. [En ligne]. Disponible : <https://ghidra-sre.org/>
- [61] BSD. (1999) ash(1) - command interpreter - linux man page. [En ligne]. Disponible : <https://linux.die.net/man/1/ash>
- [62] Mendez, Xavi. (2017) Wfuzz : The web fuzzer - wfuzz 2.1.4 documentation. [En ligne]. Disponible : <https://wfuzz.readthedocs.io/en/latest/>
- [63] NVD - CVSS v3 Calculator. [En ligne]. Disponible : <https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator>
- [64] “OWASP Secure Coding Practices-Quick Reference Guide | OWASP Foundation.” [En ligne]. Disponible : https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/migrated_content.html
- [65] S. Foster. What Is CERT C? Overview of CERT Secure Coding. [En ligne]. Disponible : <https://www.perforce.com/blog/kw/what-is-cert>
- [66] Microsoft. How Windows uses the TPM. [En ligne]. Disponible : <https://learn.microsoft.com/en-us/windows/security/information-protection/tpm/how-windows-uses-the-tpm>
- [67] M. Jadla, “L’authentification sécurisée utilisant le trusted computing,” Mémoire de maîtrise, École Polytechnique de Montréal, July 2018. [En ligne]. Disponible : <https://publications.polymtl.ca/3179/>
- [68] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch et S. Winwood, “Sel4 : Formal verification of an os kernel,” dans *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, ser. SOSP '09. New York, NY, USA : Association for Computing Machinery, 2009, p. 207–220. [En ligne]. Disponible : <https://doi.org/10.1145/1629575.1629596>
- [69] Discover YubiKeys | Strong Two-Factor Authentication for Secure Login. [En ligne]. Disponible : <https://www.yubico.com/products/>
- [70] W. Arthur et D. Challener, *A Practical Guide to TPM 2.0 : Using the Trusted Platform Module in the New Age of Security*, 1^{er} éd. USA : Apress, 2015.
- [71] S. Zhao, Q. Zhang, G. Hu, Y. Qin et D. Feng, “Providing root of trust for arm trustzone using on-chip sram,” dans *Proceedings of the 4th International Workshop on Trustworthy Embedded Devices*, ser. TrustED '14. New York, NY, USA :

- Association for Computing Machinery, 2014, p. 25–36. [En ligne]. Disponible : <https://doi.org/10.1145/2666141.2666145>
- [72] C. Helfmeier, C. Boit, D. Nedospasov et J.-P. Seifert, “Cloning physically unclonable functions,” dans *2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2013, p. 1–6.
- [73] O. Khalid, C. Rolfes et A. Ibing, “On implementing trusted boot for embedded systems,” dans *2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2013, p. 75–80.
- [74] P. Maene, J. Götzfried, R. de Clercq, T. Müller, F. Freiling et I. Verbauwhede, “Hardware-based trusted computing architectures for isolation and attestation,” *IEEE Transactions on Computers*, vol. 67, n°. 3, p. 361–374, 2018. [En ligne]. Disponible : <https://ieeexplore.ieee.org/abstract/document/7807249>
- [75] J. Wang, Y. Shi, G. Peng, H. Zhang, B. Zhao, F. Yan, F. Yu et L. Zhang, “Survey on key technology development and application in trusted computing,” *China Communications*, vol. 13, n°. 11, p. 70–90, 2016. [En ligne]. Disponible : <https://ieeexplore.ieee.org/document/7781720>
- [76] S. Pinto et N. Santos, “Demystifying arm trustzone : A comprehensive survey,” *ACM Comput. Surv.*, vol. 51, n°. 6, janv. 2019. [En ligne]. Disponible : <https://doi.org/10.1145/3291047>
- [77] (2023, janv.) Trusted Firmware-A. Original-date : 2013-10-21T11 :39 :00Z. [En ligne]. Disponible : <https://github.com/ARM-software/arm-trusted-firmware>
- [78] (2013) ARM Trusted Firmware Design. [En ligne]. Disponible : <https://chromium.googlesource.com/external/github.com/ARM-software/arm-trusted-firmware/+v0.4-rc1/docs/firmware-design.md>
- [79] (2023, janv.) Trusted Board Boot. Original-date : 2013-10-21T11 :39 :00Z. [En ligne]. Disponible : <https://github.com/ARM-software/arm-trusted-firmware/blob/master/docs/design/trusted-board-boot.rst>
- [80] E. Sanfeliix. (2019) TEE Exploitation | Bluefrostsecurity. [En ligne]. Disponible : <https://labs.bluefrostsecurity.de/publications/2019/05/27/exploiting-trusted-apps-on-samsungs-tee/>
- [81] A. Francillon, Q. Nguyen, K. B. Rasmussen et G. Tsudik, “A minimalist approach to remote attestation,” dans *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2014, p. 1–6.

- [82] K. ElDefrawy, N. Rattanaivanon et G. Tsudik, “Hydra : Hybrid design for remote attestation (using a formally verified microkernel),” 2017. [En ligne]. Disponible : <https://arxiv.org/abs/1703.02688>
- [83] Z. Ling, H. Yan, X. Shao, J. Luo, Y. Xu, B. Pearson et X. Fu, “Secure boot, trusted boot and remote attestation for arm trustzone-based iot nodes,” *Journal of Systems Architecture*, vol. 119, p. 102240, 2021. [En ligne]. Disponible : <https://www.sciencedirect.com/science/article/pii/S1383762121001661>
- [84] T. Wehbe, V. Mooney et D. Keezer, “Hardware-based run-time code integrity in embedded devices,” *Cryptography*, vol. 2, n°. 3, 2018. [En ligne]. Disponible : <https://www.mdpi.com/2410-387X/2/3/20>
- [85] A. Rein, “Drive : Dynamic runtime integrity verification and evaluation,” dans *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ser. ASIA CCS '17. New York, NY, USA : Association for Computing Machinery, 2017, p. 728–742. [En ligne]. Disponible : <https://doi.org/10.1145/3052973.3052975>
- [86] S. Wan, J. Sun, K. Sun, N. Zhang et Q. Li, “Satin : A secure and trustworthy asynchronous introspection on multi-core arm processors,” dans *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2019, p. 289–301.
- [87] embeDD GmbH. DD-WRT. [En ligne]. Disponible : <https://dd-wrt.com/>
- [88] Shibby. Tomato by Shibby » Tomato Firmware. [En ligne]. Disponible : https://tomato.groov.pl/?page_id=164
- [89] R. Brown. (2016, sept.) Welcome to the OpenWrt Project. Section : 2022-09-05T19 :03 :56-04 :00. [En ligne]. Disponible : <https://openwrt.org/start>
- [90] A. Fitzek, F. Achleitner, J. Winter et D. Hein, “The andix research os — arm trustzone meets industrial control systems security,” dans *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*, 2015, p. 88–93.
- [91]
- [92] Genode - Genode Operating System Framework. [En ligne]. Disponible : <https://genode.org/>
- [93] SierraTEE Virtualization for ARM TrustZone and MIPS. [En ligne]. Disponible : <https://www.sierraware.com/open-source-ARM-TrustZone.html>
- [94] TEE fidèle. [En ligne]. Disponible : <https://source.android.com/docs/security/features/trusty?hl=fr>

- [95] Intel® x86 Architecture Distribution of Trusty* OS. [En ligne]. Disponible : <https://www.intel.com/content/www/us/en/developer/topic-technology/open/trusty/overview.html>
- [96] About OP-TEE — OP-TEE documentation documentation. [En ligne]. Disponible : <https://optee.readthedocs.io/en/latest/general/about.html>
- [97] W. Huang, V. Rudchenko, H. Shuang, Z. Huang et D. Lie, “Pearl-tee : Supporting untrusted applications in trustzone,” 10 2018, p. 8–13.
- [98] A. Ltd. CPU Architecture. [En ligne]. Disponible : <https://www.arm.com/architecture/cpu>
- [99] ODYSSEY - X86J4105864 Most expandable Win10 Mini PC (Linux and Arduino Core) with 8GB RAM + 64GB eMMC. [En ligne]. Disponible : <https://www.seeedstudio.com/ODYSSEY-X86J4105864-p-4447.html>
- [100] Platforms supported — OP-TEE documentation documentation. [En ligne]. Disponible : <https://optee.readthedocs.io/en/latest/general/platforms.html>
- [101] Raspberry Pi 3 — OP-TEE documentation documentation. [En ligne]. Disponible : <https://optee.readthedocs.io/en/latest/building/devices/rpi3.html>
- [102] S. Verma, “U-boot environment variables,” *Linux J.*, vol. 2014, n°. 246, oct 2014.
- [103] ESPRESSObin Wiki | Bootloader recovery via UART. [En ligne]. Disponible : <http://wiki.espressobin.net/tiki-index.php?page=Bootloader+recovery+via+UART>
- [104] bootparam(7) - Linux manual page. [En ligne]. Disponible : <https://man7.org/linux/man-pages/man7/bootparam.7.html>
- [105] Google. git-repo - Git at Google. [En ligne]. Disponible : <https://gerrit.googlesource.com/git-repo>
- [106] Internet Engineering Task Force (IETF). (2015) Rfc 7617 - the 'basic' http authentication scheme. [En ligne]. Disponible : <https://datatracker.ietf.org/doc/html/rfc7617>
- [107] F. Alliance. How FIDO Works - Standard Public Key Cryptography & User Privacy. [En ligne]. Disponible : <https://fidoalliance.org/how-fido-works/>
- [108] W. A. Simpson, “PPP Challenge Handshake Authentication Protocol (CHAP),” Internet Engineering Task Force, Request for Comments RFC 1994, août 1996, num Pages : 13. [En ligne]. Disponible : <https://datatracker.ietf.org/doc/rfc1994>
- [109] Ubuntu Manpage : ent - pseudorandom number sequence test. [En ligne]. Disponible : <https://manpages.ubuntu.com/manpages/bionic/man1/ent.1.html>

- [110] MarvellEmbeddedProcessors. (2018) Trusted boot (secure mode). [En ligne]. Disponible : https://github.com/MarvellEmbeddedProcessors/u-boot-marvell/blob/u-boot-2017.03-armada-17.10/doc/mvebu/trusted_boot.txt

ANNEXE A CORRECTION DE CODE POUR AJOUTER LE DISPOSITIF OP-TEE À L'ARBORESCENCE DES PÉRIPHÉRIQUES

```

1 diff --git a/arch/arm64/boot/dts/marvell/armada-3720-espressobin-v7-emmc.dts ↵
    dts b/arch/arm64/boot/dts/marvell/armada-3720-espressobin-v7-emmc.dts
2 index 75401eab4d42..b1b89d9ede67 100644
3 --- a/arch/arm64/boot/dts/marvell/armada-3720-espressobin-v7-emmc.dts
4 +++ b/arch/arm64/boot/dts/marvell/armada-3720-espressobin-v7-emmc.dts
5 @@ -36,10 +36,6 @@ &switch0port3 {
6         label = "wan";
7     };
8
9 -&sdhci0 {
10 -     status = "okay";
11 -};
12 -
13     &led2 {
14         status = "okay";
15     };
16 diff --git a/arch/arm64/boot/dts/marvell/armada-3720-espressobin.dtsi b/↵
    arch/arm64/boot/dts/marvell/armada-3720-espressobin.dtsi
17 index 5fc613d24151..f8f4b065f9e2 100644
18 --- a/arch/arm64/boot/dts/marvell/armada-3720-espressobin.dtsi
19 +++ b/arch/arm64/boot/dts/marvell/armada-3720-espressobin.dtsi
20 @@ -28,9 +28,9 @@ memory0 {
21         reg = <0x00000000 0x00000000 0x00000000 0x20000000>;
22     };
23
24 -     vcc_sd_reg1: regulator {
25 +     vcc_sd_reg0: regulator {
26         compatible = "regulator-gpio";
27 -         regulator-name = "vcc_sd1";
28 +         regulator-name = "vcc_sd0";
29         regulator-min-microvolt = <1800000>;
30         regulator-max-microvolt = <3300000>;
31         regulator-boot-on;
32 @@ -70,7 +70,7 @@ &sata {
33     };
34
35     /* U11 */
36 -&sdhci0 {

```

```

37 +&sdhci1 {
38     /* Main DTS file for Espressobin is without eMMC */
39     status = "disabled";
40
41 @@ -94,12 +94,12 @@ mmccard: mmccard@0 {
42     };
43
44     /* J1 */
45 -&sdhci1 {
46 +&sdhci0 {
47     wp-inverted;
48     bus-width = <4>;
49     cd-gpios = <&gpionb 3 GPIO_ACTIVE_LOW>;
50     marvell,pad-type = "sd";
51 -    vqmmc-supply = <&vcc_sd_reg1>;
52 +    vqmmc-supply = <&vcc_sd_reg0>;
53
54     pinctrl-names = "default";
55     pinctrl-0 = <&sdio_pins>;
56 diff --git a/arch/arm64/boot/dts/marvell/armada-37xx.dtsi b/arch/arm64/boot/dts/marvell/armada-37xx.dtsi
57 index 5db81a416cd6..55c9aa0aed0c 100644
58 --- a/arch/arm64/boot/dts/marvell/armada-37xx.dtsi
59 +++ b/arch/arm64/boot/dts/marvell/armada-37xx.dtsi
60 @@ -35,6 +35,11 @@ psci-area@4000000 {
61     reg = <0 0x4000000 0 0x200000>;
62     no-map;
63 };
64 +
65 +    tee@4400000 {
66 +        reg = <0 0x4400000 0 0x1000000>;
67 +        no-map;
68 +    };
69 };
70
71     cpus {
72 @@ -79,10 +84,11 @@ internal-regs@d0000000 {
73     compatible = "simple-bus";
74     /* 32M internal register @ 0xd000_0000 */
75     ranges = <0x0 0x0 0xd0000000 0x2000000>;
76 +    dma-coherent;
77
78     wdt: watchdog@8300 {
79         compatible = "marvell,armada-3700-wdt";

```

```

80 -             reg = <0x8300 0x40>;
81 +             reg = <0xd064 0x4>, <0x8300 0x40>;
82             marvell,system-controller = <&cpu_misc>;
83             clocks = <&xtalclk>;
84         };
85 @@ -423,6 +429,7 @@ crypto: crypto@90000 {
86             interrupt-names = "mem", "ring0", "ring1",
87                 "ring2", "ring3", "eip";
88             clocks = <&nb_periph_clk 15>;
89 +         cell-index = <0>;
90         };
91
92         rwtm: mailbox@b0000 {
93 @@ -432,7 +439,7 @@ rwtm: mailbox@b0000 {
94             #mbox-cells = <1>;
95         };
96
97 -         sdhci1: sdhci@d0000 {
98 +         sdhci0: sdhci@d0000 {
99             compatible = "marvell,armada-3700-sdhci",
100                 "marvell,sdhci-xenon";
101             reg = <0xd0000 0x300>,
102 @@ -443,7 +450,7 @@ sdhci1: sdhci@d0000 {
103             status = "disabled";
104         };
105
106 -         sdhci0: sdhci@d8000 {
107 +         sdhci1: sdhci@d8000 {
108             compatible = "marvell,armada-3700-sdhci",
109                 "marvell,sdhci-xenon";
110             reg = <0xd8000 0x300>,
111 @@ -511,5 +518,11 @@ armada-3700-rwtm {
112             mboxcs = <&rwtm 0>;
113             status = "okay";
114         };
115 +
116 +         optee {
117 +             compatible = "linaro,optee-tz";
118 +             method = "smc";
119 +             status = "okay";
120 +         };
121     };
122 };

```
