

# Conformité de poste multiplateforme

Hugo FORRAZ

---

Master Informatique  
Master mention Informatique



DÉPARTEMENT D'INFORMATIQUE  
Faculté des Sciences et Technologies

février, 2025



*Ce mémoire satisfait partiellement les pré-requis du module de Mémoire de Master, pour la 2<sup>e</sup> année du Master mention Informatique.*

**Candidat :** Hugo FORRAZ, N° 41909290, hugo.forraz.etu@univ-lille.fr

**Encadrant(e) :** Charles PAPERMAN, charles.paperman@univ-lille.fr

**Entreprise :** Stormshield

**Tuteur :** Damien DEVILLE, damien.deville@stormshield.eu



DÉPARTEMENT D'INFORMATIQUE  
Faculté des Sciences et Technologies  
Campus Cité Scientifique, Bât. M3 extension, 59655 Villeneuve-d'Ascq

février, 2025



# Résumé

Dans le cadre d'un projet de recherche industrielle, **Stormshield** vise à mettre en place un agent *Zero Trust Network Access* (ZTNA) développé en Rust servant à sécuriser les communications dans des environnements sensibles, projet nommé **OverSEC**. Afin de respecter la propriété "Zero Trust" de cet agent, il faut mettre en place une solution vérifiant la conformité du poste sur lequel l'agent sera utilisé.

C'est autour de cette problématique que s'est construite ma mission d'alternance. Afin d'y répondre, j'ai effectué une preuve de concept visant à proposer une solution intégrant **Osquery** à **OverSEC**. Cette preuve de concept a par la suite évolué vers une solution fonctionnant aussi bien seule – que ça soit passivement ou proactivement – qu'en tant que module utilisé par d'autres applications. De part la nature de **OverSEC** – qui était la source de la preuve de concept – la solution proposée est elle aussi multi-plateforme. Afin d'étendre la portée de l'agent de conformité créé, une étude exhaustive a aussi été menée sur les appareils mobiles, cette étude met en avant comment fonctionne le principal risque de conformité que ces plateformes rencontrent, *avoir des droits d'administrateur* et comment le détecter.

**Mots-clés** : ZTNA, Conformité de poste, Multi-plateforme, Rust, Osquery



# Abstract

Within the scope of an industrial research project, **Stormshield** aims to create a *Zero Trust Network Access* (ZTNA) agent built in Rust. This agent's – called **Oversec** – goal is to secure communications in sensitive environments. To make this agent meet the "Zero Trust" property, we have to build a hostchecking process, or as it is presented here, a *fully featured hostchecker*.

Throughout my internship, my mission was to build an agent that answers this need. I realised a proof-of-concept that made **Osquery** fit into **OverSEC**. This proof-of-concept then evolved towards a *standalone* agent that can be queried by other applications. This standalone agent can be used either passively – to meet a certain level of security everywhere – or actively – to detect threats. Because **OverSEC** is a multi-platform agent, the hostchecker needs to meet this requirement too. An exhaustive study has also been carried out on "how to hostcheck a mobile device?". It outlined that the main threat comes from *rooted* or *jailbroken* devices; this thesis tries to explain how it works and some ways to detect it.

**Keywords:** ZTNA, Hostchecking, Multi-platform, Rust, Osquery





# Indice

Table des figures	vii
Listings	ix
Glossaire	xi
Liste des Acronymes	xiii
<b>1 Stormshield, MESH, ZTNA ; une mise en contexte</b>	<b>1</b>
1.1 Présentation de Stormshield . . . . .	1
1.2 MESH, qu'est ce que ça implique? . . . . .	2
1.3 Objectifs . . . . .	2
<b>2 Élucidations sur la conformité de poste</b>	<b>3</b>
2.1 Une nécessité pour <b>OverSEC</b> . . . . .	3
2.2 Qu'est ce que la conformité de poste? . . . . .	5
2.3 Mise en place . . . . .	7
2.3.1 Utilisé indépendamment . . . . .	7
2.3.2 En tant que module . . . . .	9
2.3.3 Afin de détecter des irrégularités proactivement . . . . .	10
<b>3 Une approche multi-plateforme</b>	<b>11</b>
3.1 Environnement technique . . . . .	11
3.2 Cœur du projet . . . . .	12
3.2.1 Fonctionnement de l'agent . . . . .	13
3.2.2 Unix et Windows . . . . .	15
Quand l'asynchrone doit être synchrone . . . . .	15
Faire le lien entre l'agent et l'utilisateur . . . . .	17
3.3 La principale menace pour mobile : être administrateur . . . . .	18
<b>4 Conclusion</b>	<b>21</b>
<b>A Annexe</b>	<b>23</b>
A.1 Mise en lien avec Oversec . . . . .	23
A.2 Utilisation plus offensive . . . . .	25

A.3	Vue haut niveau du fonctionnement de l'agent . . . . .	25
-----	--	----

# Table des figures

2.1	Exemple de plusieurs topologies . . . . .	4
2.2	Fonctionnement de l'agent indépendant. La partie du haut montre les deux cas possibles, là où la partie inférieure montre la version qui sera agréementée au fil des schémas du mémoire . . . . .	8
2.3	Remontée d'un parc complet vers une administration centralisée . . .	8
3.1	Schéma présentant les couches de compatibilité mises en place par le framework Thrift (Source). . . . .	12
3.2	Bootchain des appareils iOS . . . . .	20
A.1	Description de la mise en lien de Oversec et de l'agent de conformité avec le principe de tags . . . . .	24
A.2	Parcours lors d'une chasse aux menaces . . . . .	25
A.3	Schéma montrant le fonctionnement haut niveau de l'agent de conformité. . . . .	26



# Listings

3.1	Requête exemple . . . . .	13
3.2	Requête du listing 3.1 abstraite . . . . .	13
3.3	Signatures des fonctions permettant de passer du code sur un autre fil d'exécution . . . . .	16



# Glossaire

## **asynchrone**

Code dont on ne récupèrera pas le résultat au moment où il a été appelé mais plus tard

## **bootchain**

En français on peut parler de "Chaîne de démarrage". Il s'agit des différentes étapes par lesquelles un appareil passe lors du démarrage.

## **cryptographie**

Ensemble de mécanismes de sécurité fournissant les propriétés de chiffrement, intégrité et authentification.

## **EDR**

De l'anglais "Endpoint Detection and Response". Logiciel fournissant une protection active contre les menaces détectées.

## **hardening**

en français durcissement, consiste à réduire la possible surface d'attaque disponible en appliquant des défenses contre les failles de sécurité possibles sur un logiciel.

## **REST**

**RE**presentational **S**tate **T**ransfer, façon de construire des API pour des applications client-serveur

## **runtime**

En français "Exécuteur asynchrone global", dans le contexte de code asynchrone, on a besoin d'un morceau de code capable de choisir quelle portion de code exécuter et de quelle manière

## **SEPOS**

Micro-noyau fonctionnant avec le *Secure Enclave Processor* (SEP).

## **SSA**

"System Software Authorization", logiciel combinant les clés de chiffrement gravées dans le matériel avec un service en ligne qui vérifie que l'on n'a pas modifié le logiciel présent sur le téléphone.

## **Tokio**

Principale librairie utilisée pour écrire du code asynchrone en Rust





# Liste des Acronymes

<b>FV</b>	Fréquence de Vérification
<b>AP</b>	<i>Application Processor</i>
<b>API</b>	<i>Interfaces de Programmation Applicative</i>
<b>DSL</b>	<i>Domain Specific Language</i> (en français Langage Dédié)
<b>NP</b>	<i>Named Pipe</i>
<b>P2P</b>	<i>Pair-à-pair</i>
<b>ROM</b>	Read Only Memory
<b>SDS</b>	<i>Stormshield Data Security</i>
<b>SEP</b>	<i>Secure Enclave Processor</i>
<b>SES</b>	<i>Stormshield Endpoint Security</i>
<b>SNS</b>	<i>Stormshield Network Security</i>
<b>UDS</b>	<i>Socket du domaine UNIX</i>
<b>VPN</b>	<i>Réseau Privé Virtuel</i>
<b>ZTNA</b>	<i>Zero Trust Network Access</i>



## Chapitre 1

# Stormshield, MESH, ZTNA ; une mise en contexte

### 1.1 Présentation de Stormshield

*Stormshield* est une entreprise filiale d'*Airbus Defense & Space Cyber Programmes* depuis son rachat en 2013. Elle est issue de la fusion de deux entreprises, *Arkoon* et *Netasq*, et est spécialisée dans la cybersécurité. Stormshield est d'ailleurs le premier éditeur français *pure-player* dans ce domaine. Elle possède trois lignes de produits :

- *Stormshield Network Security* (SNS), des pare-feux au service de la protection du réseau d'entreprise.
- *Stormshield Endpoint Security* (SES), la protection des postes et des serveurs *Windows* en entreprise.
- *Stormshield Data Security* (SDS), le chiffrement de la donnée pour tous.

Elle est répartie sur trois sites, Issy-les-Moulineaux, Villeneuve d'Ascq et Lyon. *Stormshield* comprend environ 433 collaborateurs.

Elle fait partie des plus grandes sociétés européennes dans le domaine de la cyber-sécurité, notamment grâce aux pare-feux SNS. L'entreprise a réalisé un chiffre d'affaires de 69 millions d'euros en 2023. Ayant une taille inférieure à d'autres concurrents américains ou israéliens, le choix a été fait de privilégier les entreprises sensibles à la sécurité telles que les entreprises publiques, gouvernementales ou l'industrie.

## 1.2 MESH, qu'est ce que ça implique ?

Mon stage de M1 ainsi que l'alternance autour de laquelle ce mémoire se construit ont été réalisés au sein du département **R&D** de l'entreprise dans l'équipe **MESH**. Cette dernière est composée de 21 personnes et réalise un projet de recherche industrielle visant à produire un agent *Zero Trust Network Access* (ZTNA) multiplateforme (Windows, Linux, MacOS, Android, iOS) en Rust. Ce langage est utilisé afin de complètement éliminer une classe de bugs – et donc de potentielles failles de sécurité – liée à la mémoire.

Le terme *agent* décrit un logiciel qui va agir de façon autonome, sans que l'on ait besoin de l'appeler nous même, en tant qu'utilisateur, de manière explicite – *i.e.* une fois l'agent démarré, il agira au moment opportun, *e.g.* un "Ok Google, [...]" pour discuter avec un Google Assistant ou à l'envoi de paquets réseaux dans un *Réseau Privé Virtuel* (VPN) (*cf.* la sec. 2.1). Ce fonctionnement permet à l'utilisateur de déléguer des tâches à son ordinateur automatiquement et ce sans nécessiter de compétences techniques élevées. Ce terme sera utilisé dans la suite du mémoire pour transmettre l'idée d'un logiciel qui agit sans interaction directe avec l'utilisateur.

Quant au concept de "ZTNA", celui-ci se découpe en deux parties. La première – "Zero Trust" – est le fait de ne faire confiance à aucune partie, que ce soit le logiciel, le système d'exploitation, son utilisateur ou une tierce personne, et donc de monter le niveau de sécurité attendu pour accéder à des données. Lorsque l'on met en place une solution de ce type, on s'attend à vérifier la *conformité* d'un hôte à chaque instant et non pas à l'authentification seulement, on part du principe qu'une menace ou plus simplement un attaquant ne sera pas forcément présent à l'authentification mais apparaîtra après la phase d'authentification. Ce concept nous amène donc à nous demander comment l'on peut vérifier qu'un poste sera conforme ou fiable tout au long de son utilisation. La deuxième partie – "Network Access" – quant à elle vient proposer ce concept dans le cadre de logiciels utilisés pour des communications réseaux. Ici, l'agent sur lequel on va se baser est celui développé par l'équipe **MESH**, le dénommé **OverSEC** qui sera explicité dans la section 2.1.

## 1.3 Objectifs

Lors de mon alternance, j'ai donc pu m'intéresser à comment l'on peut donner forme au concept de **Zero Trust** dans le cadre d'un agent multi-plateforme.

Pour ce faire, on va dans un premier temps étudier les enjeux qu'apportent **OverSEC**, enjeux qui nous permettront d'explorer les différents besoins auxquels doit répondre un agent de conformité. Dans un second temps nous étudierons le fonctionnement d'un tel agent et comment ce fonctionnement diffère selon les plateformes impliquées par le terme "multi-plateforme".

## Chapitre 2

# Élucidations sur la conformité de poste

### 2.1 Une nécessité pour OverSEC

Pour accéder à des ressources qui ne sont pas sur notre poste, on va communiquer avec d'autres machines. L'étape la plus directe pour cela est de communiquer avec des machines présentes sur le même réseau physique. Lorsque l'on va vouloir récupérer des ressources qui ne sont pas disponibles sur ce réseau physique, on va vouloir y accéder depuis un réseau distant. Le moyen le plus simple et sécurisé de faire ce lien est de passer par un réseau logique et sécurisé, c'est à dire un réseau qui existe seulement grâce à un logiciel pour mettre en lien des machines distantes présentes sur des réseaux physiques différents. Une méthode employée par les administrateurs systèmes – afin d'éviter de trop exposer le réseau – pour régler ce problème est la mise en place d'un *Réseau Privé Virtuel* (VPN). Pour mettre en place ces réseaux logiques, un VPN va donc faire transiter les information par une interface virtuelle, interface qui pourra – et même ira dans la plus grande majorité des cas – mettre en place une communication sécurisée par cryptographie<sup>1</sup>. Sauf que pour des VPN, il sera assez complexe de déployer et de gérer au quotidien des configurations mettant en place des réseaux logiques différents pour chaque usage métier afin de les isoler.

---

1. Ensemble de mécanismes de sécurité fournissant les propriétés de chiffrement, intégrité et authentification.

Avec pour objectif de résoudre ce problème, **OverSEC** est développé. Ce sujet d’alternance dérive d’un besoin existant dans ce projet.

Comme évoqué plus tôt, **OverSEC** est un agent ZTNA. Cet agent est destiné à sécuriser des communications entre deux postes ou plus en *Pair-à-pair* (P2P). Les avantages du P2P sont sur la modularité ainsi que la résilience qu’offrent ce modèle de communication. Le principe de base de l’agent est assez similaire à celui d’un VPN, mais ici le principe serait d’égrainer plus finement qu’avec un simple VPN en séparant les utilisateurs par réseau métier – ou topologies – plutôt qu’en faisant plusieurs configurations VPN pour se connecter aux différents points plus ou moins critiques.

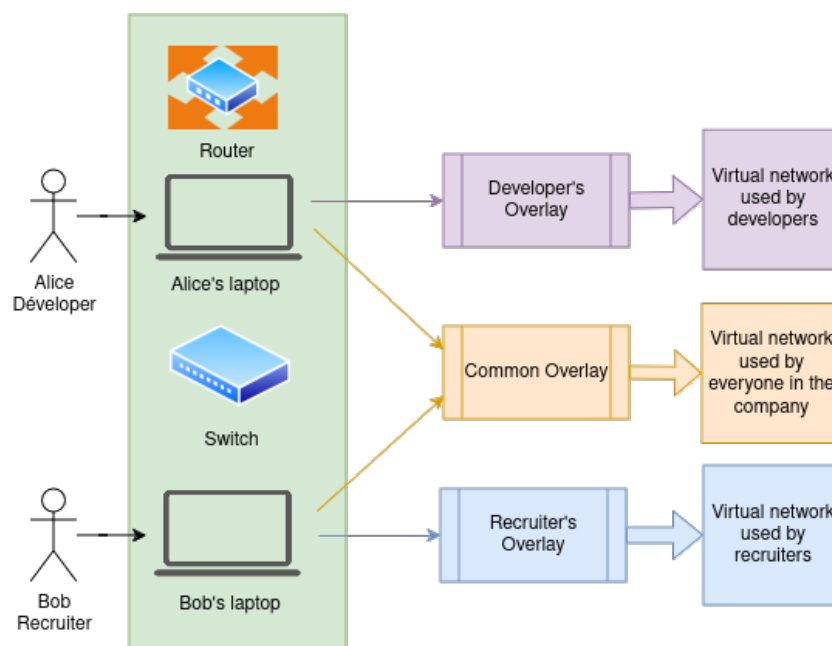


FIGURE 2.1 – Exemple de plusieurs topologies

Par exemple, sur la figure 2.1, on peut voir Alice et Bob qui disposent d’une “*configuration commune*” ainsi que d’une topologie pour accéder à des ressources selon le métier de l’individu, respectivement “développeur” et “recruteur”.

Pour continuer sur l’analogie du VPN, avant de démarrer l’agent, on est donc sur un réseau physique avec des postes client, des serveurs et des équipements réseaux ; tous les appareils présents sur ce réseau vont donc pouvoir communiquer de manière non sécurisée. Afin de sécuriser les communications sur ce réseau ou avec d’autres réseaux, on va créer un réseau logique qui sera lui composé d’agents. Ces communications sur le réseau logique passant par les interfaces virtuelles décrites plus haut, celles-ci seront sécurisées à l’aide de cryptographie. Un autre moyen de protection disponible grâce à ces réseaux logiques est aussi de mettre en place une politique de filtrage plus stricte où l’on pourrait seulement autoriser certains types

de paquets ou messages (liste blanche) ou au contraire passer par des interdictions particulières (liste noire).

Une autre partie importante de **OverSEC** est le fait que ce projet vise aussi bien les plateformes PC (Windows, Linux, MacOS) que les plateformes mobiles (Android, iOS). Le fait de viser plusieurs plateformes permet donc de proposer le produit à une scène plus large mais apporte aussi son lot de difficultés du côté développement. Parmi celles-ci, on peut assez facilement considérer la différences des *Interfaces de Programmation Applicative* (API) système, *e.g.* faire traverser un paquet réseau sur une interface Windows ou MacOS passe à travers des processus différents. Un autre exemple peut être le fait de compiler les versions mobiles des applications, le faire pour Android ne nécessite pas de machine particulières, seulement des bonnes chaînes de compilation ; là où compiler une application pour iOS nécessite un Mac ainsi que d'un compte développeur Apple.

Ce projet restant dans la ligne de conduite des agents dits "*Zero Trust*", il faut donc pouvoir vérifier à chaque instant du cycle de vie d'un agent que le système d'exploitation sur lequel il s'exécute est dans l'état attendu en terme de sécurité. La suite de ce mémoire va donc explorer comment s'assurer de la conformité d'un poste, d'abord d'un point de vue administrateur, puis d'un point de vue logiciel, et ce, quelle que soit la plateforme.

## 2.2 Qu'est ce que la conformité de poste ?

Le thème de la conformité de poste a pu être abordé plus tôt mais n'a pas été défini. Ici, la "conformité de poste" s'apparente au concept de "hostchecking" dans la littérature anglophone, *i.e.* un processus de sécurité permettant de s'assurer que le poste utilisé respecte certaines règles et directives, que ça soit d'experts en sécurité (*e.g.* Stormshield) ou d'administrateurs système. Des exemples simples de règles de conformité pourrait être les phrases suivantes : "L'anti-virus est-il à jour ?" ou "Le poste utilise-t-il effectivement une *smartcard*<sup>2</sup> ?". Dans la suite de ce mémoire, on parlera donc de l'**agent de conformité** qui a été développé.

Il faut aussi noter qu'un agent de conformité seul pourrait faire face à un problème majeur : si le poste est compromis, un attaquant peut dissimuler que celui ci est compromis. Pour éviter ce problème, on peut coupler cet agent à des solutions telles que des EDR<sup>3</sup>, du hardening<sup>4</sup>, un anti-virus ou la mise en place de bonnes pratiques de configuration d'un poste. Ces solutions annexes n'étant pas le sujet de ce mémoire, elles n'y seront pas abordées et on supposera que l'environnement

---

2. carte à puce physique permettant d'authentifier un utilisateur sur son poste

3. De l'anglais "Endpoint Detection and Response". Logiciel fournissant une protection active contre les menaces détectées.

4. en français durcissement, consiste à réduire la possible surface d'attaque disponible en appliquant des défenses contre les failles de sécurité possibles sur un logiciel.

sur lequel agit l’agent de conformité est un environnement fiable – *i.e.* on peut être confiant sur le fait que l’agent renvoie des résultats qui n’ont pas été falsifiés par un attaquant.




Lors du balisage des nécessités du projet, trois niveaux d’utilisation ont été définis :

- **Niveau 1** : un agent indépendant chargeant un jeu de règles et pouvant régulièrement faire des rapports de l’état de ces règles, que ça soit à l’aide
  - (a) d’une API,
  - (b) d’une interface graphique ou
  - (c) d’un rapport au format textuel

dans ce niveau d’utilisation. On ne s’attend pas à ce que plusieurs jeux de règles soient utilisables en même temps. On s’attend à pouvoir distribuer le jeu de règles à autant de consommateurs que voulu.

- **Niveau 2** : On veut pouvoir utiliser plusieurs jeux de règles différents pour un consommateur donné. De plus ces jeux de règles doivent pouvoir être mis-à-jour dynamiquement. Un exemple de cet usage serait de permettre à un administrateur système de mettre en place un jeu de règles plus flexible pour différencier des configurations de **OverSEC** et de pouvoir en plus changer le jeu de règles au jour le jour selon des besoins pouvant émerger.
- **Niveau 3** : Une fonctionnalité de *Threat Hunting* ou en français, ”chasse aux menaces” (terme que l’on va garder dans la suite). Cette fonctionnalité permet d’exécuter dynamiquement une série de requêtes sur tout ou partie d’un parc, par exemple dans le cas où une nouvelle attaque vient d’être détectée, s’assurer de quels postes peuvent être compromis par celle-ci ou de diagnostiquer un problème sur le poste d’un utilisateur.

Lors de la création d’un rapport de conformité, l’agent va pour chaque règle produire un résultat parmi les trois niveaux de conformités suivants :

-  : Conforme
-  : Risque(s) pour la conformité, non bloquant (ex : Mise à jour de sécurité Windows sortie il y a 2 jours et non critique). L’intérêt de ce niveau de conformité est de laisser un sursis à l’utilisateur. Ce sursis trouve son utilité dans des cas où remédier au problème n’est pas important ou que l’on a besoin de ressources internes afin de corriger ce problème.
-  : Non conforme, bloquant (ex : Mise à jour de l’agent **OverSEC** pour accéder à des ressources internes)

Dans la suite de cette section, nous allons donc étudier quel est le public visé par ce genre de logiciel ainsi que les façons de l’utiliser, que ça soit indépendamment,



sous forme d'agent connecté à une console d'administration centralisée, ou encore en tant que brique d'un projet plus imposant.

L'agent de conformité est pensé pour être utilisé par :

- (1) l'utilisateur qui aura l'agent sur son poste, celui-ci pouvant mettre en évidence les points qui ne respectent pas une politique de conformité définie par son administrateur ainsi que la manière de remédier à ces défauts.
- (2) un administrateur voulant vérifier :
  - (a) l'état de son parc ;
  - (b) forcer les postes des utilisateurs à respecter certaines caractéristiques pour accéder à des ressources données.

## 2.3 Mise en place

### 2.3.1 Utilisé indépendamment

L'agent de conformité dispose d'un mode *standalone* ou en français *indépendant*. Lorsqu'on l'utilise de cette façon, l'objectif est de s'assurer qu'un poste respecte une politique de conformité. Pour cela, on commence par fournir à l'agent la liste de règles de conformité qu'il devra vérifier périodiquement. Celui-ci va par la suite les vérifier et fournir un rapport de conformité. Quand toutes les règles de ce rapport renvoient un résultat positif (✓), aucun problème n'est à signaler.

Si au contraire, une règle ou plus n'est pas validée (⚠ ou ✗), ces règles apparaîtront dans le rapport comme étant "défaillantes" et les méthodes de remédiation liées à ces règles seront fournies à l'utilisateur afin que ce rapport soit conforme. De plus, tant qu'une règle n'est pas conforme, celle-ci peut se voir affectée une fréquence de re-vérification plus élevée selon la nécessité ainsi que la facilité de la remédiation (définition faite par l'administrateur ou créateur de la règle).

Cet agent indépendant peut être intégré dans la vue haut niveau décrite avec la figure 2.3, on appelle cette fonctionnalité utilisant une administration centralisée le mode *console*.

Cette fonctionnalité centralisée permet à un administrateur de déployer le même jeu de règles à tout un parc d'utilisateurs et ainsi de connaître l'état de son parc. Dans ce jeu de règles créé par l'administrateur coexistent des règles proposées par *Stormshield*, ainsi que des règles mises en place par l'administrateur afin de personnaliser sa politique de conformité. Les rapports de conformité de chaque poste sont par la suite tous remontés à l'administrateur du parc afin de détecter quels postes ne sont pas conformes.

Mais ce fonctionnement peut manquer de flexibilité étant donné que l'on donne une seule et unique base de règles à tous les utilisateurs d'un parc, certains cas

### Stand-alone

Simple and straight-forward behavior. This module is self-sufficient in term of features.

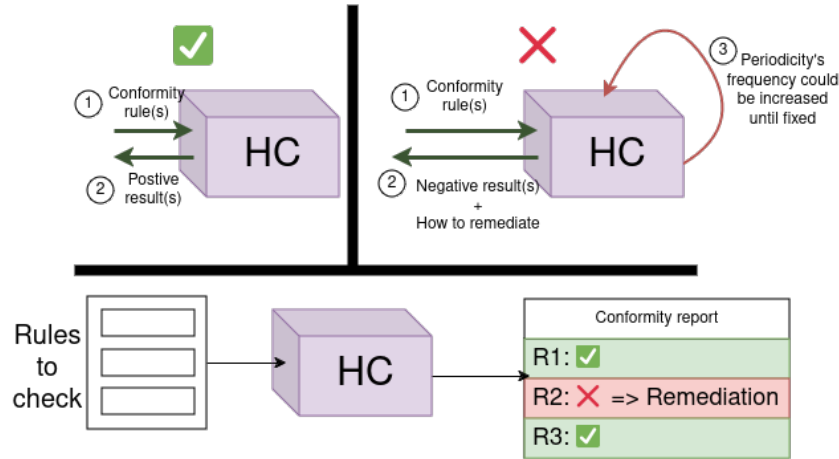


FIGURE 2.2 – Fonctionnement de l'agent indépendant. La partie du haut montre les deux cas possibles, là où la partie inférieure montre la version qui sera agrémentée au fil des schémas du mémoire

concrets pourraient nécessiter plus de flexibilité. On pourrait décrire un usage métier dans une entreprise de sécurité informatique où un ordinateur est utilisé pour développer une solution ainsi que de communiquer avec ses collègues et un deuxième ordinateur serait utilisé pour accéder à Internet. Les deux postes ne suivraient pas forcément la même politique de conformité ce qui ne convient pas avec le fonctionnement assez rigide introduit ci-dessus.

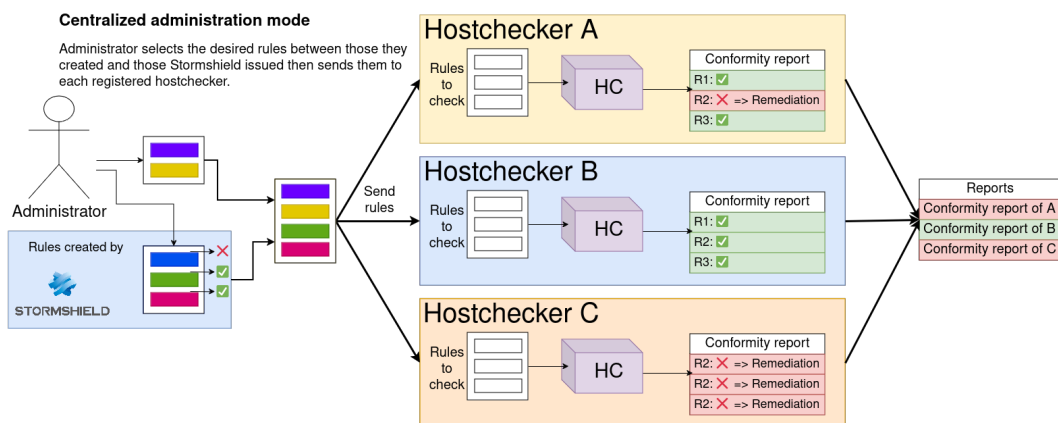


FIGURE 2.3 – Remontée d'un parc complet vers une administration centralisée

### 2.3.2 En tant que module

Afin de répondre à la problématique énoncée plus haut, on peut avancer le principe de *tag* (ou d'**étiquette** en français) qui rend l'usage d'autres modules à politiques de conformité personnalisées – tels que **OverSEC** – plus simple. À l'aide de ces étiquettes, on peut donc plus facilement regrouper un ensemble de règles pour identifier une politique de conformité particulière liée à un usage métier dans l'entreprise (stagiaire, développeur, administrateur, gestionnaire de paie, etc...), un accès particulier à un VPN, ou encore selon le système d'exploitation utilisé.

Afin de donner un exemple de ce fonctionnement, prenons le cas où l'agent de conformité est utilisé en lien avec **OverSEC**. On pourrait par exemple distinguer deux types d'étiquettes : "Global" et "RD\_Linux". On pourrait lier à ces étiquettes les règles suivantes :

- *Global* prendrait en compte le jeu de règles *Stormshield*.
- *RD\_Linux* forcerait un utilisateur à utiliser un système sous une certaine version de Linux et à respecter d'autres prérogatives demandées par l'administrateur.

Du côté de **OverSEC**, on pourrait imaginer les topologies suivantes :

- **Social** : Réseau social d'entreprise, ne nécessite pas de configuration particulière.
- **Corp** : Ressources internes à l'entreprise nécessitant un niveau de sécurité inclus avec l'étiquette *Global*.
- **R&D** : Ressources accessibles seulement aux membres de la R&D de l'entreprise qui nécessiterait les étiquettes *RD\_Linux* et *Global*.

La figure A.1 représente l'utilisation de ces étiquettes avec les différentes topologies décrites. On peut y observer comment l'agent de conformité se comporterait avec un poste qui respecte l'étiquette *Global* mais pas la *RD\_Linux*. Le poste est donc autorisé à accéder aux topologies **Social** et **Corp** mais ne pourra pas accéder à celle de **R&D**.

A contrario, si l'on enlève ce module, **OverSEC** n'est plus capable de déduire si le poste sur lequel il tourne respecte les contraintes demandées et va fonctionner seulement à l'aide de caractéristiques telles que des configurations nominatives, ou de mots de passe. De plus, manquer ce module lui ferait perdre la mention de ZTNA étant donné que celle-ci implique – comme énoncé plus tôt (sec. 2.1) – de pouvoir vérifier à chaque instant du cycle de vie de l'agent que celui-ci soit conforme vis-à-vis de la politique.

### 2.3.3 Afin de détecter des irrégularités proactivement

Cette dernière section vient présenter une fonctionnalité permettant à un administrateur sécurité de s'assurer sporadiquement de l'état de son parc vis à vis de règles  $X$  ou  $Y$  déclarées sur l'instant. Ces règles sont volatiles, au sens où une fois que l'une d'elles a été envoyée aux différents agents, la règle ne sera pas re-vérifiée à un instant  $T + 1$ . La fonctionnalité de chasse aux menaces amène l'idée que l'on va pouvoir proactivement vérifier que tout ou partie du parc a été compromis par une attaque ou non.

Le fonctionnement est assez proche de celui décrit dans la section 2.3.1 avec la figure 2.3. Néanmoins, plutôt que de distribuer un ensemble de règles et de renvoyer périodiquement un rapport de conformité à l'utilisateur ainsi qu'à l'administrateur, on va ici envoyer une règle qui doit dans l'instant  $T$  être évaluée et le rapport sera remonté directement à l'administrateur. Ce fonctionnement est décrit dans la figure A.2.

## Chapitre 3

# Une approche multi-plateforme

L’agent de conformité et **OverSEC** sont prévus pour fonctionner sur les systèmes d’exploitation les plus communs, Windows, MacOS et Linux. Sauf que le fait de viser des plateformes avec des coeurs aussi différents avec un même logiciel implique de devoir mettre en place des abstractions pour les composants les plus bas niveau, par exemple, les interfaces de communication. Les plateformes mobiles étant une autre cible de ces projets, il faut comprendre les enjeux que ces plateformes ajoutent à une telle application.

Dans ce chapitre, nous allons étudier comment on a pu mettre en place cette solution visant à vérifier la conformité d’un poste, le fonctionnement de son moteur ainsi que les difficultés qui ont pu être rencontrées lors des différentes itérations du développement permettant d’étendre les cibles touchées par l’agent.

### 3.1 Environnement technique

Comme énoncé plus tôt, ce projet a été écrit complètement en *Rust*, langage de programmation visant à éliminer toute la classe de bugs mémoires sans faire de compromis sur la vitesse d’exécution du programme qui sera écrit. Le code écrit étant compilé de manière à rivaliser avec du code C ou C++, ce langage est parfait pour l’écriture de modules kernel, d’agents réseau – comme **OverSEC** – ou encore, dans notre cas, de services qui vont être utiles tout au long de l’utilisation de son poste.

Afin de ne pas partir de zéro, l'agent de conformité vient se baser sur un projet open-source du nom de Osquery. Le principe de Osquery est de pouvoir vérifier les différentes données liées à un poste en utilisant des requêtes écrites en SQL, par exemple connaître l'état de l'anti-virus Windows, connaître la distribution Linux, ou encore savoir quelles interfaces réseau sont actuellement entrain de fonctionner. Osquery étant un logiciel multi-plateforme, celui-ci utilise les protocoles mis en place par Apache Thrift afin d'offrir la plus grande compatibilité entre les différents langages de programmation et systèmes d'exploitation pouvant être utilisés (voir fig. 3.1).

Apache Thrift Layered Architecture

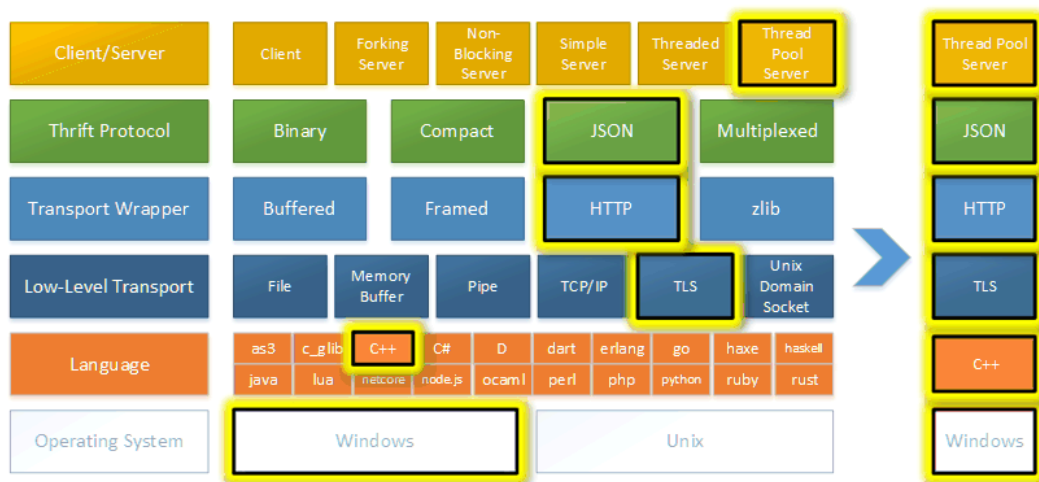


FIGURE 3.1 – Schéma présentant les couches de compatibilité mises en place par le framework Thrift (Source).

L'intérêt de Thrift est de mettre en place des API où les différents services seront décrits à l'aide d'un *Domain Specific Language* (en français Langage Dédié) (DSL). Ce dernier permet de faire en sorte que le client et le serveur utilisent les mêmes données sans que le développeur ait besoin de deviner ce qui va arriver de l'autre côté. Thrift mettant en place une librairie Rust, on peut donc créer un logiciel qui viendra communiquer avec Osquery afin de s'assurer de l'état du poste sur lequel on opère.

## 3.2 Cœur du projet

Ici, l'objectif va être de comprendre comment l'agent fonctionne ainsi que les défis qui ont été rencontrés lors des développements pour les différentes plateformes.

### 3.2.1 Fonctionnement de l'agent

L'agent de conformité est une application qui va servir de pont entre Osquery et tout autre logiciel tierce (e.g. : un client VPN ou **OverSEC**) en proposant une abstraction à l'aide d'un DSL basé sur le format KDL afin de décrire les règles introduites dans la section 2.2. Les règles sérialisées de cette façon fonctionnent comme des textes à trous où chaque trou dispose d'indices afin de remplir la requête. Ainsi, la requête proposée dans le listing 3.1 sera abstraite en la requête présente dans le listing 3.2. Cet usage des règles sous forme de texte à trous permet à un administrateur de filtrer plus ou moins finement certaines propriétés selon, par exemple, les services visés.

---

```

1 SELECT interface, mac
2 FROM interface_details
3 WHERE (
4     mtu > 1000 AND
5     mtu < 1800
6 ) AND
7     interface IN ("eth%", "wlan0", "docker%");

```

---

Listing 3.1 – Requête exemple

---

```

1 SELECT interface, mac
2 FROM interface_details
3 WHERE (
4     mtu > ... /*numeral expected*/ AND
5     mtu < ... /*numeral expected*/
6 ) AND
7     interface IN ... /*multiple strings expected*/;

```

---

Listing 3.2 – Requête du listing 3.1 abstraite

Les règles sont différenciées par l'agent à l'aide d'un nom, elles peuvent avoir une Fréquence de Vérification (*FV*) différente (e.g : chaque seconde, minute, heure ou encore toutes les 5, 10, 20 ou 45 secondes). Quand l'administrateur a mis en place ses règles paramétrables et a rempli les différents trous pour son lot de requêtes, il va pouvoir se connecter aux différents agents de conformité grâce à une API REST<sup>1</sup> afin de leur faire remonter les différentes règles devant être vérifiées périodiquement.

Une fois ces requêtes reçues sous la forme de règles, celles-ci vont suivre le chemin décrit dans la figure A.3 à travers trois circuits.

---

1. **RE**presentational **S**tate **T**ransfer, façon de construire des API pour des applications client-serveur

Le premier circuit (A, en violet) est celui de communication extérieure. Il permet de simplement recevoir des règles mais aussi de vérifier leur état, de recevoir un rapport de conformité. Ce fonctionnement est celui qui est privilégié en temps normal mais il existe aussi la possibilité de recevoir une notification si l'état d'une règle actuellement observée change. Le but concret de ce circuit est de répondre à l'utilisateur dès que celui-ci fait une requête.

Afin de transmettre la réponse voulue au plus vite, vient le deuxième circuit (B, en vert) qui sert de liant entre les circuits A et C. Ce circuit renvoie les résultats présents dans le cache au circuit A quand ceux-ci sont demandés, et remplit périodiquement son cache à l'aide du circuit C.

Enfin, le dernier circuit (C, en bleu) est celui décrivant les interactions entre l'agent et Osquery. Lors du passage dans ce circuit, on envoie une requête SQL à Osquery qui va s'occuper de construire la réponse en se basant sur l'état courant de notre système. Une fois ce résultat récupéré par l'agent, celui-ci sera inséré dans le cache utilisé par le circuit B. Par ailleurs, il existe un circuit interne dans Osquery (circuit jaune) permettant de mettre en place des règles qui seront vérifiées périodiquement à tout moment du cycle de vie de l'application. Le point important avec ce circuit interne est qu'il faut que ces règles soient décidées avant le lancement de Osquery, si l'on veut que des règles soient modifiées dynamiquement, on doit donc passer par le circuit mis en place par l'agent.

Une brique importante de l'agent est son ordonnanceur présent dans le circuit B (dans la brique "engine"). Afin de comprendre son intérêt et son fonctionnement, nous allons décrire le flot du moteur de vérifications en ajoutant progressivement les morceaux qui permettent d'arriver à l'ordonnanceur actuellement utilisé. Quand cette brique n'existe pas, il existe deux options :

- Ne pas vérifier les règles périodiquement, une demande de l'utilisateur est donc une demande à Osquery.
- Regarder à chaque instant du cours du programme quelle règle doit être vérifiée.

Le premier fonctionnement est assez simple et direct mais, comme dit lors de son énoncé, ne convient pas à une vérification périodique. Quant au second, celui-ci peut assez vite devenir coûteux en terme de performances quand le jeu de règles devient assez conséquent.

Afin d'optimiser cela, une première étape est à la réception d'une règle de créer un processus léger qui sera dédié à vérifier toutes les  $N$  secondes l'état de sa règle. Sauf que ce fonctionnement peut lui aussi assez vite coûter cher aussi bien en terme de mémoire – on aura  $N$  processus légers contenant chacun sa règle, son réveil<sup>2</sup>, ainsi que les données nécessaires au fonctionnement du processus – qu'en terme de

---

<sup>2</sup> mécanisme permettant à un processus en attente de reprendre son exécution à un instant  $T + 1$



performances, les changements de contextes – induits par le nombre de processus – pouvant eux aussi coûter cher quand ils sont demandés trop fréquemment. La meilleure option pour éviter d’avoir ces coûts qui augmentent trop est de créer un processus léger par *FV* désirée plutôt que par règle vérifiée. Étant donné que plusieurs règles partagent la même *FV*, on peut de cette manière borner :

- (1) le nombre de processus légers,
- (2) le nombre de mécanismes de *réveils*,
- (3) la fréquence des changements de contexte,

par le nombre de *FV* différentes présentes dans le jeu de règles. Par la suite, l’ordonnanceur a seulement à envoyer au bon processus les règles à vérifier lors de leur réception.

### 3.2.2 Unix et Windows

Lors de la mise en place de ce programme, la première étape était de faire une première preuve de concept sous Linux pour étudier comment les différentes composantes fonctionnaient. Par la suite, il a fallu porter cette preuve de concept pour qu’elle fonctionne sous Windows.

Cette sous-section va explorer les principales différences entre ces plateformes, l’interface de communication entre l’agent et Osquery, et comment exécuter le programme en arrière plan.

#### Quand l’asynchrone doit être synchrone

Lors du démarrage de Osquery, celui ci va à l’aide de Thrift créer sous Unix une *Socket du domaine UNIX* (UDS), une interface de communication inter-programme. Windows n’étant pas basé sous Unix comme le sont Linux ou MacOS, on va ici créer un *Named Pipe* (NP). La librairie Rust de Thrift étant assez jeune, toutes les fonctionnalités n’existent pas et parmi celles-ci, une assez importante dans le cadre de communications, le fait d’utiliser du code asynchrone<sup>3</sup>.

Dans le cas des UDS, ce n’est pas un problème, celles-ci existant dans la librairie standard du langage (ici) en synchrone. Sauf que lors du passage vers Windows, il faut donc adapter le code pour que celui-ci fonctionne avec les NP, qui en Rust n’existent qu’en version asynchrone, avec le support le plus complet dans la librairie Tokio<sup>4</sup>. Les interfaces utilisées par Thrift nécessitant forcément d’utiliser du code synchrone, il existe peu d’options :

- (1) Réécrire une partie de Thrift pour faire appel à du code aussi bien synchrone qu’asynchrone.

---

3. Code dont on ne récupérera pas le résultat au moment où il a été appelé mais plus tard

4. Principale librairie utilisée pour écrire du code asynchrone en Rust

- (2) Créer une librairie proposant des NP synchrones en Rust.
- (3) Faire un pont qui consommera la version asynchrone existant déjà vers une version synchrone.

La première option n’a pas été retenue étant donné que ça demandait beaucoup d’effort d’ajouter dans un projet monolithique tel que Thrift une fonctionnalité permettant d’utiliser du code asynchrone. Cette modification implique de reprendre une partie du compilateur Thrift afin de générer du code Rust, il faut par la suite que ce développement soit intégré dans la version fournie par Thrift du compilateur ou que nous maintenions nous même notre propre version de celui-ci.

La deuxième option n’a pas été retenue non plus étant donné qu’implémenter des NP de zéro demande une expertise dans ces interfaces assez forte et les implémentations qui existent font des milliers de lignes de code.

La troisième option par contre pouvait si elle fonctionnait directement ne pas coûter très cher à développer, ce qui permettait derrière de pouvoir avancer sur la suite du projet assez vite. Afin d’écrire et lire des données de façons synchrone, la librairie standard de Rust propose les traits `Read` et `Write`, là où Tokio propose `AsyncRead` et `AsyncWrite` afin d’exécuter ces mêmes opérations de façon asynchrone. Pour résoudre ce problème on va donc prendre les interfaces asynchrones déjà existantes et essayer de les utiliser de manière synchrone en exécutant le code asynchrone dans un processus léger dont on attendra le résultat. dans la littérature anglophone on parle ici de "Bridge async and sync code".

Pour réaliser cette approche, on ne peut pas se servir de `std::thread::spawn` qui permet de simplement paralléliser un code synchrone et il n’existe pas de fonction dans Tokio permettant d’appeler du code asynchrone depuis un contexte synchrone sans faire appel au runtime<sup>5</sup> créé par Tokio, ce *runtime* n’étant pas forcément adéquat à être remonté dans toutes les fonctions. On pourrait pour palier à ce problème créer un nouveau *runtime* gérant seulement le NP, sauf que disposer d’un *runtime* dans un autre n’est pas une solution très satisfaisante (ni autorisée par Tokio) étant donné qu’un *runtime* vient bloquer le fil d’exécution courant, ce qui bloquerait le *runtime* parent et donc potentiellement d’autres tâches. Le meilleur moyen est donc de faire ces opérations asynchrones dans un exécuteur annexe qui bloquera seulement le fil d’exécution qu’on lui aura accordé (quatrième fonction du listing 3.3).

```
1  /// Standard library's function at std::thread
2  pub fn spawn<F, T>(f: F) -> std::thread::JoinHandle<T>
3  where
4      F: FnOnce() -> T + Send + 'static, // This must be a function
```

5. En français "Exécuteur asynchrone global", dans le contexte de code asynchrone, on a besoin d’un morceau de code capable de choisir quelle portion de code exécuter et de quelle manière

```

5      T: Send + 'static, { /* ... */ }
6
7      /// Tokio's spawn function.
8      /// JoinHandle is a Future that can be awaited
9      pub fn spawn<F>(future: F) -> tokio::task::JoinHandle<F::Output>
10     where
11         F: Future + Send + 'static, // F must be an async block
12         F::Output: Send + 'static, { /* ... */ }
13
14     /// Tokio's block_on function. Needs a Runtime to be used
15     impl Runtime {
16         pub fn block_on<F>(&self, future: F) -> F::Output
17         where // F must be an async block
18             F: Future, { /* ... */ }
19     }
20
21     /// futures::executor's function, the one we chose to use.
22     pub fn block_on<F>(f: F) -> <F as Future>::Output
23     where // F must be an async block
24         F: Future, { /* ... */ }

```

Listing 3.3 – Signatures des fonctions permettant de passer du code sur un autre fil d'exécution

### Faire le lien entre l'agent et l'utilisateur

On peut aussi noter des différences majeures sur le démarrage de l'agent en arrière-plan. Sous Unix, on appelle ce type de processus un "démon" (ou en anglais *daemon*). La méthode la plus récente de créer un démon est de passer par `systemd`. Pour cela, on va utiliser un fichier `unit` qui sert à lister où trouver les différents dossiers utilisés par l'agent mais surtout, comment le démarrer. Par exemple, dans notre cas, on spécifie comment lancer Osquery et où trouver le binaire de notre programme. Lorsque `systemd` n'est pas présent sur le poste, une autre méthode consiste en les quatre étapes suivantes :

- (1) Lancer le programme
- (2) Démarrer un processus lourd depuis ce programme
- (3) Tuer le programme de départ en laissant le processus lourd créé plus tôt
- (4) Reprendre son ordinateur pour effectuer les tâches que l'on veut

A contrario, sous Windows on va paramétrer l'installateur de manière à ce qu'il sache comment mettre en place le service, pour cela on peut utiliser WixToolset

qui va permettre de créer un fichier MSI<sup>6</sup>. De plus, dans le code de l'agent on doit faire en sorte de démarrer le programme en tant que service. Ce processus passe par le fait de notifier le gestionnaire de service – géré par le système d'exploitation – que notre programme doit être enregistré en tant que service et de donner à ce dernier un moyen de contacter notre service. Ce lien entre le gestionnaire de service et notre programme permet au système d'exploitation d'envoyer des instructions au programme telles que se démarrer ou s'arrêter.

### 3.3 La principale menace pour mobile : être administrateur

Plus tôt il a été énoncé que OverSEC fonctionnerait sur les plateformes mobiles en plus des postes Windows ou Unix plus classique. Les plateformes mobiles majeures (Android et iOS) sont pensées de manière plus restrictives que des ordinateurs plus classiques, chaque application agit dans un bac-à-sable auquel elle est restreinte. La plupart des informations dont l'on a besoin sont d'ailleurs proposées sous forme de fonctions ou méthodes dans le code de base des frameworks Android et iOS.

Les accès sur un téléphone sont tellement restreints que l'on n'est pas administrateur de ceux ci et la majeure partie des risques que l'on peut trouver sont liés au mode développeur – permettant d'obtenir beaucoup d'informations sur les applications utilisées – mais aussi et surtout au fait de devenir administrateur du téléphone, concepts de *root* sous Android et de *jailbreak* sous iOS.

Devenir administrateur sur un mobile revient à ouvrir une boîte de Pandore, qui une fois ouverte supprime les restrictions logicielles présentes pour la sécurité de notre appareil. Étant donné que pour accéder à ces privilèges on a besoin de passer par une faille de sécurité présente sur l'appareil (quelle soit logicielle ou matérielle), on parle ici d'une **escalade de privilèges**.

Afin de comprendre comment l'on peut détecter qu'un mobile est administrateur, il est utile de comprendre comment on peut arriver à ce niveau. La première problématique autour de laquelle on peut développer est "Comment démarre un appareil mobile?". Android étant basé sur Linux (que l'on connaît plus communément) et de plus en plus de mobiles tendant à intégrer des processus de sécurisation existant dans la *bootchain*<sup>7</sup> de Apple, on va commencer par étudier la *bootchain* que ces derniers proposent. Historiquement, la première *bootchain* utilisée par iOS se basait sur les 5 niveaux suivants :

- Boot Read Only Memory (ROM) : Contient du code à lecture seule et le certificat d'authentification administrateur de Apple qui seront chargés par le

---

6. fichiers servant à installer des applications sous Windows

7. En français on peut parler de "Chaîne de démarrage". Il s'agit des différentes étapes par lesquelles un appareil passe lors du démarrage.

processeur. Dans la littérature anglophone, un mot clef qui est assez récurrent pour parler de cette brique est "Hardware Root of Trust", ce mot clef transmet l'idée que l'on va faire confiance à cette brique pour que la suite du processus se déroule sans problème.

- LLB : Présent sur les processeurs A9 et antérieurs, vient vérifier la signature d'iBoot (la brique suivante) et va aussi vérifier la BootROM.
- iBoot : Sur les versions ultérieures aux processeurs A9 ce mécanisme est lancé par le code contenu dans la BootROM. Authentifie et valide le système d'exploitation avant le démarrage. Communique avec le SSA<sup>8</sup>.
- Noyau iOS & Applications iOS (les deux derniers niveaux) : se rapprochent d'architectures plus "classiques" à partir de cette étape, iBoot démarre le noyau, noyau qui s'occupera de gérer les applications.

Cette *bootchain* se basait sur le fonctionnement du *Application Processor* (AP) seulement. De nos jours sous iOS la *bootchain* utilise un deuxième processeur, le *Secure Enclave Processor* (SEP). Ce processeur est dédié à la cryptographie et aux opérations d'authentifications telles que le TouchID – la fonctionnalité permettant de déverrouiller son téléphone à l'aide de son empreinte digital. Toutes les opérations du SEP sont gérées à l'aide d'un système d'exploitation dédié, SEPOS<sup>9</sup>. À ce jour, le processus de démarrage est similaire à celui illustré dans la figure 3.2.

Étant donné le fonctionnement de la *bootchain* iOS, on peut identifier les points suivants comme étant visés afin de devenir administrateur :

- Passer par la ROM du AP
- Passer par la ROM du SEP
- Passer par iBoot

On peut noter que certains constructeurs de téléphones Android – tels que Google et Samsung – viennent intégrer des processus de sécurité assez similaires.

Par la suite, que l'on utilise un appareil sous iOS ou Android, l'objectif sera le même. Une fois la faille exploitée, on va vouloir installer un programme nous permettant d'utiliser les dits droits d'administrateur, sous Android on retrouvera par exemple le `sudo` assez connu des utilisateurs de Linux. Avec ce programme on peut aussi régulièrement trouver des magasins alternatifs tels que Cydia, Zebra ou Sileo sous iOS et F-droid ou Magisk (qui n'est pas un magasin mais plutôt une application permettant de gérer un téléphone Android avec les droits d'administrateur) sous Android.

---

8. "System Software Authorization", logiciel combinant les clefs de chiffrement gravées dans le matériel avec un service en ligne qui vérifie que l'on n'a pas modifié le logiciel présent sur le téléphone.

9. Micro-noyau fonctionnant avec le *Secure Enclave Processor* (SEP).

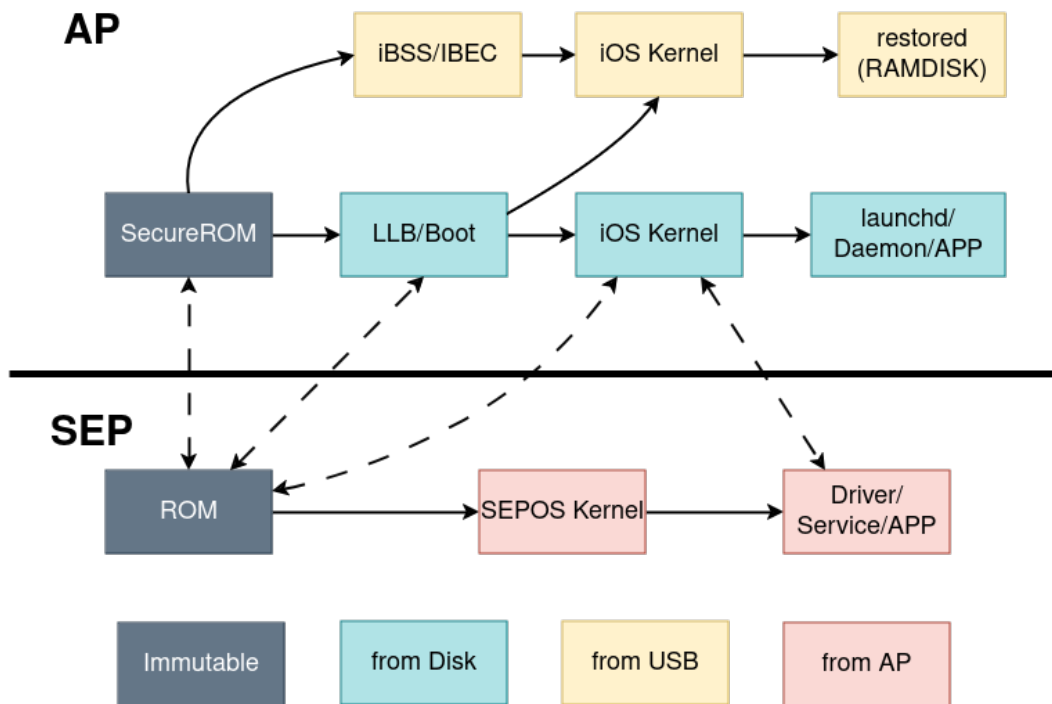


FIGURE 3.2 – Bootchain des appareils iOS

Afin de détecter si un mobile est son propre administrateur, on peut essayer d'utiliser ces droits pour accéder à des informations auxquelles l'on n'est pas censé accéder ou encore la présence d'applications qui sont utilisées avec ces droits, telles que celles citées dans le paragraphe précédent.

## Chapitre 4

# Conclusion

Pour conclure, l'objectif de ce mémoire était de répondre au besoin suivant : "Comment mettre en place un agent vérifiant la conformité d'un poste, quel que soit son système d'exploitation ?".

Plus tôt, on a omis les solutions améliorant cette fiabilité (sec. 2.2). On peut donc se demander comment ces différentes solutions peuvent s'intégrer entre elles et ainsi se bénéficier mutuellement. Typiquement, un anti-virus pourrait mettre en place un jeu de règles venant à détecter les menaces logicielles qui sont sur le poste mais aussi bloquer activement la mise en place de celles-ci, que ce soit par exemple lors d'un téléchargement sur un navigateur, d'une pièce jointe d'un mail ou encore lors du transfert d'un fichier provenant d'une clef USB. On pourrait aussi aider l'agent de conformité avec un EDR qui bloquerait activement les possibles menaces qu'il détecterait, *e.g.* lors de l'exécution d'un script. L'agent

De plus, on a pu explorer la première partie de cette problématique à l'aide de trois fonctionnalités : utiliser un agent qui a pour seul rôle de s'assurer de la conformité d'un parc informatique, ainsi que ses dérivés étant de proposer un agent qui sera

- utilisable par d'autres applications présentes sur un poste
- ou encore visant à détecter les problèmes présents dans ce parc proactivement.

Mais on pourrait imaginer d'autres usages à cette solution. Typiquement, un usage qui n'a pas été développé pourrait être utiliser l'agent de conformité comme un agent d'identification de poste, on sait que le poste  $P$  est censé présenter les

caractéristiques  $X$ ,  $Y$  et  $Z$  ; si l'une d'elles n'est plus présente ou est modifiée ou encore qu'une caractéristique  $W$  qui n'existait pas antérieurement est apparue, l'agent devrait être capable de les identifier et ainsi de remonter ce risque.

La seconde partie de cette problématique a pu être abordée en explicitant comment mettre au point un tel agent d'un point de vue plus fonctionnel que ce soit sous Linux, Windows ou MacOS. On y répond par la suite en explicitant le risque de conformité principal des mobiles : *être son propre administrateur*. Cette tâche est assez compliquée à mettre en place et les versions mobiles n'ont actuellement pas de démon mis en place pour vérifier leur conformité. Malgré les efforts des constructeurs afin de durcir de plus en plus les politiques de sécurité afin d'empêcher leurs appareils d'atteindre ces droits, des méthodes restent découvertes afin de passer outre celles-ci. Certaines solutions permettant d'obtenir ces droits vont d'ailleurs faire en sorte de les cacher au maximum afin d'éviter d'être détectées.

On pourrait imaginer un démon qui avance dans le sens des constructeurs en empêchant le moindre accès à la moindre détection suspecte, ce qui soulève une première question technique : "Comment peut-on détecter les droits d'administrateur au fil de l'avancée du développement de l'agent mais surtout des méthodes d'obfuscation de ces droits?". Cette question pourrait évoluer en prenant en compte un appareil qui aurait volontairement ces droits. Un usage métier nécessitant une sécurité différente de celle fourni par les constructeurs – qu'elle soit moins élevée, équivalente ou plus élevée – pourrait nécessiter un agent qui comme on pourrait le faire sur un PC ne s'opposerait pas à leur présence mais les prendrait dans la politique de conformité.



## Annexe A

# Annexe

### A.1 Mise en lien avec Oversec

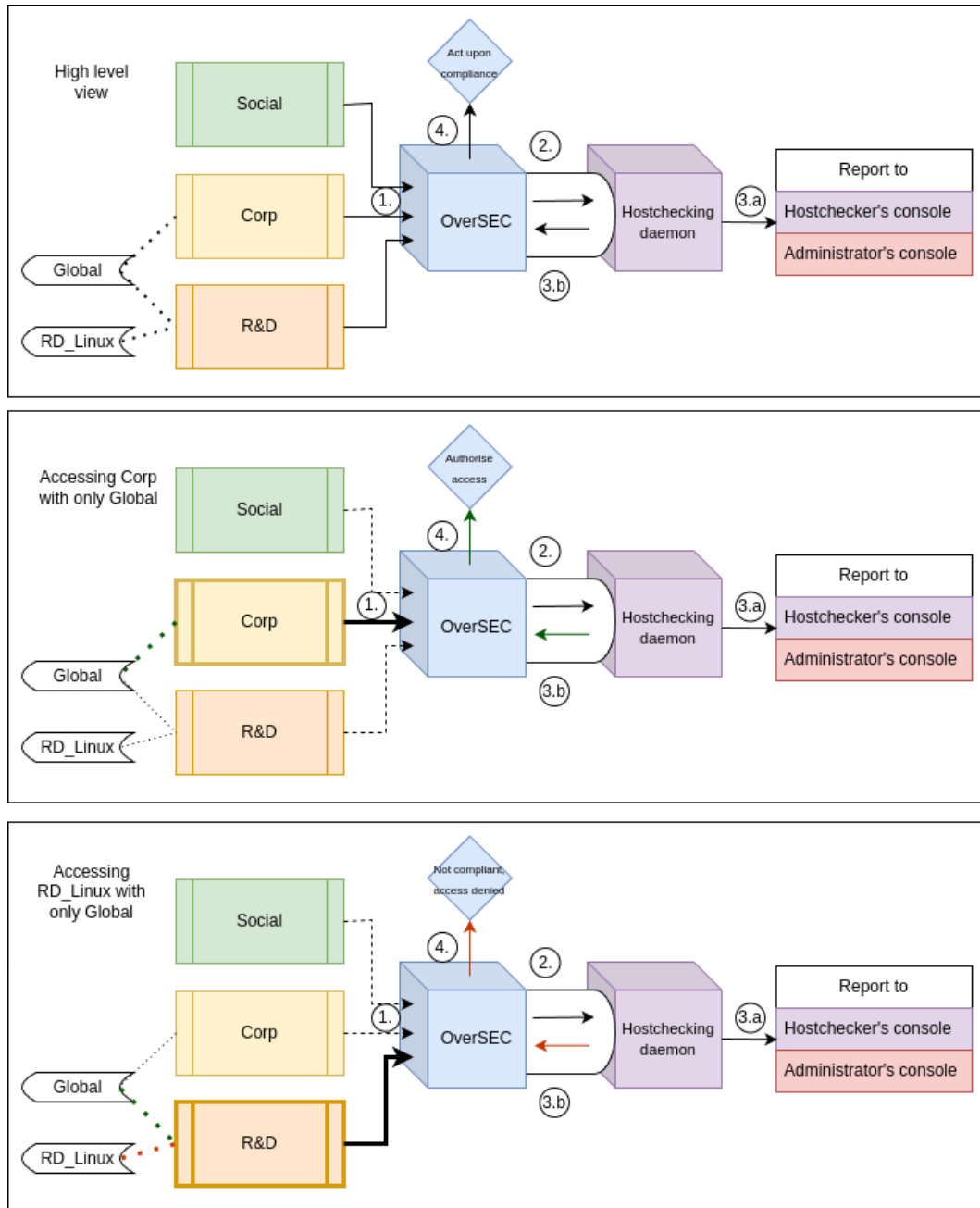


FIGURE A.1 – Description de la mise en lien de Oversec et de l’agent de conformité avec le principe de tags

## A.2 Utilisation plus offensive

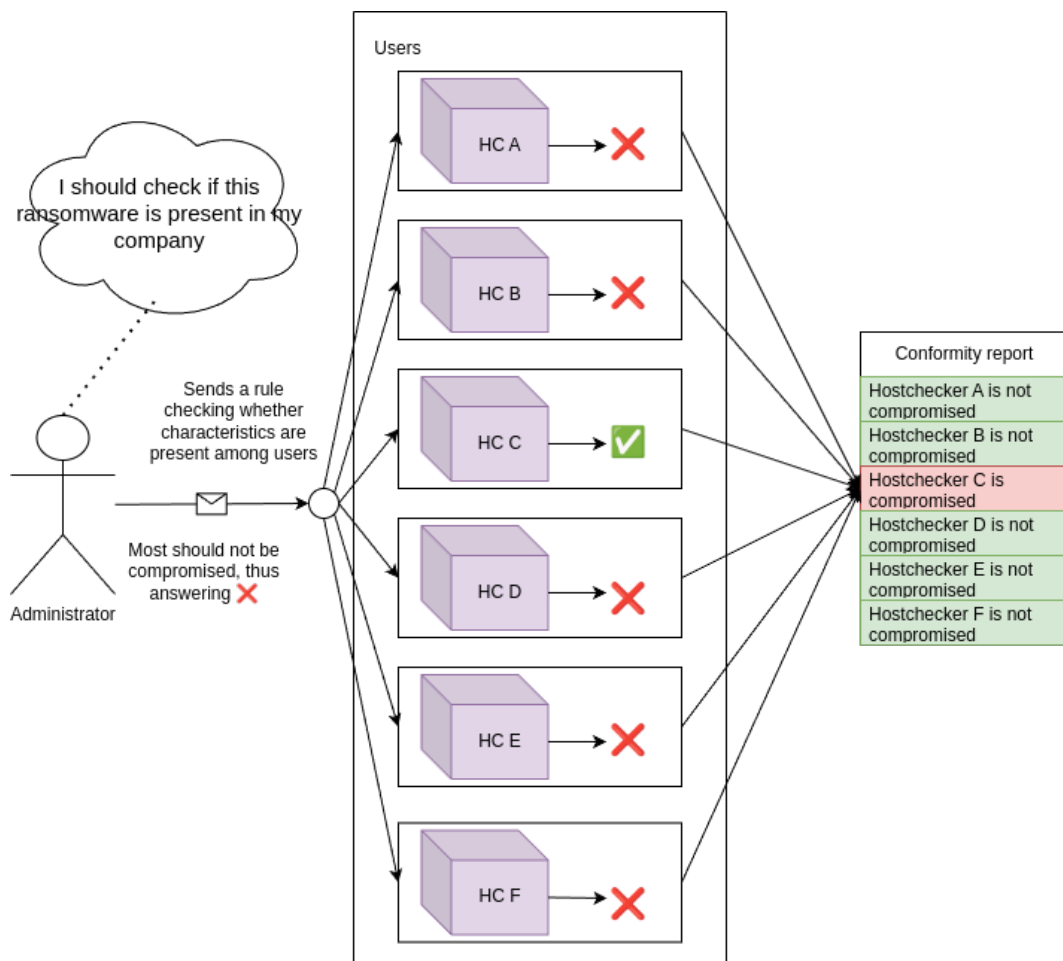


FIGURE A.2 – Parcours lors d'une chasse aux menaces

## A.3 Vue haut niveau du fonctionnement de l'agent

## HostCheck (HC)

### Periodic

This schematic describes how the hostchecking works internally. In the other schematics, the purple box on the right will always be used to describe the HC.

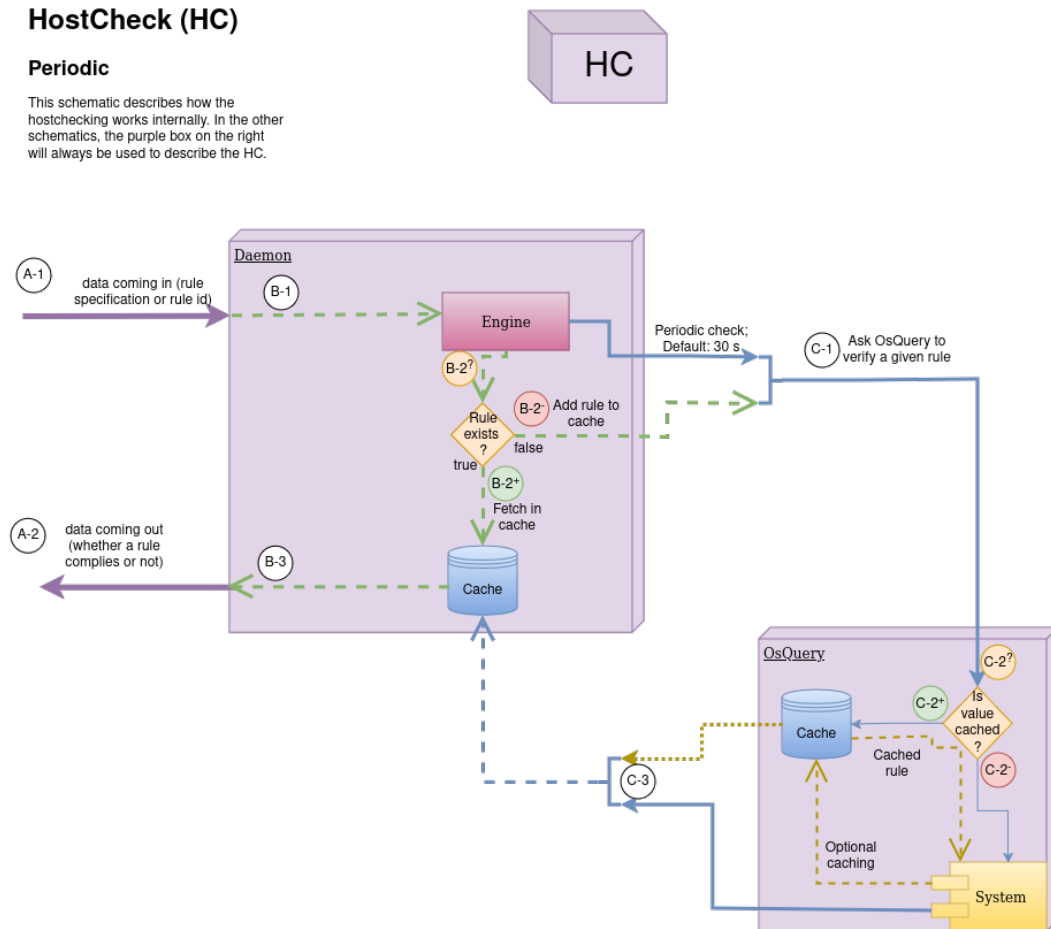


FIGURE A.3 – Schéma montrant le fonctionnement haut niveau de l'agent de conformité.