Titre: Title:	Sécurité véhiculaire: Une approche par système de détection d'intrusion hybride
Auteur: Author:	Bastien Regis Michel Morantin
Date:	2024
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Morantin, B. R. M. (2024). Sécurité véhiculaire: Une approche par système de détection d'intrusion hybride [Mémoire de maîtrise, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/61602/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/61602/
Directeurs de recherche: Advisors:	Nora Boulahia Cuppens, Frédéric Cuppens, & Tarek Ould-Bachir
Programme: Program:	Génie Informatique

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Sécurité véhiculaire: Une approche par système de détection d'intrusion hybride

BASTIEN REGIS MICHEL MORANTIN
Département de génie informatique et génie logiciel

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

Décembre 2024

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

Sécurité véhiculaire: Une approche par système de détection d'intrusion hybride

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

Guy BOIS, président
Frédéric CUPPENS, membre et directeur de recherche
Nora BOULAHIA CUPPENS, membre et codirectrice de recherche
Tarek OULD-BACHIR, membre et codirecteur de recherche
Omar ABDUL WAHAB, membre

DÉDICACE

À mes parents, pour leur amour inconditionnel, et les valeurs qu'ils m'ont transmises, qui m'ont guidé chaque jour dans ce parcours.

REMERCIEMENTS

J'aimerai tout d'abord remercié Nora et Frédéric CUPPENS, pour m'avoir intégré à leur laboratoire de cybersécurité et m'avoir fait découvrir le monde de la recherche scientifique.

Je remercie également Tarek OULD-BACHIR, pour son soutien et pour l'expertise qu'il m'a transmise durant cette recherche.

Un grand merci à Marc-André GUERETTE et Rheinmetall Canada de m'avoir supporté dans ce projet, et pour leurs conseils sur la réalisation du prototype.

Merci à MITACS pour leur généreux soutien qui a rendu possible la réalisation de ce mémoire.

Mes remerciements les plus sincères à Robin HATIER et à Charles KHOURY qui ont contribué au développement de ce projet avec moi. Leur collaboration a été essentielle à l'accomplissement de ce travail.

Enfin, je tiens à exprimer ma gratitude à tous mes collègues et amis pour leurs encouragements constants tout au long de ce mémoire.

RÉSUMÉ

Le transport routier se trouve depuis plusieurs années dans une période de révolution technologique, avec l'avènement des véhicules autonomes et connectés. Cette révolution technologique apporte son lot de solutions, mais également certaines problématiques nouvelles, compte-tenu principalement du besoin de connectivité, qui ouvre l'accès à ces véhicules au monde extérieur. En interne les architectures des réseaux de communications évoluent aussi, en intégrant des réseaux Ethernet, ouverts à la connexion de composants extérieurs comme nos smartphones. Ces réseaux s'ajoutent au bus CAN, aujourd'hui standard pour les communications critiques internes, qui gère notamment les système d'aide à la conduite. Malheureusement, le CAN bus a été créé sans prendre en considération la sécurité informatique, et son absence d'authentification et de chiffrement le rend vulnérable à nombre d'attaques, notamment par injection de messages. Puisque ce système est très largement adopté, il est difficile d'y ajouter de la protection informatique, et la meilleure méthode pour se prémunir des risques d'attaques est à l'heure actuelle l'implémentation d'un système de détection d'intrusion (*Intrusion Detection System* : IDS).

Afin de prémunir les véhicules contre les nouvelles attaques faisant surface, ce travail propose un IDS hybride CAN-Ethernet. Pour cela, nous avons défini les objectifs suivants :

— Établir un état de l'art des attaques et méthodes de détection existantes ;
— Proposer une architecture pour l'intégration d'un tel IDS ;
— Mettre en place un simulateur de conduite pour l'étude des attaques et la génération de données ;
— Évaluer la performance de notre IDS, en terme de précision et de besoins en ressources matérielles.

Nous avons donc établi notre état de l'art à partir de la littérature, et conduit une analyse de risque par la méthode "véhicule routier automatisé", dite VeRa. Ensuite, nous avons tenté de déterminer les stratégies de détection les plus précises et peu coûteuses en ressources, un critère primordial en embarqué. Dans un second temps, nous avons proposé une architecture d'IDS CAN-Ethernet, en détaillant le support matériel et en proposant une méthode de personnalisation de la détection efficace. Il se compose de deux (2) éléments distincts : un IDS CAN bus, basé sur des analyses par règles et une analyse statistique des métriques les plus importantes du bus, et un IDS Ethernet basé sur Snort, un logiciel de détection d'intrusion réseau ouvert et libre de droits. Cette solution a été pensée pour être intégrée en lieu et place de la passerelle CAN-Ethernet existante dans les véhicules, et il a été implémenté

sur deux cartes de développement FPGA, à savoir les cartes PYNQ Z2 et PYNQ ZU.

Pour évaluer cet IDS, nous avons par la suite mis en place deux simulateurs de communications, l'un pour tester la détection d'intrusion, et le second pour analyser l'impact des attaques testées et la génération de données réalistes. Le premier simulateur génère des données aléatoires, tandis que le second peut être utilisé avec un système de conduite réel, couplé avec un simulateur physique réaliste, afin de générer des données similaires à ce qui pourrait être observé en situation réelle.

Nous avons finalement conduit une étude de notre capacité de détection, qui a pu prouver la validité de notre système. Nous avons obtenu une détection par règle d'une précision de 100% sur le bus CAN, nous permettant de vérifier la conformité de chaque message à partir de la spécification du véhicule. L'analyse statistique du bus CAN a quant à elle obtenu une précision de 98,1%, qui détecte toutes les attaques mais remonte quelques faux positifs. Cet IDS est peu coûteux en ressource et peut tourner sur petite plateforme comme la carte Pynq Z2. Côté Ethernet, notre jeu de règles Snort nous a permis d'obtenir une détection de 100% des scénarios d'attaques considérés, pour un usage en ressources raisonnable, autour des 30% de la capacité computationnelle d'une Pynq ZU.

En conclusion, ce travail nous a permis de valider la faisabilité de la détection d'intrusion au sein de ces deux réseaux véhiculaires, ainsi que la possibilité de les implémenter sur des plateformes classiques sans nécessiter de ressources importantes. Cette approche novatrice accompagnée de notre analyse de performance complète, permettra, nous l'espérons, de faciliter la mise en place de telles solutions dans l'industrie. De plus, notre simulateur de conduite constitue un outil d'analyse de cybersécurité complet et réaliste, permettant la génération de nouvelles bases de données, un aspect primordial à l'évaluation d'un tel système. Ce simulateur permet également de mesurer l'impact de divers scénarios d'attaques dans un cas réel, en incluant la réponse du conducteur à la remontée des alertes. Ces fonctionnalités innovantes permettront de futures analyses des vulnérabilités d'un véhicule ou d'un IDS.

ABSTRACT

For several years, road transportation has been undergoing a technological revolution, with the rise of autonomous and connected vehicles. This technological shift brings various solutions but also introduces new challenges, primarily due to the connectivity requirements that open these vehicles to external access. Internally, the architecture of communication networks is also evolving by integrating Ethernet networks, allowing connections with external components like smartphones. These networks supplement the CAN bus, which is currently the standard for critical internal communications, such as managing driver assistance systems. Unfortunately, the CAN bus was created without consideration for cybersecurity, and its lack of authentication and encryption makes it vulnerable to numerous attacks, particularly message injection. Given the widespread adoption of this system, adding cybersecurity protections is challenging, and the best current method to mitigate attack risks is the implementation of Intrusion Detection Systems (IDS).

For this reason, in our work, we propose a hybrid CAN-Ethernet IDS to counter emerging attack vectors. To achieve this, we defined the following objectives:

— Establish a review of existing attacks and detection methods
— Propose an architecture for integrating such an IDS
— Develop a driving simulator for studying attacks and generating data
— Evaluate the performance of our IDS in terms of accuracy and hardware resource requirements.

We conducted a literature review and performed a risk analysis using the VeRa method. Next, we aimed to identify detection strategies that are both highly accurate and resource-efficient—a crucial criterion for embedded systems.

We then proposed a CAN-Ethernet IDS architecture, detailing the hardware support and proposing a method for efficient, customizable detection. This architecture comprises two distinct components: a CAN bus IDS, which utilizes rule-based analysis and statistical analysis of key bus metrics, and an Ethernet IDS based on Snort. This solution was designed to be integrated as a replacement for the existing CAN-Ethernet gateway in vehicles, and it was implemented on two FPGAs: the PYNQ Z2 and PYNQ ZU.

To evaluate this IDS, we developed two communication simulators: one for intrusion detection testing, and another to assess the impact of tested attacks and generate realistic data. The first simulator produces random data, while the second can be used with a real driving system, coupled with a realistic physical simulator, to generate data akin to real-world conditions.

Finally, we conducted a study of our detection capabilities, which validated our system's effectiveness. We achieved 100% rule-based detection on the CAN bus, allowing us to verify the compliance of each message with the vehicle's specifications. Statistical analysis of the CAN bus achieved a detection accuracy of 98.1%, identifying all attacks but yielding a few false positives. This IDS is resource-efficient and can operate on compact platforms like the Pynq Z2. On the Ethernet side, our Snort rule set achieved 100% detection for the considered attack scenarios, with a reasonable resource usage around 30% of the computational capacity of a Pynq ZU.

In conclusion, this work allowed us to validate the feasibility of intrusion detection within these two vehicular networks and demonstrated the possibility of implementing them on conventional platforms with modest resource requirements. We hope this novel approach, along with our comprehensive performance analysis, will facilitate the adoption of such solutions in the industry. Additionally, our driving simulator offers a complete and realistic cybersecurity analysis tool, enabling the generation of new databases—a crucial aspect for evaluating such systems. This simulator also allows the impact measurement of various attack scenarios in real conditions, including the driver's response to alerts. These innovative features will support future analyses of vehicle or IDS vulnerabilities.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

ABD	Automatic Brake Differential
ABS	Anti-lock Braking System
AES	Advanced Encryption Standard
APP	Accelerator Pedal Position
ARP	Address Resolution Protocol
AV	Autonomous Vehicle
AVB/TSN	Audio Video Bridge-Time-Sensitive Network
AXI	Advanced Extensible Interface
BSP	Board Support Package
CAN	Controller Area Network
CAN-FD	Controller Area Network Flexible Data-Rate
CRC	Cyclic Redundancy Check
DB9	D-subminiature 9 pins
DBC	Database CAN
DDR	Double Data Rate
DHCP	Dynamic Host Configuration Protocol
DoIP	Diagnostics over Internet Protocol
DoS	Denial of Service
ECT	Engine Coolant Temperature
ECU	Electronic Control Unit (ou parfois Engine Control Unit)
ESP	Electronic Stability Program
FPGA	Field-Programmable Gate Arrays
HDL	Hardware Description Language
HIDS	Host Intrusion Detection System
HITL	Hardware-In-The-Loop
HMAC	Hash-based Message Authentication Code
IAT	Intake Air Temperature
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IoT	Internet of Things
IP	Internet Protocol
IPC	Instructions per Cycle
MAC (Adresse)	Media Access Control

MAC (Code)	Message Authentication Code
MD5	Message-Digest algorithm
MITM	Man-In-The-Middle
ML	Machine-Learning
MMAC	Mixed Message Authentication Code
NIDS	Network Intrusion Detection System
OBD	On-Board Diagnostics
OS	Operating System
OTA	Over The Air
PL	Programmable Logic
PMU	Performance Monitoring Unit
PS	Processing System
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
RJ45	Registered Jack 45
RPM	Revolutions Per Minute
RTOS	Real-Time Operating System
SHA	Secure Hash Algorithm
SMTP	Simple Mail Transfer Protocol
SOME/IP	Scalable Service-Oriented Middleware over Internet Protocol
SPI	Serial Peripheral Interface
TCC	Transmission Torque Converter Clutch
TCP	Transmission Control Protocol
TCU	Transmission Control Unit
TOT	Transmission Oil Temperature
TPS	Throttle Position Sensor
UART	Universal Asynchronous Receiver Transmitter
UDP	User Datagram Protocol

LISTE DES ANNEXES

CHAPITRE 1 INTRODUCTION

1.1 Contexte et problématique

Dans notre monde moderne, le transport terrestre est devenu un outil indispensable à l'ensemble de la population. Le transport de marchandises par camion représente 65% du transport domestique aux États-Unis [1], et les véhicules personnels représentent 45% de la mobilité mondiale [2], loin devant tout autre moyen de transport. Par conséquent, de nouvelles technologies sont développées chaque jour afin d'améliorer la sécurité et le rendement de ces systèmes, ce qui les rend également plus complexes, et par conséquent plus vulnérables à de potentielles attaques.

Les véhicules routiers, à l'origine purement mécaniques, et donc loin de toute menace informatique, ont dû évoluer avec le monde qui les entoure, au point d'être aujourd'hui pilotés par des dizaines d'ordinateurs et capteurs [3]. Ces ordinateurs ayant besoin de communiquer entre eux, l'architecture interne du véhicule a dû être repensée afin d'inclure un réseau stable qui assure une latence très faible et une perte de données quasi-nulle. C'est ainsi que le bus Controller Area Network (CAN) [4,5] est né et est devenu un des standards pour les réseaux de communication intra-véhiculaires. Ce bus de communication est le support pour tous les échanges de messages essentiels au bon fonctionnement du véhicule, qui inclut le groupe motopropulseur, la commande du véhicule et les systèmes de sécurité du véhicule tels que l'ABS et l'ESP, entre autres. Chacun des ordinateurs de ce réseau interne performe de nombreuses opérations afin d'assister le conducteur du véhicule, et ils hébergent donc des programmes informatiques complexes, qui dépassent souvent les 100 millions de lignes de code [6] en cumulé, ce qui implique nécessairement des failles potentielles qui permettraient de prendre le contrôle du véhicule.

Cette révolution ne concerne pas uniquement l'architecture fonctionnelle du véhicule. En effet, les systèmes d'infodivertissement ont également beaucoup évolué ces dernières années. En premier lieu pour intégrer des fonctionnalités de connexion avec les smartphones des utilisateurs, mais aussi pour les communications externes, comme la connexion au réseau cellulaire, ou à Internet. Ainsi, cette nouvelle génération de systèmes d'infodivertissement augmente considérablement la surface d'attaque. Lors de leurs expérimentations, les auteurs de [7] ont révélé des failles permettant de pirater le bus CAN d'un véhicule, à l'aide d'un téléphone intelligent connecté au système d'infodivertissement, contenant une application malveillante. Cette attaque exploite une faille dans le code des fonctions binaires de communication entre le bus CAN et le système d'infodivertissement.

De plus, les fonctionnalités de connectivité cellulaires peuvent elles aussi induire des points d'entrée au réseau interne du véhicule, comme l'ont montré C. Valasek et C. Miller [8]. En 2015, ces chercheurs ont attaqué une Jeep Cherokee via le système d'alerte connecté au réseau cellulaire et en ont fait une démonstration en direct pour le média *Wired*, suscitant une prise de conscience chez de nombreux utilisateurs et acteurs de l'industrie automobile. Pour cette attaque, les chercheurs ont montré non seulement que le système d'infodivertissement d'une Jeep Cherokee de 2014 permettait d'altérer le bon fonctionnement du bus CAN, mais ils ont également montré que le système d'infodivertissement pouvait être manipulé via le réseau cellulaire, et donc à distance, sans accès physique au véhicule. Lors de cette attaque, les attaquants ont pu démontrer qu'ils avaient un accès complet au véhicule, qui leur permettait d'empêcher l'accélération, mais également de le faire avancer et même le faire tourner.

Ces dernières années, la menace a pris de l'ampleur avec l'avènement d'une nouvelle révolution technologique : les véhicules autonomes. Bien que ces véhicules soient encore peu présents dans notre environnement routier, Renub Research prévoit que le marché croîtra de près de 20%par an jusqu'en 2030, et 51% des Américains envisageraient de passer à une voiture autonome dans les cinq prochaines années [9]. Afin de répondre à cette demande, les manufacturiers automobiles travaillent sur des modèles de plus en plus sophistiqués, dont l'autonomie est mesurée selon la taxonomie J3016 de la Society of Automotive Engineers (SAE) [10]. Cette taxonomie divise l'autonomie d'un véhicule en 6 niveaux, dont les 3 premiers concernent des assistances, quand les 3 derniers concernent une autonomie plus complète, sous certaines conditions ou non. C'est dans ces 3 dernières catégories que le plus gros danger réside, car dans le cas d'une cyberattaque cela impliquerait que l'attaquant pourrait lui aussi tout contrôler du véhicule. Afin de fonctionner correctement, la majorité de ces véhicules sont contraints de communiquer avec leur environnement extérieur, que ce soit entre véhicules (V2V), de véhicules à infrastructure (V2I), de véhicule à piéton (V2P) ou véhicule à réseau (V2N), ce qui induit une nouvelle surface d'attaque extérieure. Afin de mieux cerner les risques liés à cette nouvelle génération de véhicules, G. De La Torre et al. [11] ont défini un ensemble de challenges à relever pour en assurer la sécurité. Parmi ces objectifs se trouve le développement d'un système de détection d'intrusion pour le bus CAN. Ils précisent en outre que le gateway permettant d'accéder à ce réseau devrait être protégé, pour se prémunir notamment d'attaques venant de réseaux moins sécurisés, comme Ethernet. T. Rosenstatter et al. [12] ont également tenté de définir avec plus de précisions les enjeux de la sécurité au sein des véhicules, en commençant par analyser différents types d'attaques, mais en analysant également la régulation en vigueur, la sûreté des véhicules, ou encore l'impact économique. Parmi les solutions proposées pour mitiger les risques de ces véhicules, ils ont placé les systèmes de détection d'intrusion (IDS) comme des requis des futurs

réseaux véhiculaires internes. Ils ont notamment comparé différents types d'IDS, proposé différents capteurs pouvant être inclus dans ceux-ci, et ont en sus cherché comment mitiger ces nouvelles formes d'attaques. Ainsi, nous pouvons voir que les véhicules intègrent dans leur réseau interne des vulnérabilités importantes, de plus en plus accessibles grâce à une surface d'attaque grandissante, et une automatisation de la conduite qui donne accès à l'ensemble de la structure du véhicule, ce qui représente un risque considérable pour la sécurité des usagers. L'Agence de l'Union Européenne pour la Cybersécurité (ENISA) a d'ailleurs identifié les systèmes legacy exploités au sein des systèmes cyber-physiques comme étant le 3e plus important risque informatique d'ici 2030 [13]. Les systèmes legacy sont des systèmes qui ont été créés il y a plusieurs années et qui sont aujourd'hui devenus standards dans l'industrie, au point d'être encore largement utilisés malgré leur manque d'adaptation aux nouveaux besoins. Le bus CAN est un système de ce type, puisqu'il n'est équipé d'aucun système de sécurité informatique, mais il continue d'être utilisé dans la grande majorité des véhicules de par son quasi-monopole. C'est pourquoi nous avons tenté, dans cette recherche, de sécuriser le réseau interne des véhicules nouvelle génération en développant un système de détection d'intrusion hybride, capable de détecter des attaques sur le bus CAN ainsi que le réseau Ethernet interne afin de couvrir le plus largement possible la surface d'attaque du véhicule.

1.2 Objectifs de la recherche

Ce mémoire contribue au domaine en présentant un système de simulation du réseau interne d'un véhicule, en incluant un système de bus CAN et réseau Ethernet physique, ainsi que diverses fonctionnalités de détection d'intrusion et d'analyses de performances. Tout d'abord, nous allons, afin de tester notre système, développer un banc d'essai flexible et à faible coût pour simulation d'un réseau intra-véhiculaire, puis nous allons développer un système de détection d'intrusion (IDS) hybride pour analyser les réseaux CAN et Ethernet, basé sur FPGA. Les objectifs de cette recherche sont les suivants :

- Établir un état de l'art des attaques existantes et des méthodes de détection associées au sein d'un réseau véhiculaire hybride CAN-Ethernet
- Démontrer la faisabilité d'un Système de Détection d'Intrusion (IDS) pour réseaux internes véhiculaires hybrides CAN-Ethernet
- Développer un banc d'essai de réseaux internes véhiculaires flexible et réaliste permettant des analyses de cybersécurité
- Analyser la performance de détection et les besoins matériels d'un tel système

L'accomplissement de ces objectifs permettra de mettre en évidence une solution de sécurisation et de la valider à travers des tests réalistes.

1.2.1 Contributions de la recherche

En répondant aux objectifs de recherche énoncés plus haut, ce travail permettra les contributions suivantes au domaine :

- Réalisation d'une analyse de risque détaillée des attaques sur les bus CAN et les réseaux Ethernet
- Création d'un banc d'essai à faible coût pour l'étude des attaques et de la détection sur réseaux CAN et Ethernet, avec duplication matérielle des communications
- Analyse hybride du bus CAN par règle et par analyse statistique
- Adaptation facile de l'IDS à partir d'un fichier Database CAN (DBC), notamment grâce à l'ajout de règles
- Analyse du placement au sein du véhicule et caractérisation des ressources nécessaires afin de réduire l'écart entre le prototype de recherche et une application dans un environnement réel.

Ainsi nous proposerons un outil pratique et efficace pour améliorer la sécurité des véhicules autonomes, grâce au test de divers scénarios d'attaque et stratégies de détection, le tout dans un cadre facilement adaptable aux différentes architectures.

1.3 Organisation du mémoire

La suite de ce mémoire est organisée comme suit.

Dans un premier temps, nous introduirons le contexte de cette recherche et les besoins en termes de systèmes de détection d'intrusion intra-véhiculaires. Afin de préparer notre prototype, nous procéderons à une analyse des méthodes et bonnes pratiques permettant de développer des composants embarqués sécurisés, en mettant l'accent sur les contraintes liées au monde du transport. Nous poursuivrons avec un état de l'art détaillé des attaques recensées sur le bus CAN, ainsi que des systèmes de détection d'intrusion existants sur ce réseau, dans le but d'obtenir une classification desdits IDS. Nous analyserons ensuite les attaques Ethernet recensées dans le cas d'un réseau véhiculaire, avant d'analyser les IDS Ethernet spécialisés dans ce contexte. Nous finirons notre état de l'art en analysant d'autres IDS véhiculaires, qui se basent sur des sources d'informations non-incluses dans les catégories précédentes, telles que les analyses par canal auxiliaire (Side-Channel Analysis).

Dans un second temps, nous effectuerons une étude de l'architecture de notre prototype. Cette étude commencera par le choix d'un emplacement stratégique au sein du réseau interne du véhicule, et se poursuivra par le choix du support matériel et logiciel utilisé, afin d'obtenir

un système capable d'interagir avec son environnement en temps réel. Nous procéderons ensuite à la sélection de multiples méthodes de détection d'attaques pour notre IDS. Nous commencerons par l'élaboration de règles pour le réseau CAN, avant d'établir des méthodes de détection statistiques basées sur différentes métriques du réseau que nous détaillerons. Nous terminerons l'élaboration du système de détection par une étude de Snort dans le cas de l'embarqué, afin d'établir un jeu de règles adaptées. Nous préciserons finalement les techniques utilisées pour remonter les alertes et faire de la journalisation des événements et communications.

Dans un troisième temps, nous détaillerons l'implémentation finale de notre prototype, en commençant par la configuration de notre FPGA pour supporter l'IDS, ainsi que les composants annexes utilisés. Nous exposerons ensuite la partie simulation logicielle du véhicule, qui nous permettra de valider notre prototype. Nous finirons cette partie par une analyse de performance de notre système, selon deux axes spécifiques : la détection d'intrusion, et la consommation de ressources.

Dans un quatrième temps, nous synthétiserons nos travaux de recherches, et de les analyser afin d'en trouver les limitations, et de proposer des améliorations futures.

Nous finirons ce mémoire par une conclusion sur les apports de cette recherche et les résultats obtenus.

CHAPITRE 2 REVUE DE LITTÉRATURE

2.1 Contexte

L'émergence des véhicules autonomes révolutionne le paysage technologique, introduisant des défis significatifs en termes de sécurité et d'efficacité. Ces véhicules dépendent fortement des protocoles de communication, tels que le bus CAN ou Ethernet, cruciaux pour l'échange de données en temps réel entre les différents composants du véhicule. À mesure que les véhicules autonomes deviennent plus répandus, il est essentiel de garantir des mesures de cybersécurité robustes pour leurs systèmes de communication, afin de les protéger contre les vulnérabilités et les menaces potentielles, et d'empêcher les attaques malveillantes de compromettre la sécurité et la fonctionnalité des véhicules [14].

Dans un rapport publié en 2016 sur la cybersécurité et la résilience des voitures intelligentes [15], l'ENISA a souligné l'importance de la sécurité par design de cette nouvelle génération de véhicules, afin de mettre en place des stratégies de défense en profondeur. Ce type de stratégie définit les prérequis pour obtenir un système sécurisé du plus bas niveau jusqu'au plus haut, en prenant en compte les considérations de cybersécurité dans l'ensemble des étapes de développement, et ce pour l'ensemble des acteurs d'un projet. Ils soulignent dans ce même rapport l'absence d'application de ces méthodes dans l'industrie, ce qui provoque des manques en termes de protection des communications, notamment l'absence d'authentification et de système d'autorisation. De plus, les auteurs regrettent l'absence de mécanismes de mitigation des attaques, mêmes connues, ainsi que le manque d'outils permettant le diagnostique des attaques, comme par exemple la journalisation. Les manufacturiers ont un rôle important à jouer dans la sécurisation de leurs véhicules, mais le rapport indique que nombre d'entre eux ralentissent l'émergence de ces solutions, pour des raisons légales, de compatibilité, et des raisons économiques. Ils concluent en détaillant les risques liés à cette nouvelle génération de véhicule, à cause d'une surface d'attaque accrue, d'un accès facile au véhicule, et de la possibilité de mettre en danger des utilisateurs dans le monde réel. Ces attaques sont classées par risques, et la quasi-totalité de celles-ci sont considérées comme critiques ou majeures. L'une des pistes suggérées pour se protéger des attaques est le développement d'un IDS, en particulier pour le bus CAN.

CAN est un protocole de communication largement adopté dans l'industrie automobile, conçu pour une transmission de messages efficace et fiable entre les unités de contrôle électronique (ECU). Malgré ses avantages, la sécurisation du bus CAN pose des défis considérables en raison de sa nature de diffusion et de l'absence d'authentification et de chiffrement [16]. Les

mesures de sécurité traditionnelles sont souvent insuffisantes, laissant le système vulnérable à des attaques telles que le déni de service (DoS), l'injection de messages et l'écoute clandestine. La littérature met en évidence diverses méthodes pour renforcer la sécurité du bus CAN, notamment le chiffrement, l'authentification et les IDS [17].

Hussain et al. [18] ont analysé les enjeux futurs pour les voitures autonomes et les ont divisés en 4 catégories : Coût, Cartes, Complexité logicielle, Simulation. Le coût limite le développement de solutions avancées incluant de la cybersécurité, et la seule solution pour limiter cet augmentation de prix est de parvenir à une production de masse, ce qui implique l'acceptation par l'industrie des nouvelles solutions de sécurité. Dans le but de faciliter cette transition, il convient de développer des solutions compatibles avec la majorité des produits existants sur le marché, avec suffisamment de flexibilité pour être intégrées dans des contextes variés. La complexité logicielle pose quant à elle des difficultés pour le test et la validation, tout en induisant de possibles failles. Il faut donc développer des systèmes les plus simples possibles tout en s'assurant d'obtenir un produit sûr et sécurisé. Enfin, le 3e enjeu est la simulation, car il n'existe pas de simulateur universel, et la majorité étant propriétaires, il peut être difficile d'y avoir accès. L'article suggère donc l'usage d'outils open-source, et la mise à disposition des données permettant de bonnes analyses. Enfin, ils ajoutent que la confiance des utilisateurs est un enjeu social primordial, et qu'il faut donc pouvoir les rassurer, vis-à-vis par exemple de risques informatiques.

Pour répondre à ce besoin de nouveaux composants sécurisés à faible coût, M. Amin et al. [19] ont étudié la relation manufacturiers-fournisseurs pour voir comment elle pourrait diminuer les vulnérabilités. Ils se sont pour cela appuyés sur la comparaison de deux projets de co-développement technique manufacturier-fournisseur par Cabigiosu et al. [20]. Cette analyse leur a permis de conclure qu'une approche intrusive de la part du manufacturier permettait de simplifier le processus de création, tout en obtenant un meilleur produit, et surtout une maîtrise du produit pour le manufacturier, ce qui lui permet de mieux l'utiliser dans son environnement. Bien que prometteuses, ce genre de méthodes d'ingénierie est encore trop peu fréquentes, à cause de la complexité de mise en place de ces collaborations.

Cependant, la sécurité des composants n'est pas le seul aspect à prendre en compte pour sécuriser un véhicule. Dans leur article *Demystifying Plaform Cyber Resilience* [21], Vriesenga et al. découpent la résilience en 5 piliers :

1. Préparer *(Prepare)*
2. Prévenir *(Prevent)*
3. Détecter *(Detect)*
4. Répondre *(Respond)*

5. Récupérer *(Recover)*

Ce découpage nous permet de voir que, bien que la production de composants sécurisés permette de préparer et prévenir les attaques, assurer la résilience passe également par la détection d'attaques, la réponse apportée à celles-ci et et la manière de revenir dans un état sûr. De plus, ils introduisent la notion de "Resilience-In-Depth", et donnent 5 niveaux dans lesquels la résilience doit être mise en place :

1. Puce *(Chip)*
2. Carte *(Board)*
3. Assemblage *(Assembly)*
4. Bus *(Bus)*
5. Plateforme *(Platform)*

Nous avons vu en début d'état de l'art que les trois premiers niveaux dépendent des fournisseurs et de l'interaction avec les manufacturiers, nous allons maintenant donc analyser les bus de communication. Le niveau plateforme ne sera pas détaillé ici, puisqu'il concerne les communications externes au véhicule et est donc hors des objectifs de recherche de ce mémoire. Dans la suite de leur rapport [21], les auteurs indiquent que la méthode permettant de détecter le plus d'attaques venant du niveau bus est le développement d'un *Système de Prévention et Détection d'Intrusion*. Nous allons donc commencer par analyser les bus de communications, avant de confirmer cette théorie par d'autres sources. La figure 2.1 présente les 5 niveaux de la cyberrésilience d'une plateforme véhiculaire, ainsi que le niveau où se situe notre IDS.

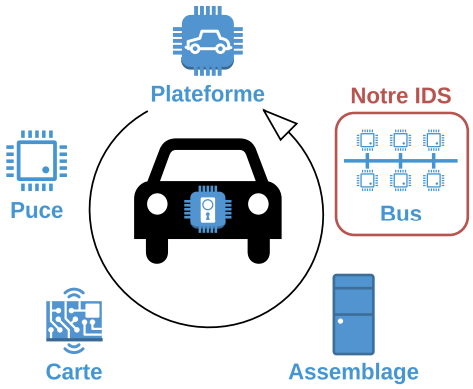

FIGURE 2.1 Cyberrésilience d'une plateforme.

S. Tuohy et al. [22] ont proposé un état de l'art de ces réseaux de communication interne, en prenant en compte les réseaux legacy mais également les réseaux émergents. Ils commencent donc par une classification des protocoles de la couche physique, selon les besoins auxquels ils

répondent, avant de les comparer aux solutions émergentes, utilisant Ethernet. Ils montrent ainsi que CAN reste l'un des protocoles les plus importants puisque c'est celui qui va permettre la gestion du train roulant, du moteur et de tous les composants critiques. Lors de leur analyse des protocoles Ethernet, ils ont montré la diversité des protocoles possibles, et ont conclu en indiquant que ces technologies remplaceront inévitablement le bus CAN sur le long terme, en commençant par remplacer les réseaux non critiques tels que l'infodivertissement ou les caméras d'aide à la conduite.

Dans un autre rapport de l'ENISA [23], une taxonomie des menaces est présentée, et toutes les attaques informatiques présentées concernent le bus de communication interne, avec des attaques de rejeu, d'écoute, ou d'Homme au milieu ou encore de déni de service. Les attaques réelles se basant sur ces vulnérabilités sont classées comme à haut risque, et des bonnes pratiques pour limiter ce risque sont données en fin de rapport. Parmi ces pratiques, les systèmes de détection d'intrusion sont proposés, notamment au sein du véhicule. Dans un article plus technique, Koscher et al. [24] ont procédé à une analyse de la sécurité d'un véhicule moderne, ils ont établi une longue liste d'attaques que nous détaillerons plus loin dans ce mémoire, puis ont tenté d'analyser comment limiter ces attaques. Ils ont conclu que bien que la prévention soit la solution idéale, elle est dans les faits très compliquée à mettre en place, et ils suggèrent donc la recherche d'un IDS comme travail futur.

2.1.1 Système de Détection d'Intrusion (IDS)

L'IDS est un composant critique de la cybersécurité, conçu pour surveiller le trafic réseau à la recherche d'activités suspectes et de menaces potentielles. Il peut être classé en deux types principaux : la détection basée sur les signatures et la détection basée sur les anomalies. En détectant et en alertant les administrateurs sur les violations potentielles de la sécurité, l'IDS joue un rôle vital dans le maintien de l'intégrité et de la sécurité des systèmes de communication, tels que le bus CAN dans les véhicules autonomes. Son emplacement stratégique et sa mise en œuvre appropriée peuvent améliorer considérablement la résilience d'un système contre les cyberattaques [25].

2.2 Outils d'analyse

Dans cette partie, nous allons définir les différents outils que nous utiliserons dans ce mémoire pour analyser les risques au sein du véhicule, les différentes attaques recensées, ainsi que les IDS véhiculaires. Tout d'abord, afin de faciliter la lecture, nous utiliserons les termes techniques issus de l'article *A Common Lamguage for Computer Security Incidents* [26] sans

les détailler. Ces termes sont aujourd'hui très largement répandus dans le milieu de la sécurité informatique, et bien que destinés à décrire des incidents dans un contexte classique, ils sont également adaptés au monde de l'embarqué. Le lecteur curieux est invité à se référer à ce document pour toute information complémentaire sur les termes techniques.

2.2.1 Analyse de risques et modèle de menace

Afin de pouvoir garantir la sécurité d'un véhicule moderne, il convient avant tout d'en faire une analyse de risques. Les méthodes d'analyses de risques sont nombreuses, et pas toujours adaptées au monde de l'automobile, nous nous contenterons donc d'en présenter deux dans cet état de l'art, qui ont été créées spécifiquement pour les réseaux véhiculaires.

Schmittner et al. [27] ont comparé deux méthodes d'analyse de risques dans le cas d'un véhicule : Failure Mode, Vulnerabilities and Effects Analysis (FMVEA) et Combined Harm Assessment of Safety and Security for Information Systems (CHASSIS), qui ont deux approches différentes du problème. FMVEA part du système complet, et pour chaque composant, conduit une analyse des modes de vulnérabilités ou d'échecs. Il faut ensuite en détailler les effets afin de mesurer le sévérité du risque, et finalement il faut chercher les situations pouvant provoquer un changement d'état vers ces modes, ce qui permet d'établir une probabilité d'occurrence de l'incident. CHASSIS quant à lui, propose de commencer par établir un diagramme de cas d'usage, avec une description textuelle, et un diagramme de séquence permettant de définir le fonctionnement du système. Dans un second temps, les requis de sécurité et de sûreté et des cas d'usages impropres sont définis à l'aide de discussions avec des experts du domaine. La 3e étape consiste à définir un second diagramme, mais cette fois-ci décrivant les cas d'usage impropre, en utilisant également une description textuelle et des diagrammes de séquence. Finalement, une fois les diagrammes d'usage impropres établis, une analyse des méthodes de mitigation de ces cas d'usage est conduite afin d'en estimer les enjeux. Les auteurs ont appliqué ces deux méthodes à un système de mise à jour en direct d'un ECU, et ont comparé les résultats obtenus. Leur analyse montre que FMVEA est un outil plus précis et technique, mais qu'il nécessite une bonne connaissance de l'architecture du dispositif, ce qui le rend dépendant du niveau de détails connu et très sensible aux changements de contextes. CHASSIS quant à lui est réutilisable facilement dans divers contextes de par son niveau d'abstraction, mais son analyse du risque est dépendante du niveau d'expertise des professionnels impliqués. On remarque aussi que l'article ne propose pas de méthode claire permettant d'évaluer le risque, et laisse le soin aux experts de choisir cette métrique.

Cui et Zhang [28] ont proposé leur propre méthode d'analyse de risque pour véhicule "Vehicles Risk Analysis Method (VeRA)". Ils ont pour cela opté pour une approche différente, qui se

base cette fois sur les attaques plutôt que l'architecture du dispositif considéré. Il est donc plus difficile d'identifier les potentielles failles et attaques, mais leur méthode leur permet de prendre en compte d'autres aspects de l'attaque comme le niveau d'autonomisation du véhicule et la capacité pour le conducteur de corriger cette attaque. Ainsi les auteurs proposent une méthode simple de calcul du risque :

$$R = H + S \times P \tag{2.1}$$

Dans cette formule :

- R représente le risque,
- H représente le contrôle humain sur le système, qui dépend du niveau d'autonomie du véhicule et de la connaissance de l'utilisateur,
- S représente la sévérité de l'attaque pour l'usager, selon les critères de sûreté, confidentialité, impact économique et impact opérationnel,
- P représente la probabilité de l'attaque, qui dépend elle-même de deux critères :
  - E : l'équipement nécessaire pour effectuer l'attaque,
  - K : la connaissance nécessaire pour que l'attaquant puisse effectuer l'attaque.

Cette formule permet donc une estimation réaliste du risque d'une attaque, prenant en compte les spécificités de l'automobile, notamment des niveaux d'autonomie du véhicule. Nous voyons donc que cette méthode permet une classification précise des risques, mais requiert une connaissance préalable des vulnérabilités du système.

Pour conclure cette section, nous pouvons voir qu'en cumulant une méthode telle que FMVEA ou CHASSIS pour identifier les risques, à une méthode d'évaluation comme VeRA permet d'obtenir une bonne classification du risque dans le cas des véhicules autonomes.

En complément de l'analyse de risques, il est nécessaire, lors de l'analyse de la sécurité d'un système, de mettre en place un modèle de menace. Un tel modèle permet de représenter d'une manière facilement compréhensible la structure d'une attaque, en analysant quels atouts sont visés, dans quel but, par qui, et par quels moyens.

Hamad et al. [29] ont proposé un modèle pour répondre à ce besoin dans le cas des véhicules, qu'ils ont baptisé "SAVTA (Software-Asset-Vulnerability-Threat-Attacker)". Ce modèle est hybride, puisqu'il se base sur une approche mêlant les 5 points de vue habituels des modèles de menace, et qu'il inclut des informations sur les attaques par des arbres d'attaque. Pour utiliser ce modèle d'attaque, nous devons commencer par établir les arbres d'attaque, en définissant 3 points essentiels : le profil de l'attaquant, le composant considéré, et les requis de sécurité associés à ce composant. Pour chacun de ces points, il est essentiel de donner des

détails : les moyens et la connaissance de l'attaquant, les vulnérabilités du composant, ou les détails techniques des requis. Une fois cette première étape effectuée, la seconde consiste à déduire de la première étape une variété d'arbres d'attaques, qui nous donnerons finalement la base de notre modèle final. Chacun de ces arbres trouve en sa racine un objectif final, et une suite de porte logiques et de nœuds qui définissent comment réaliser l'attaque. Si une attaque peut être conduite par plusieurs méthodes, nous mettrons une porte "ou", puis chacune de ces méthodes sera insérée dans le niveau inférieur. Si nous avons besoin de plusieurs requis pour une attaque, nous utiliserons une porte "et". A la fin, nous obtenons un arbre qui décrit de manière explicite chaque attaque et comment l'effectuer, ce qui facilite grandement l'analyse de risques. Pour la troisième étape de l'analyse, nous allons associer les arbres d'attaque avec les atouts visés, en incluant quelques informations supplémentaires. Pour chaque atout, nous allons regarder quelles attaques les concernent, et pour chacune de ces attaques, nous allons chercher quel aspect de la triade CIA (Confidentialité, Intégrité, Disponibilité) cette attaque menace. Ensuite, nous allons définir l'accessibilité d'une telle attaque, qui se divise en 3 catégories : à distance, en accès direct, ou une combinaison des deux. Une fois ces étapes effectuées, nous allons pouvoir obtenir un graphe des attaques connues, qui partent d'un composant spécifique et arrivent aux moyens de l'effectuer. La dernière étape de modélisation consiste à identifier quels types d'attaquants sont susceptibles d'effectuer quels types d'attaques, avec quelles motivations, et surtout quelles connaissances et quelles ressources. Ainsi, nous obtenons finalement un graphe détaillé de l'ensemble des attaques possibles sur un système, avec toutes les informations nécessaires pour conduire une analyse de risques comme par exemple VeRA présentée plus haut.

2.2.2 Classification des attaques

Une fois notre analyse de risque effectuée, nous devons nous demander quelles sont les attaques existantes qui ont déjà été recensées, et comment les classifier. Pour cela, la littérature a proposé plusieurs outils.

Studnia et al. [30] ont proposé un état de l'art des attaques et défenses existantes dans les réseaux internes véhiculaires. Dans leur article, ils proposent une classification pour ces deux éléments, qu'ils utilisent ensuite. Pour la classification des attaques, ils proposent une taxonomie simplifiée qui se limite à identifier l'objectif de l'attaque, ainsi que le vecteur, qui sera divisé en deux catégories : attaque interne ou attaque externe. Ce modèle leur permet de classer les attaques facilement, mais le niveau d'abstraction implique une perte d'information considérable, comme par exemple les outils utilisés ou les failles exploitées. Cependant, l'article propose également une classification des défenses qui permet de facilement lier les deux,

ce que le reste de la littérature ne fait pas. Cette partie ne sera pas détaillée ici puisqu'elle est hors du sujet de ce mémoire, mais il est intéressant de remarquer que parmi ces défenses, la détection d'intrusion est incluse dans les défenses passives de détection d'attaques.

Thing et al. [31] ont proposé une classification similaire, mais avec des informations complémentaires. Tout d'abord, le vecteur d'attaque est une nouvelle fois divisé en deux catégories similaires aux précédentes, mais cette fois-ci nommées "Accès Physique" et "Accès Distant". Ensuite, trois nouveaux critères de classification sont donnés : la cible, l'objectif et la conséquence potentielle. La cible décrit un composant du véhicule que l'attaquant cherche à compromettre, comme par exemple un ECU. L'objectif définit l'action malveillante que l'attaquant tente d'effectuer, comme par exemple l'injection de messages malveillants. Finalement, la conséquence potentielle décrit le résultat final que l'attaque va provoquer, comme par exemple forcer le véhicule à s'arrêter. Grâce à ces critères supplémentaires, nous pouvons mieux comprendre le fonctionnement des attaques et donc analyser les risques et les défenses associées.

Shalaka [32] propose également une comparaison des classifications possible des attaques. Il compare les modèles centrés sur les attaques, les composants, les vulnérabilités et les logiciels. L'approche centrée sur les attaques correspond aux deux modèles présentés plus haut, mais ils proposent dans leur modèle un critère intéressant, à savoir les limitations de l'attaque. Cette information permet de mieux comprendre la difficulté liée à l'exécution de l'attaque, mais aussi la difficulté de mise en place des mécanismes de défenses associés. L'approche basée sur les composants est similaire aux méthodes d'analyse de risque présentées plus haut, elle consiste à établir une mesure du risque lié à un composant parmi les plus critiques, en identifiant des menaces dans le fonctionnement. L'approche par vulnérabilité est peu détaillée dans ce rapport, elle consiste en une liste de vulnérabilités connues, puis il faut ensuite établir une liste d'attaques pouvant les exploiter, avant de les trier selon leur priorité. Finalement, l'approche basée sur les logiciels propose un nouvel outil intéressant : STRIDE [33]. STRIDE est un modèle d'attaque développé par Microsoft, qui classifie les attaques selon 6 catégories :

— Vol d'identité (Spoofing Identity)
— Manipulation de données (Tampering with data)
— Répudiation (Repudiation)
— Divulgation d'information (Information Disclosure)
— Déni de service (Denial of Service)
— Élévation de privilèges (Elevation of privilege)

Cette classification permet de faire correspondre les attaques à des notions communes au monde de l'embarqué et de l'informatique classique.

Finalement, nous pouvons conclure que bien que divers modèles d'attaques existent dans la littérature, la majorité se basent sur les mêmes critères, que nous résumons ici :

- Attaquant
- Vecteur d'attaque
- Cible
- Objectif
- Conséquence
- Limitations

A l'aide de ces six critères, nous pouvons établir une classification satisfaisante des attaques.

2.2.3 Classification des IDS

Après avoir analysé les risques associés aux attaques possibles sur notre système, nous allons procéder à un bref état de l'art des méthodes de classification de notre solution proposée : les IDS.

Dans le mémoire de Shalaka [32], une classification est suggérée. Tout d'abord, les IDS sont divisés en 3 catégories :

- IDS basé sur les anomalies (Anomaly-based IDS)
- IDS basé sur les signatures (Signature-based IDS)
- IDS composés (Compound IDS)

Ensuite, la catégorie "IDS basés sur les anomalies" est divisée en sous-catégories :

- IDS par apprentissage autonome (Self-learning IDS)
- IDS programmés (Programmable IDS)

La première catégorie concerne les IDS basés sur les technologies telles que l'apprentissage machine ou l'intelligence artificielle, tandis que les seconds représentent les IDS "traditionnels" qui sont développés à l'aide des connaissances d'experts du domaine. Pour les IDS basés sur les signatures, 3 exemples sont donnés : Snort, Suricata et Bro IDS. Ces IDS sont open-source, et bien connus dans le milieu de la cyber-sécurité, ils ont été éprouvés dans de nombreux cas d'applications, ce qui leur permet d'avoir une base de données de signatures d'attaques volumineuse, ce qui implique donc de bonnes performances de détection. Ensuite, l'article propose de caractériser un IDS selon plusieurs critères :

- Temps de détection (Time of detection)
  - Temps réel (Real-time)
  - Temps compensé (Non-Real-Time)

- Granularité de la détection (Granularity of Detection)
- Source des données (Source of audit data)
- Réponse (Response)
  - Passif (Passive)
  - Actif (Active)
- Lieu du traitement des données (Locus of Data Processing)
- Sécurité (Security)
- Degré d'interopérabilité (Degree of inter-operability)

L'ensemble de ces caractéristiques nous permettent d'obtenir une classification très précise des IDS. Tout d'abord, le temps de détection est un critère primordial puisqu'il définit si l'IDS permet une réaction en temps réel ou bien s'il sert à des analyses en temps compensé, ce qui ne permet aucune réaction à l'attaque. Ensuite, la granularité de la détection va nous permettre d'établir à quel niveau l'IDS effectue ses analyses, par exemple les messages individuels envoyés sur un réseau, ou l'état de ce réseau dans sa globalité. La source des données est également essentielle, puisque c'est ce critère qui différencie les IDS réseau des IDS hôtes par exemple. Mais nous pouvons également donner plus de détails, en indiquant par exemple quels champs d'une trame sont utilisés. La réponse différencie les IDS classiques des systèmes de détection et de prévention d'intrusion, qui peuvent prendre des décisions pour mitiger l'attaque. Le lieu de traitement des données est également primordial, notamment dans un contexte embarqué, puisqu'il indique si l'IDS est autonome ou s'il nécessite le support d'un autre service pour traiter les données, comme par exemple un serveur. Cela a également une incidence sur la notion de confidentialité des données. La sécurité d'un IDS définit les mesures prises lors du développement de celui-ci afin de le protéger lui-même contre des attaques potentielles. Enfin, le degré d'interopérabilité indique la capacité de l'IDS à collaborer avec d'autres IDS. C'est souvent nécessaire lorsque l'IDS doit couvrir une grande surface, auquel cas nous pouvons être amené à diviser la surface en plusieurs parcelles, chacune surveillée par un IDS différent. En complément de cette classification détaillée, l'article propose également de catégoriser les IDS basés sur l'apprentissage machine en plusieurs sous catégories dépendamment des algorithmes utilisés. Toutefois puisque ce type d'IDS n'est pas le sujet de ce travail, nous nous contenterons de les laisser dans une catégorie commune pour la suite du travail, et nous ne détaillerons pas ici les différentes sous-catégories

Qayyum et al. [34] proposent une taxonomie des techniques de détection d'anomalies basée sur l'analyse statistique pour les IDS. Dans un premier temps, les IDS sont classifiés selon deux critères classiques donnés plus haut : la source des données et la réponse. Le troisième critère utilisé concerne la méthode d'analyse, leur article portant sur les IDS basés sur les

anomalies, cette sous-catégorie est détaillée. Les anomalies sont divisées en 3 sous-catégories :

— Basé sur les statistiques (Statistical-based)
— Basé sur la spécification (Specification-based)
— Basé sur les chaînes (String-based)

Les IDS basés sur la spécification utilisent une logique formelle pour établir une norme, selon laquelle l'IDS analysera les données afin de les définir comme normales ou non. Les IDS basés sur les chaînes utilisent une suite d'évènement comme par exemple des appels systèmes afin de définir si le comportement est normal ou non. Finalement, les IDS basés sur les statistiques se basent sur des propriétés statistiques du dispositif surveillé afin d'établir un profil de comportement normal, et déclenchera une alerte si la donnée analysée ne rentre pas dans ce profil. Pour différencier les différentes analyses statistiques possibles, les auteurs proposent 5 sous-catégories :

— Par seuil (Threshold Metric)
— Processus de Markov (Markov Process Model)
— Moments Statistiques (Statistical Moments)
— Modèle Multivarié (Multivariate Model)
— Modèle par séries chronologiques (Time Series Model)

La détection par seuil consiste à définir des seuils de détection et à les comparer avec les mesures prises. Une fenêtre de fonctionnement normal est définie à la programmation, et n'est plus modifiée ensuite. La détection par processus de Markov consiste à analyser le système à des intervalles réguliers et à construire ainsi un processus de Markov avec les états analysés au fur et à mesure. Chaque prochain état possible se voit ensuite associé à une probabilité, et si le système arrive dans un état de faible probabilité, alors une alerte est levée. La détection par moments statistiques se fait en analysant des métriques telles que la moyenne, la déviation standard et autres métriques classiques. Lorsqu'une nouvelle métrique est analysée, elle est comparée à une fenêtre acceptable définie au préalable pour lever ou non une alerte. La méthode est similaire à la détection par seuil, mais cette fois la fenêtre acceptable est mise à jour de manière dynamique avec l'état du système. La détection par modèle multivarié est similaire aux moments statistiques, mais cette fois-ci plusieurs métriques devant être corrélées sont prises en compte, pour une détection plus précise. La détection par séries chronologiques utilise un intervallomètre ainsi qu'une métrique comme un compteur d'évènements, et prend en compte, en plus des valeurs des variables observées, leur ordre et leur périodicité, afin de donner une probabilité d'occurrence. Si cette probabilité est trop faible, alors une alerte est levée. Finalement, nous pouvons voir que cette classification centrée sur les analyses statistiques permet une analyse détaillée de cette famille d'IDS et

permet donc une plus fine granularité pour la distinction des IDS.

Grâce à ces deux méthodes de classification des IDS, nous pouvons établir une classification hybride pour la suite de nos travaux, prenant en compte les critères donnés en figure 2.2.

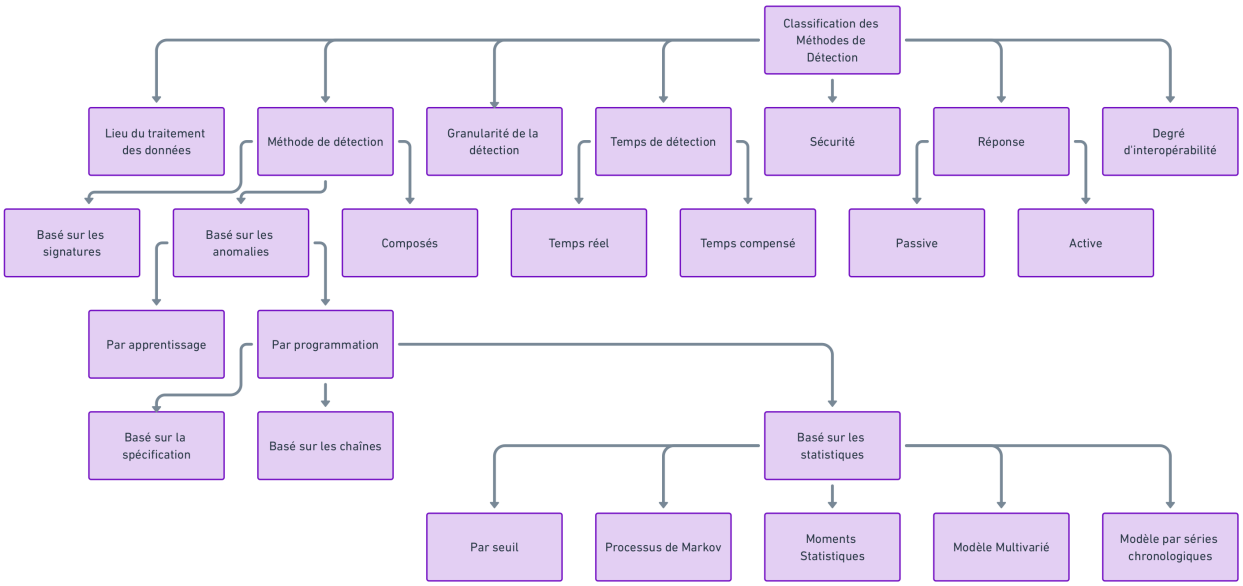

FIGURE 2.2 Classification des IDS.

Ces critères permettront une classification précise, notamment pour les IDS basés sur les anomalies statistiques.

2.3 CANBus

2.3.1 Attaques

Une fois nos critères de classification des attaques établis, nous allons pouvoir dresser une table des attaques connues, et utiliser pour chacune d'entre elles l'outil d'analyse de risque VeRA. Ainsi, nous obtiendrons une hiérarchie des attaques possibles, qui dépendra d'un ensemble de critère liés à la sécurité informatique mais également du niveau d'autonomie du véhicule.

Cui et al. [35] ont établi un état de l'art des vulnérabilités liées aux véhicules autonomes, basé sur les réseaux VANETs, la manipulation de leur environnement, ainsi que les menaces internes au véhicules. Dans leur article, ils ont pu démontrer la possibilité d'effectuer de l'écoute sur le bus CAN ainsi que des injections de trames. De plus, ils ont montré que les réseaux de communication du véhicule avec son environnement permettent un accès plus simple, notamment à distance.

Islam et al. [36] ont proposé une analyse de risque pour les systèmes embarqués automobiles, et parmi les menaces considérées, ils ont exploité le bus CAN afin de modifier le comportement du limiteur de vitesse.

Rizvi et al. [37] ont mis en place deux attaques sur le bus CAN, une de déni de service, et une de rejeu. Ils ont également souligné dans leur article le risque que représente le port de diagnostique OBD qui est directement connecté par le bus CAN aux ECU les plus critiques. Ils conseillent la mise en place de pare-feu pour limiter les risques.

Woo et al. [38] ont utilisé ce même port pour montrer la possibilité d'effectuer des attaques à distance sur les véhicules, à condition d'insérer dans ce port un dispositif compromis par l'attaquant. Ce cas précis sert à illustrer les risques liés à la personnalisation des véhicules par leurs propriétaires, qui peuvent utiliser des outils de tierce partie contenant des risques. Dans leur article, les auteurs utilisent un outil de diagnostique légitime, qui peut donc envoyer des trames de diagnostique protégées. Ainsi, ils ont pu effectuer diverses attaques, en commençant par de l'écoute passive, puis en injectant divers messages, leur permettant de perturber l'affichage, tourner le volant, accélérer, ou encore éteindre le moteur.

Pan et al. [39] ont proposé une analyse des attaques possibles sur un véhicule moderne, et ont montré la possibilité de prendre le contrôle des freins et des clignotants, de modifier la vitesse affichée, de limiter la rotation du volant ou encore l'activation des freins et la coupure du moteur.

Checkoway et al. [40] ont fait l'expérience de hacker un véhicule par le biais du système d'infodivertissement, en particulier trois fonctionnalités : le lecteur CD, le Bluetooth, et le système d'appel mains libres. A l'aide de ces points d'entrée, les chercheurs ont montré qu'ils arrivaient à obtenir un accès au bus CAN leur permettant d'injecter n'importe quelle trame souhaitée. Il est intéressant de remarquer que leur attaque leur a permis de déverrouiller, démarrer et conduire une voiture sans les clés, avec un fonctionnement normal.

Liu et al. [41] ont dressé un état de l'art des attaques possibles afin d'établir les défis à venir et proposer des mitigations. Ils ont trouvé des possibilités d'écoute passive, et de falsification de trames CAN grâce au port OBD. Il ont également découvert qu'à l'aide d'un ECU compromis, il est possible d'injecter ou rejouer n'importe quelle trame, et même de mettre en place un déni de service.

Cho et al. [42] ont démontré la possibilité de déconnecter un ECU du bus, en injectant au même moment un paquet quasi-identique, mais en changeant un bit de celui-ci lors de l'envoi (grâce à l'usage de bits récessifs). Ainsi, l'ECU victime pense que son message a été mal transmis, et incrémente son compteur d'erreur pour chaque message manipulé. En envoyant

suffisamment de messages de ce type, le compteur d'erreur de l'ECU cible dépasse le seuil maximal, et celui-ci se déconnecte donc automatiquement du bus, ce qui provoque une perte de fonctionnalité.

Ken Tindell [43] a publié un article afin de montrer le risque d'une inversion de priorité dans le protocole CAN, pour certains ECU. Si les ECUs ne sont pas équipés d'un système de priorisation des trames en attente d'envoi, ils pourraient se retrouver en attente d'envoyer une trame peu prioritaire, pendant qu'une autre plus prioritaire est bloquée derrière. C'est un dysfonctionnement, qui pourrait être exploité par un attaquant, par exemple dans le cas d'un gateway, en remplissant le buffer de messages peu prioritaires.

Carsten et al. [44] ont dressé une liste des catégories d'attaques possibles pour les véhicules, dont l'écoute passive, la prise de contrôle du véhicule, l'envoi de fausses informations au conducteur et aux autres usagers, ou encore la possibilité de dégrader le véhicule, en injectant des petites modifications, potentiellement imperceptibles pour le conducteur.

Lee et al. [45] ont mis en place une attaque de fuzzing sur le bus CAN. Ils ont vérifié leur attaque sur 3 véhicules milieu de gamme populaires dans les années 2010, sans préciser lesquels, et ont obtenu des résultats concluants.

Chris Valasek et Charlie Miller [46], [8, 47], ont effectué plusieurs travaux d'analyse en profondeur des attaques possibles sur des véhicules modernes. Le premier [46] consiste à trouver et exploiter des vulnérabilités dans les réseaux véhiculaires. Ils ont utilisé pour cela 3 vecteurs d'attaques, le port OBD, un ECU piraté, et le port OBD avec un outil de diagnostique, afin de pouvoir envoyer d'autres types de trames. Ils ont ainsi pu mettre en place de l'injection de messages, un déni de service, mais aussi la reprogrammation de certains ECU (Code injection). Ils ont par ces méthodes pu modifier la majorité des données affichées au conducteur et aux autres usagers (vitesse, clignotant, kilométrage...), mais aussi prendre le contrôle des fonctionnalités essentielles du véhicule (accélérateur, volant...) grâce à l'outil de diagnostic. Ils n'ont pas implémenté d'attaques concrètes utilisant la mise à jour des ECUs, mais ceci représente une grande menace puisque l'attaquant aurait tous les droits sur ce dispositif. Dans leur seconde analyse [8, 47], ils se sont intéressés à la possibilité de contrôler le véhicule à distance. Pour cela, ils ont piraté le système Uconnect (système d'infodivertissement très commun) d'une Jeep Cherokee de 2014, et ils l'ont utilisé pour envoyer des trames sur le bus CAN. A l'aide de leurs précédentes études des vulnérabilités automobiles, ils ont pu effectuer diverses attaques pour prendre le contrôle du véhicule, en particulier grâce au reverse engineering des outils de diagnostiques propriétaires des constructeurs. Ils ont donc pu prendre le contrôle total d'un véhicule, à condition que celui-ci se déplace à faible allure, sans quoi les ECUs ignorent les messages de diagnostique.

Nie et al. [48] ont piraté une Tesla Model S, le modèle de voiture autonome le plus commun, et réputé pour être l'un des plus sécurisés. Ils ont réussi grâce au navigateur internet, à prendre le contrôle de divers composants de la voiture jusqu'à atteindre le bus CAN. Ensuite, ils ont montré la possibilité de mettre à jour le firmware d'un ECU, d'injecter des trames, mais aussi la possibilité d'empêcher l'envoi de trames par un ECU, en le passant en mode diagnostique. Une dernière attaque novatrice a été reportée, qui consiste à empêcher le passage de trames d'un sous-réseau vers un autre, grâce au contrôle du gateway piraté.

Les mêmes auteurs [49] ont montré que la connectivité Bluetooth des véhicules modernes permet d'en prendre le contrôle en envoyant tous types de messages CAN, y compris des messages de diagnostique. Ils ont effectué la démonstration de cette attaque sur une Lexus NX300.

Enfin, Rogers et al. [50] ont récemment montré la possibilité de pirater une Tesla Model S, grâce à un port Ethernet laissé accessible. Ils ont pu prendre le contrôle de l'ensemble du véhicule par une méthode similaire à celle de Nie et al. [48], malgré de nombreuses difficultés d'exploitation, qui montrent une amélioration de la sécurité de ce type de véhicule. Ils ont notamment félicité le travail effectué sur l'architecture du véhicule, qui sépare le réseau Ethernet du bus CAN, dont seul un gateway sécurisé permet l'accès.

En complément de ces articles, nous avons identifié une base de données qui recensent les dernières attaques identifiées sur les véhicule [51], dont certaines concernent le bus CAN. Ces attaques ne sont pas renseignées en détail dans la version gratuite, mais elles permettent d'avoir une idée de la variété des risques existants aujourd'hui.

L'ensemble des attaques considérées dans cet état de l'art a été résumé en figure 2.1, en incluant les critères présentés plus haut.

Notons que le facteur H a été fixé à 2 pour tous les articles ne précisant pas le véhicule utilisé, ce qui correspond à un conducteur moyennement expérimenté avec une voiture équipée d'aides à la conduite simples. De plus, l'attaquant n'étant pas précisé dans les articles, ce critère ne sera pas considéré.

2.3.2 Défenses

Maintenant que nous avons vu quelles étaient les attaques recensées sur le bus CAN et les risques liés à celles-ci, nous allons dresser un état de l'art des défenses existantes. L'analyse précédente nous a montré que la grande majorité des attaques à haut risque sont liées à l'injection de messages sur le bus. La meilleure solution pour empêcher ces attaques semble naturellement être de mettre en place un mécanisme d'authentification, afin d'autoriser les

Tableau 2.1 Classification des attaques recensées sur le bus CAN

Article	Attaque	Vecteur d'attaque	Cible	Objectif	Conséquences	Limitations	Risque			
							H	S	P	R
Cui et al. [35]	Message Injection	CANBus	ECU	Contrôler le véhicule	Variable selon les trames injectées	Besoin d'accès au bus	2	3	2	8
	Eavesdropping	ECU compromis	CANBus	Obtenir de l'information sur le véhicule	/	Besoin d'accès au bus	2	1	3	5
Islam et al. [36]	Message Injection	Port OBD	ECU Limiteur de vitesse	Modifier le comportement du limiteur de vitesse	Arrêt du véhicule	Sécurité du port OBD	2	2	3	8
Rizvi et al. [37]	DoS	CANBus	Bus CAN	Bloquer la transmission de données CAN	Difficulté a conduire le véhicule	Besoin d'une grosse capacité d'envoi	2	3	2	8
	Replay	CANBus	ECU	Rejouer un comportement du véhicule	Variabe selon la trame rejouée	Besoin de capturer les trames	2	2	2	6
Woo et al. [38]	Eavesdropping	Port OBD	CAN Bus	Obtenir de l'information sur le véhicule	Aucune	Sécurité du port OBD	2	0	3	2
	Message Injection	Port OBD	ECU Affichage	Contrôle a distance du véhicule	Déformation de l'affichage	Sécurité du port OBD	2	1	1	3
	Message Injection	Port OBD	ECU Moteur	Contrôle a distance du véhicule	Arrêt moteur	Sécurité du port OBD	2	2	1	4
	Message Injection	Port OBD	ECU Volant	Contrôle a distance du véhicule	Contrôle de la direction	Sécurité du port OBD	2	3	1	5
	Message Injection	Port OBD	ECU Accélérateur	Contrôle a distance du véhicule	Accélération	Sécurité du port OBD	2	3	1	5
Pan et al. [39]	Flooding	CANBus	ECU	Arrêter le moteur	Arrêt du véhicule	Besoin d'accès au bus	2	3	1	5
	Message Injection	CANBus	ECU	Modifier la vitesse affichée	Mauvaise information pour le conducteur	Besoin d'accès au bus	2	1	2	4
	DoS	CANBus	CANBus	Limiter la rotation du volant	Difficulté a conduire le véhicule	Besoin d'une grosse capacité d'envoi	2	2	2	6
	Message Injection	CANBus	ECU	Activer les freins	Arrêt du véhicule	Besoin d'accès au bus	2	3	1	5
	Message Injection	CANBus	ECU	Manipuler les clignotants	Mauvaise information pour les autres usagers	Besoin d'accès au bus	2	2	2	6
Checkoway et al. [40]	All possible	Lecteur CD	CANBus	Prendre le contrôle du véhicule	Variable selon les trames injectées	Créer un CD, et contourner les sécurités	2	3	1	5
	All possible	Bluetooth	CANBus	Prendre le contrôle du véhicule	Variable selon les trames injectées	Avoir un appareil de confiance ou contourner la sécurité	2	3	1	5
	All possible	Système d'appel	CANBus	Prendre le contrôle du véhicule	Variable selon les trames injectées	Accès au réseau cellulaire et pouvoir joindre le véhicule	2	3	1	5
Liu et al. [41]	Eavesdropping	OBD/Telematics	CANBus	Obtenir de l'information sur le véhicule	/	Sécurité du port OBD	2	1	2	4
	CAN Frame Falsifying	OBD/Telematics	CANBus	Créer de faux paquets à injecter	/	Sécurité du port OBD	2	0	3	2
	Message Injection	ECU compromis	ECU	Contrôler le véhicule	Variable selon les trames injectées	Besoin d'accès au bus	2	3	2	8
	Replay	ECU compromis	ECU	Rejouer un comportement du véhicule	Variable selon les trames injectées	Besoin d'accès au bus	2	3	2	8
	DoS	ECU compromis	CANBus	Bloquer la transmission de données CAN	Difficulté a conduire le véhicule	Besoin d'accès au bus	2	2	1	4
Cho et al. [42]	Message Injection	ECU malicieux ajouté	ECU	Déconnecter un ECU du bus CAN	Coupure d'une fonctionnalité	Connaissance parfaite de l'ECU victime	2	3	1	5
Ken Tindell [43]	Buffer manipulation	ECU	ECU	Retarder l'envoi de trames prioritaires	Augmentation de la latence	Besoin d'accès a l'ECU cible	2	2	1	4
Carsten et al. [44]	Eavesdropping	Port OBD / ECU compromis / Accès distant	CAN Bus	Obtenir de l'information sur le véhicule	/	Besoin d'accès au bus / Procédures de sécurité pour accès distant	2	1	3	5
	Message Injection	Port OBD / ECU compromis / Accès distant	ECU	Contrôler le véhicule	Variable selon les trames injectées	Besoin d'accès au bus / Procédures de sécurité pour accès distant	2	3	2	8
	Message Injection	Port OBD / ECU compromis / Accès distant	ECU	Dégradation du véhicule	Baisse de performance	Besoin d'accès au bus / Procédures de sécurité pour accès distant	2	2	2	6
	Message Injection	Port OBD / ECU compromis / Accès distant	ECU	Falsification des données	Mauvaise information pour le conducteur	Besoin d'accès au bus / Procédures de sécurité pour accès distant	2	1	2	4
Lee et al. [45]	Fuzzing	Port OBD	ECU	Comprendre les effets de différentes trames	Malfonctionnement de certains services	Sécurité du port OBD	2	2	3	8
Valasek et Miller [46]	Message Injection	Port OBD	ECU Affichage	Afficher une fausse vitesse	Mauvaise information pour le conducteur	Sécurité du port OBD	1	1	2	3
	Message Injection	Port OBD	ECU Odomètre	Augmenter l'odomètre	Mauvaise information pour le conducteur	Sécurité du port OBD	1	0	2	1
	Message Injection	Port OBD	ECU Guidage écran	Afficher une fausse position sur le GPS	Mauvaise information pour le conducteur	Sécurité du port OBD	1	1	2	3
	DoS	Port OBD	ECU Volant	Limiter la capacité a tourner	Difficulté a conduire le véhicule	Sécurité du port OBD	1	2	2	5
	Message Injection	Port OBD	ECU Volant	Faire tourner les roues	Contrôle de la direction du véhicule	Sécurité du port OBD / Synchronisation de plusieurs messages	1	3	2	7
	Message Injection	Port OBD	ECU Frein	Freiner le véhicule	Arrêt du véhicule	Sécurité du port OBD / Compteur de messages	1	2	2	5
	Message Injection	ECU Compromis	ECU Accélérateur	Accélérer le véhicule	Accélération temporaire du véhicule	Partiellement fonctionnel	1	3	1	4
	Message Injection	Outil de diagnostic via Port OBD	ECU Frein	Activer les freins	Arrêt du véhicule	Sécurité du port OBD / Reverse Engineering du logiciel Ford-Toyota	1	2	1	3
	Message Injection	Outil de diagnostic via Port OBD	ECU Frein	Désactiver les freins	Impossible d'arrêter le véhicule	Sécurité du port OBD / Reverse Engineering du logiciel Ford-Toyota	1	3	1	4
	Message Injection	Outil de diagnostic via Port OBD	ECU SJB (Lumière, Radio etc.)	Éteindre les lumières	Mauvaise information pour les autres usagers	Sécurité du port OBD / Reverse Engineering du logiciel Ford-Toyota	1	1	1	2
	Message Injection	Outil de diagnostic via Port OBD	ECU moteur	Couper le moteur	Arrêt du véhicule	Sécurité du port OBD / Reverse Engineering du logiciel Ford-Toyota	1	2	1	3
	Message Injection	Outil de diagnostic via Port OBD	ECU SJB (Lumière, Radio etc.)	Clignoter les lumières	Mauvaise information pour les autres usagers	Sécurité du port OBD / Reverse Engineering du logiciel Ford-Toyota	1	1	1	2
	Message Injection	Outil de diagnostic via Port OBD	ECU Klaxon	Activer le klaxon	Mauvaise information pour les autres usagers	Sécurité du port OBD / Reverse Engineering du logiciel Toyota	1	1	1	2
	Message Injection	Outil de diagnostic via Port OBD	ECU Ceinture	Serrer la ceinture de sécurité	Inconfort conducteur	Sécurité du port OBD / Reverse Engineering du logiciel Toyota	1	1	1	2
	Message Injection	Outil de diagnostic via Port OBD	ECU Portes	Ouverture/Fermeture des portes	Désactive la sécurité enfant par exemple	Sécurité du port OBD / Reverse Engineering du logiciel Toyota	1	1	1	2
	Message Injection	Outil de diagnostic via Port OBD	ECU Jauge essence	Corrompre le niveau d'essence	Mauvaise information pour le conducteur	Sécurité du port OBD / Reverse Engineering du logiciel Toyota	1	1	1	2
	Malicious Code Injection	Outil de diagnostic via Port OBD	ECU	Modifier le firmware d'un ECU	Variable selon le code injecté et l'ECU visé	Sécurité du port OBD / Reverse Engineering du logiciel Ford	1	3	1	4
	Malicious Code Injection	Outil de diagnostic via Port OBD	ECU	Modifier le code de l'ECM	Variable selon le code injecté	Sécurité du port OBD / Reverse Engineering du logiciel Toyota	1	3	1	4
Valasek et Miller [8, 47]	Message Injection	Système multimédia (Uconnect)	ECU Clignotants	Activer les clignotants	Mauvaise information pour les autres usagers	Hack Uconnect	2	2	2	6
	Message Injection	Système multimédia (Uconnect)	ECU Portes	Verrouiller/Déverrouiller le véhicule	Désactive la sécurité enfant par exemple	Hack Uconnect	2	1	2	4
	Message Injection	Système multimédia (Uconnect)	ECU Affichage	Modifier les RPMs du véhicule	Mauvaise information pour le conducteur	Hack Uconnect	2	1	2	4
	Message Injection	Outil de diagnostic via Port OBD	ECU Moteur	Couper le moteur	Arrêt du véhicule	Accès aux outils de diagnostique fabriquant / Basse vitesse seulement	2	3	1	5
	Message Injection	Outil de diagnostic via Port OBD	ECU Frein	Couper les freins	Impossible d'arrêter le véhicule	Accès aux outils de diagnostique fabriquant / Basse vitesse seulement	2	3	1	5
	Message Injection	Outil de diagnostic via Port OBD	ECU Volant	Tourner le volant	Contrôle de la direction du véhicule	Accès aux outils de diagnostique fabriquant / Basse vitesse seulement	2	3	1	5
Nie et al. [48]	Malicious Code Injection	Web Browser	ECU	Modifier le firmware d'un ECU	Variable selon le code injecté et l'ECU visé	Connaissance avancée des logiciels du fabricant / Élévation de privilèges	3	3	1	6
	Message Injection	Web Browser	ECU	Contrôler le véhicule	Variable selon les trames injectées	Connaissance avancée des logiciels du fabricant / Élévation de privilèges	3	3	1	6
	Message Blocking	Web Browser	CANBus	Empêcher les trames de passer d'un bus CAN a un autre	Coupure d'une fonctionnalité	Connaissance avancée des logiciels du fabricant / Élévation de privilèges	3	2	1	5
	Suspend/Disable ECU	Web Browser	ECU	Mettre un ECU en mode diagnostique pour qu'il n'envoie plus de trames	Coupure d'une fonctionnalité	Connaissance avancée des logiciels du fabricant / Élévation de privilèges	3	3	1	6
Nie et al. [49]	Message Injection	Bluetooth	ECU	Contrôler le véhicule	Variable selon les trames injectées	Hack du système Bluetooth par BufferOverflow	2	3	2	8
Rogers et al. [50]	Message Injection	Port Ethernet accessible	ECU	Contrôler le véhicule	Variable selon les trames injectées	Reverse Engineering / Escalade de privilèges	3	3	1	6

messages risqués seulement s'ils viennent du nœud autorisé à les envoyer. Nous commencerons donc par analyser quelques mécanismes de ce type proposés dans la littérature.

Authentification

Radu et al. [52] ont proposé LeiA, un protocole d'authentification léger, qui se base sur une clé de session dérivée d'une clé symétrique, qui sera mise à jour à chaque démarrage du véhicule. En complément, des compteurs sont mis en place fin de valider la fraîcheur de chaque message et donc prévenir les attaques par rejeu. Les informations sont contenues dans les 18 bits des ID CAN étendu ce qui évite d'avoir besoin de revoir le protocole. Ce champ est utilisé de base par certains systèmes qui ne pourront donc pas utiliser cette technologie, mais la grande majorité des véhicules personnels n'en font pas usage, bien que les composants soient compatibles. Les auteurs ont prouvé mathématiquement la sécurité du protocole.

Wang et al. [53] ont proposé VeCure pour authentifier les messages passant entre un bus basse sécurité et un bus haute sécurité. Ainsi, chaque message de données sera suivi d'un message d'authentification, qui contient l'ID de nœud envoyeur, un compteur de message, un MAC, et un marqueur d'authentification le liant au message de données. Ils bloquent ainsi l'injection de nouveaux messages sur le bus, et compliquent grandement la mise en place d'une attaque par rejeu, et ce pour un coût temporel de 50 µs.

Mundhenk et al. [54,55] ont proposé LASAN, un mécanisme se basant sur une authentification des ECUs au démarrage du véhicule, à l'aide de cryptographie asymétrique, puis à une autorisation des échanges en temps réel par cryptographie symétrique. Chaque ECU nécessite ainsi un certificat, ce qui peut être coûteux à mettre en place, mais le protocole protège de nombreuses attaques, et permettrait notamment de sécuriser les mise à jour Over-The-Air (OTA).

Luo et al. [56] ont choisi d'exploiter le champ CRC du protocole pour y inclure leur MAC d'authentification. Leur système se base sur un gateway qui partage des clés aux ECUs, et les met régulièrement à jour, notamment lorsque des ECUs sont ajoutés ou retirés du véhicule. Pour calculer le MAC des messages, les ECUs font appel à leur horloge interne, afin d'éviter toute désynchronisation des compteurs de message en cas de problème. Cette méthode permet également de décoder les MAC qui sont limités à 15 bits, et leur analyse montre que cela bloque toute attaque par rejeu, tout en complexifiant grandement la possibilité d'injecter des messages par le calcul du MAC.

Groza et al. [57] ont proposé LiBrA-CAN, qui se base cette fois sur un Mixed-MAC (M-MAC) afin de pouvoir être vérifié par les clés de tous les récepteurs. Ce système nécessite une étape

de distribution des clés pour chaque nœud, les nœuds sont rassemblés en groupes de communication et partagent donc leurs clés au sein du groupe. Les auteurs ont également proposé deux variantes, une distribuée, et une autre orientée maître afin de résoudre le problème du coût computationnel. Ce système a été vérifié sur différents banc d'essais avec succès.

Herrewege et al. [58] ont proposé une solution utilisant CAN+ afin d'agrandir le champ des données du message et d'y inclure un compteur de messages et un Hash-based-MAC (HMAC), afin de limiter le coût computationnel pour chaque MAC. Ainsi, leur système bloque tout attaque par rejeu et permet de sécuriser le protocole à faible coût, à condition d'utiliser CAN+ pour transmettre les informations de sécurité également.

Nous voyons donc qu'une variété de protocoles d'authentification ont été proposés par la littérature, mais notre recherche a montré que ces protocoles ne sont pas en réalité appliqués par l'industrie. Ces protocoles nécessitent souvent l'usage de variantes de CAN, comme CAN-FD ou CAN+, or les constructeurs utilisent encore largement le protocole classique. De plus, même si le protocole classique est suffisant, ces fonctionnalités impliquent souvent d'avoir une puissance de calcul plus élevée, qui se traduit par un coût matériel plus élevé pour les fabricants, et comme ces composants sont produits en très grande quantité, même quelques centimes de différence représentent finalement des millions d'euros pour le fabricant. Enfin, ces protocoles nécessitent forcément une mise à jour du code des ECUs, ce qui est complexe à mettre en place, notamment si aucun protocole n'est universellement adopté par les acteurs de l'industrie.

Autres

En complément des protocoles d'authentification, la littérature a proposé d'autres approches de sécurisation.

L'usage d'honeypots [1] a été étudié par Verendel et al. [59]. Ils ont proposé trois architectures possibles, qui se basent chacune sur les retours du conducteur et les entrées de l'environnement pour simuler un réseau interne du véhicule, avec des retours du simulateur en entrée, ou non. Cette approche semble intéressante, mais contient quelques limitations. Le honeypot doit être réaliste afin que l'attaquant ne soit pas conscient qu'il ne manipule pas le vrai véhicule, mais ceci peut être coûteux en ressources, ce qui est critique dans le contexte embarqué. De plus, les données d'entrée doivent être collectées dans le véhicule d'une bonne manière, afin d'éviter tout impact du honeypot sur le système réel. Cependant, cette approche permettrait de réunir de nombreuses données sur des attaques pour les analyser ensuite, ce qui pourrait

1. Honeypots : Leurres informatiques copiant le réseau cible afin de contenir les attaques dans un environnement non critique.

permettre de faire progresser le domaine.

Un autre approche pour confiner l'attaquant est proposée par Muhammad et al. [60], en se basant sur une architecture spécifique du bus CAN. Ils proposent ainsi de mettre en place une topologie en étoile afin d'intégrer un nœud maître, qui confinera certains bus ou nœuds du reste du réseau en cas d'erreur détectée. Le nœud maître met en place une horloge partagée entre les nœuds, afin de détecter les problèmes, en envoyant des messages périodiques que chaque nœud doit reconnaître. Ce système est intéressant mais nécessite un gros changement d'architecture interne, et décuple les communications au sein du bus.

Enfin, Wael A. Farag [61] ont proposé CANTrack, un Computer Assisted Design (CAD) Tool. Leur système propose d'intégrer les fonctionnalités permettant une étude de sécurité du véhicule, notamment du bus CAN, en proposant par exemple de la cryptographie ou de l'authentification.

IDS

En complément des défenses cryptographiques, il est nécessaire de détecter des potentielles anomalies de comportement en temps réel, pour se prémunir des attaques non mitigées par celles-ci. Pour cela, nous allons analyser plusieurs IDS, aux architectures diverses.

Tout d'abord, un IDS par signature a été proposé par Zhang et al. [62], dans le but de détecter des ECUs manipulés, qui contiendraient des malware. Pour cela, chaque programme qui s'exécute ou qui est reçu par une mise à jour est analysé localement, en comparant sa signature avec une base de données locale, et avec une whitelist des programmes vérifiés. Si ces critères ne sont pas vérifiés, ou si le programme concerne des composants critiques, alors celui-ci est envoyé à un serveur distant qui procède à une vérification du code. Ainsi, nous sommes sûrs que chacun des programmes est vérifié par une entité de confiance et n'est donc pas dangereux, sans quoi il est blacklisté et bloqué.

Jin et al. [63] ont proposé un IDS par signature, analysant les IDs et les données pour détecter des comportements malicieux. Pour chacun de ces champs, une fenêtre de validité est établie au préalable, et une alerte est levée si le message reçu est en dehors. Ce fonctionnement peu coûteux peut donc être effectué en local. Cependant, ce système ne permet pas de se protéger d'une attaque bien construite par un attaquant informé. Ils introduisent donc en complément un IDS par anomalie basé sur les temps d'arrivée des messages. Un modèle statistique est établi, et on en déduit un modèle de prédiction, qui nous permettra de lever une alerte si on dépasse le pire cas prédit. Cette 2e analyse protège des injections de messages bien formés.

Khan et al. [64] ont proposé une autre architecture par anomalie, qui se base sur l'analyse

statistique (ID moyen, déviation standard, etc.) d'une fenêtre de messages, et qui compare cette analyse aux données historiques. Le comportement prévisible du bus CAN rend la détection possible, l'expérimentation montre un score F1 de 1.

Lee et al. [65] ont proposé une autre approche, dans laquelle chaque nœud récepteur doit valider en envoyant un accusé de réception. Leur analyse montre qu'en temps normal, le délai de réponse est très peu variable, contrairement au cas sous-attaque, ce qui leur permet de lever les alertes avec une bonne performance.

Larson et al. [66] ont évalué une approche par spécification en se basant sur CANopen. Ils analysent les données des messages et les vérifient en les comparant à la spécification et en faisant des analyses croisées. De plus, ils établissent des taux d'envoi/réception pour chaque ECU. Ce système est efficace, mais se limite aux systèmes utilisant CANopen. Weber et al. [67] ont suivi une approche similaire mais en suivant la spécification du véhicule plutôt que du protocole, en vérifiant 8 aspects des messages CAN. Leur IDS est complété par une analyse par ML, qui sera détaillée plus bas dans cette section.

Taylor et al. [68] utilisent une fenêtre glissante afin de détecter des changements dans les périodes de réception des messages. Leur travail a montré que les attaques détectées par cette méthode ne pouvaient pas l'être par une analyse des champs de données, mais leur taux de fausses alertes est encore élevé. Gmiden et al. [69] ont adopté une approche très similaire, mais ils n'ont pas évalué leur performance. Moore et al. [70] ont analysé cette métrique par un algorithme plus complexe, qui se base sur les données historiques pour lever les alertes. Leur système obtient un score de 0.294% de faux positifs et 99,98% de vrais positifs.

Cho et al. [71] ont eux aussi analysé cette période de réception à l'aide d'une somme cumulative afin de lever des alertes pour des attaques de vol d'identité. Leur système est plus précis avec seulement 0.055% de faux positifs.

Müter et al. [72] ont choisi de s'intéresser à l'entropie du bus CAN, afin de tirer profit de sa stabilité. Leur méthode mesure l'entropie du champ ID des messages sur une fenêtre glissante pour lever des alertes. En effet, une attaque modifierait la distribution de ces IDs et donc l'entropie. Leur système ne peut cependant pas détecter des attaques bien construites comme les attaques par mascarade.

Ling et al. [73] se sont aussi basés sur les IDs, et ont établi un taux normal de réception par IDs, si ces taux ne sont pas respectés, une alerte est levée. Ce système tire aussi parti de la stabilité du réseau CAN, mais n'est pas capable de détecter des attaques sophistiquées comme la mascarade.

Ansari et al. [74] se sont intéressés à un IDS distribué sur chaque ECU du réseau. Chaque

nœud écoute le réseau pour recevoir des messages transmis en utilisant leur ID, et s'ils en reçoivent un qu'ils n'ont pas envoyé, ils lèvent une alerte de vol d'identité. Leur système est performant mais nécessite d'implémenter l'IDS sur chaque ECU. Abbott-McCune et al. [75] ont mis en place cette même technique, en ajoutant une détection basée sur la spécification afin de détecter les messages d'IDs non autorisés.

Ces dernières années, l'essor du machine learning a poussé la communauté scientifique à analyser l'applicabilité à divers domaines, dont la détection d'intrusion. Nous allons donc établir un bref état de l'art de ces travaux dans le cas du bus CAN.

Lampe et al. [76] ont proposé d'établir une matrice de transition pour analyser les séquences de messages, et détecter des enchaînements suspicieux. Leur IDS semble lever peu de faux positifs, mais ces métriques ne sont pas données. Marchetti et al. [77] ont opté pour une approche similaire, qu'ils ont évaluée, en terme de détection et de coût en ressources. Leurs résultats montrent un faible besoin matériel, pour une détection pouvant monter jusqu'à 100% selon les scénarios d'attaques. Narayanan et al. [78] ont suivi cette même méthode, mais ils n'ont pas fourni d'analyse de performance, et leur système n'est simulé qu'en logiciel avec Matlab.

Weber et al. [67] ont établi un algorithme de ML qui analyse les suites de valeur de chaque signal, afin d'établir un score d'anomalie. Ils font pour cela une normalisation de la nouvelle valeur, en se basant sur la moyenne et l'écart-type historique, puis ils utilisent leur algorithme pour détecter des valeurs suspectes. Malheureusement, le système n'est pas évalué avec des données autres que celles de l'entraînement, et aucun résultat de performance n'est donné par les auteurs. Wasicek et al. [79] ont adopté une approche similaire pour détecter des ECUs malicieux, qu'ils ont évalué à l'aide d'un simulateur. Leurs résultats sont intéressants, mais un travail d'optimisation est nécessaire pour atteindre un IDS utilisable dans le monde réel. Tomlinson [80] a également établi son IDS selon le même principe, et a atteint une précision de plus de 99%, pour un très faible taux de faux positifs.

Wang et al. [81] ont utilisé la mémoire hiérarchique temporelle (HTM), pour analyser les champs de données de chaque ID, et prédire les séquences de messages. Ils obtiennent des précisions de 98-99% selon les IDs, ce qui semble prometteur.

Zhang et al. [82] ont proposé une approche utilisant une descente de gradient (GDM), mais leur travail ne considère que les attaques par rejeu, ce qui limite son efficacité dans le monde réel.

Moulahi et al. [83] ont comparé une variété d'algorithmes de ML pour la détection d'intrusion, en se basant sur une base de données d'un véhicule réel. Leurs résultats sont prometteurs,

avec une détection en 96,1% et 98,5%, mais leur article ne précise pas les caractéristiques de la base de données utilisée par l'IDS, et aucune implémentation matérielle n'est proposée.

Jun Li [84] propose une approche de ML basée sur la prédiction du comportement physique du véhicule, afin de prédire les futures données transmises sur le bus CAN. En cas de probabilité trop faible, une alerte est levée. L'IDS n'a pas été évalué, mais son modèle de prédiction a été comparé à des données réelles, et celui-ci a montré une très forte similarité.

Enfin, Nowdehi et al. [85] ont adopté une approche similaire afin de mettre en place un IDS capable de fonctionner sur plusieurs architectures différentes. Leur analyse leur a permis de détecter plusieurs attaques différentes, mais les scores exacts ne sont pas communiqués.

Nous pouvons donc voir que la littérature a développé de nombreux IDS basés sur l'apprentissage machine, et que ceux-ci semblent prometteurs. Cependant, nous remarquons que cette technologie nécessite beaucoup de données d'apprentissage, dont nous manquons actuellement. De plus, ces méthodes sont pour la plupart limitées, puisqu'elles ne sont efficaces que pour les architectures sur lesquelles elles ont été entraînées.

Pour conclure notre état de l'art des IDS CAN recensés dans la littérature, nous avons établi une classification, donnée en tableau 2.2.

2.4 Ethernet

Comme nous avons pu le voir dans les sections précédentes, la complexité des véhicules modernes a grandement augmenté ces dernières années avec les aides à la conduite. Avec l'avènement des véhicules autonomes, nous sommes maintenant contraints de faire circuler de grandes quantités d'informations au sein même du véhicule, et ceci ne peut se faire par le biais du bus CAN, dont la bande passant maximale est de $1\,\mathrm{MB\,s^{-1}}$. De nouveaux canaux sont donc intégrés aux véhicules, et parmi eux, Ethernet est en passe de devenir le standard pour ces communications. Dans cette partie, nous proposons donc d'établir un état de l'art des attaques recensées, ainsi que des défenses associées.

2.4.1 Attaques

Les réseaux Ethernet intra-véhiculaires étant encore assez peu développés, les attaques spécifiques à ceux-ci sont encore assez mal connues. Cependant, certains chercheurs se sont proposés de passer en revue les attaques recensées, et susceptibles d'arriver prochainement.

B. C. Douss et al. [86] ont établi un état de l'art des protocoles Ethernet pour l'automobile, avec les vulnérabilités associées. Ils ont inventorié 5 protocoles différents :

Tableau 2.2 Classification des IDS CAN

Article	Source des données	Méthode de détection	Temps de détection	Granularité de la détection	Réponse	Lieu de traitement des données
Zhang et al. [62]	Code des firmwaresDonnées des bus internes	Signature	Temps compensé	Pour chaque firmware	Actif	Local et cloud
Jin et al. [63]	Messages individuels	Signature	Temps réel	Pour chaque message	Passif	Local
	Métriques du bus	Anomalie (Prog)	Temps réel	Pour chaque ID	Passif	Local
Lampe et al. [76]	Séquences de messages	Anomalie (ML)	Temps réel	Pour chaque message	Passif	Local
Khan et al. [64]	Fenêtre de messages	Anomalie (Prog)	Temps réel	Pour chaque suite de message	Passif	Local
Lee et al. [65]	Fenêtre de messages	Anomalie (Prog)	Temps réel	Pour chaque ID	Passif	Local
Larson et al. [66]	ECU	Anomalie (Prog)	Temps réel	Pour chaque ECU	Passif	Local
Taylor et al. [68]	Fenêtre de messages	Anomalie (Prog)	Temps réel	Pour chaque ID	Passif	Local
Weber et al. [67]	Messages individuels	Anomalie (Prog)	Temps réel	Pour chaque message	Passif	Local
	Fenêtre de messages	Anomalie (ML)	Temps réel	Pour chaque ID	Passif	Local
Cho et al. [71]	Séquences de messages	Anomalie (Prog)	Temps réel	Pour chaque ID	Passif	Local
Wang et al. [81]	Fenêtre de messages	Anomalie (ML)	Temps réel	Pour chaque ID	Passif	Local
Müter et al. [72]	Fenêtre de messages	Anomalie (Prog)	Temps réel	Pour chaque ID	Passif	Local
Ling et al. [73]	Messages individuels	Anomalie (Prog)	Temps réel	Pour chaque ID	Passif	Local
Gmiden et al. [69]	Messages individuels	Anomalie (Prog)	Temps réel	Pour chaque ID	Passif	Local
Moore et al. [70]	Fenêtre de messages	Anomalie (Prog)	Temps réel	Pour chaque ID	Passif	Local
Zhang et al. [82]	ECU	Anomalie (ML)	Temps réel	Pour chaque ID	Passif	Local
Wasicek et al. [79]	ECU	Anomalie (ML)	Temps réel	Pour chaque ECU	Passif	Local
	Messages individuels	Anomalie (ML)	Temps réel	Pour chaque ECU	Passif	Local
Marchetti et al. [77]	Séquences de messages	Anomalie (ML)	Temps réel	Pour chaque ID	Passif	Local
Tomlinson [80]	Fenêtre de messages	Anomalie (ML)	Temps réel	Pour chaque ID	Passif	Local
Narayanan et al. [78]	Fenêtre de messages	Anomalie (ML)	Temps réel	Pour chaque message	Passif	Local
Moulahi et al. [83]	Messages individuels	Anomalie (ML)	Temps réel	Pour chaque message	Passif	Local
Ansari et al. [74]	Messages individuels	Anomalie (Prog)	Temps réel	Pour chaque message	Passif	Local
Abbott-McCune et al. [75]	Messages individuels	Anomalie (Prog)	Temps réel	Pour chaque message	Passif	Local
Nowdehi et al. [85]	Messages individuels	Anomalie (ML)	Temps réel	Pour chaque message	Passif	Local
Jun Li [84]	Messages individuels	Anomalie (ML)	Temps réel	Pour chaque message	Passif	Local

— 100-Base-TX
— BroadR-Reach
— Audio Video Bridge-Time-Sensitive Network (AVB/TSN)
— Diagnostics over Internet Protocol (DoIP)
— Scalable Service-Oriented Middleware over Internet Protocol (SOME/IP)

Le premier risque lié à Ethernet selon les auteurs vient de la faible sécurité des switchs. Si un attaquant a un accès physique au switch, il peut aisément rejoindre le réseau, ajouter son propre switch pour connecter son réseau, voire même prendre le contrôle du switch, puisque ceux-ci utilisent souvent des identifiants par défaut. Une fois connecté au réseau, un attaquant pourrait donc faire de nombreuses attaques. Tout d'abord, une attaque passive lui permettrait d'écouter le réseau pour obtenir de l'information sur le véhicule, ou bien sur ses occupants. Les messages n'étant pas diffusés à tous les nœuds, l'écoute est plus complexe que sur le bus CAN, mais un attaquant expérimenté pourrait mettre en place des mécanismes pour contourner cela. Pour les attaques actives, ils ont recensé 9 attaques possibles (les noms ont été traduits et adaptés pour une meilleure compréhension) :

— Injection de trames (Frame Injection)
— Usurpation d'identité (Spoofing Attack)
— Mascarade (Impersonation Attack)
— Déni de service (DoS Attack)
— Empoisonnement du cache ARP (ARP Cache Poisoning)
— Empoisonnement DHCP (DHCP Poisoning)
— MITM (MITM Attack)
— Rejeu (Replay Attack)
— Données aléatoires (Fuzzing Attack)

L'injection de trames pourrait permettre de perturber le fonctionnement du véhicule en provoquant des comportements non désirés. L'usurpation d'identité et la mascarade permettraient de prendre le rôle d'un nœud légitime, en faisant en sorte que celui-ci soit inactif dans le second cas. Le déni de service pourrait bloquer l'usage de certaines fonctionnalités. Les empoisonnements ARP et DHCP permettraient de recevoir des communications destinées à d'autres nœuds, et même de contrôler le trafic dans certains cas. L'attaque de l'homme au milieu (MITM) permettrait à l'attaquant de contrôler les communications d'un nœud victime. Les attaques par rejeu permettraient de reproduire certains comportements du véhicule. Enfin, les attaques par données aléatoires (Fuzzing) permettraient de perturber le fonctionnement normal du véhicule, et éventuellement de trouver des trames permettant de mettre en place d'autres attaques par la suite. Nous avons donc une variété d'attaques qui

pourraient conduire à des scénarios critiques pour la sécurité des usagers du véhicule et de leur environnement, une combinaison de plusieurs de ces attaques mènerait à un contrôle total du véhicule si elles permettaient de compromettre un accès au bus CAN par exemple. Les auteurs proposent donc des défenses, qui seront analysées dans la section suivante 2.4.2.

Vincenzi et al. [87] ont créé une grande base de données d'articles traitant des attaques sur l'Ethernet automobile sur les 10 dernières années, afin de proposer un résumé. Ils ont donc retrouvé toutes les attaques citées précédemment, ainsi qu'une variété d'attaques plus rares, que nous listons ici.

— Contrefaçon (Forgery)
— Détournement (Hijacking)
— Débordement de tampon (BufferOverflow)
— Attaque par sauts (Hopping)
— Altération (Tampering)
— Force Brute (Brute Force)
— Jumeaux (Sybil)
— Inondation SYN (SYN Flooding)

La contrefaçon est une attaque au cours de laquelle une victime authentifiée et connectée sur une session sécurisée navigue sur le site web d'un attaquant, l'amenant à invoquer des actions non désirées. Dans ce cas précis, l'eavesdropping concerne l'écoute du protocole d'identification dans le but d'usurper l'identité de la victime ultérieurement. L'hijacking est similaire au MITM, à la différence près qu'elle est mise en place après le protocole d'authentification entre les deux. Le buffer overflow consiste à envoyer des données d'une taille supérieure à celle attendue afin d'exécuter du code malicieux chez la victime. Le hopping consiste à contourner les protections entre deux sous réseaux pour se déplacer latéralement. Le tampering consiste à effectuer des actions interdites pour perturber le fonctionnement du système. Le bruteforce consiste à forcer un mécanisme d'authentification en tentant de très nombreuses combinaisons différentes. L'attaque par jumeaux est une attaque dans laquelle l'attaquant va créer des copies de sa victime pour les utiliser à son avantage. Enfin, le SYN Flooding est un cas particulier de déni de service utilisant les trames de synchronisation du réseau. Nous pouvons donc voir que ce second article nous montre de nouvelles attaques plus sophistiquées, pouvant découler sur des risques considérables pour les usagers.

Nous voyons donc par cet état de l'art que les réseaux Ethernet véhiculaires sont susceptibles d'introduire de nouvelles vulnérabilités, qui pourraient conduire à des risques critiques pour les usagers. Les attaques Ethernet sont variées, et concernent une variété de protocoles, ce qui peut les rendre plus difficiles à identifier. Cependant, la majorité d'entre elles correspondent

à des attaques bien connues dans d'autres types de réseaux plus classiques, et nous pourrions donc probablement utiliser les connaissances de ces attaques pour mieux les mitiger dans le cas automobile. Afin de résumer cette section, nous dressons le tableau 2.3, et nous classons les attaques selon deux critères : le modèle STRIDE présenté plus haut [33], et la triade usuelle Confidentialité-Intégrité-Disponibilité. Une analyse de risque ne sera pas donnée ici, compte-tenu de la variété des cas d'applications pour chaque scénario, cependant nous donnerons pour chaque attaque le nombre d'occurrences dans la littérature, issue de l'article de Vincenzi et al. [87], pour donner une idée de la probabilité d'application réelle.

Tableau 2.3 Classification des attaques recensées sur le réseau Ethernet véhiculaire

Attaque	**STRIDE**	**CIA**	**Occurrences**
Déni de service	D	A	20
Rejeu	T	CI	16
Injection de trames	T	I	8
MITM	STRI	CI	8
Usurpation d'identité	S	CI	7
Mascarade	S	CI	6
Eavesdropping	I	C	6
Empoisonnement du cache ARP	T	CI	4
Empoisonnement DHCP	T	CI	4
Contrefaçon	TRE	I	4
Hijacking	STRI	CI	4
BufferOverflow	TE	I	2
Hopping	IE	CI	2
Tampering	T	IA	2
Force Brute	E	A	1
Jumeaux	ST	CI	1
SYN Flooding	D	A	1
Données aléatoires	TD	I	N/A

2.4.2 Défenses

Afin de contrer cette grande variété d'attaques la littérature a proposé divers mécanismes de défense, qui se divisent principalement entre les protocoles d'authentification et de chiffrement, et les systèmes de détection d'intrusion.

Authentification et chiffrement

La première catégorie contient une grande variété de protocoles différents utilisant le même principe, mais avec des technologies différentes. Afin d'alléger la lecture, nous proposons de

résumer les défenses données par les deux articles précédents en quelques lignes, le lecteur curieux est invité lire les références [86, 87] pour plus d'informations.

L'authentification des messages passe principalement par le calcul et la vérification de MAC, qui sont calculés selon diverses méthodes. Parmi celles proposées, nous retrouvons les algorithmes MD5 et HMAC-SHAx, ou x représente la variante exacte utilisée (par exemple 256, 512, 1). Ces deux algorithmes sont courants mais présentent certains inconvénients. L'algorithme SHA peut être victime d'attaques [88], et notamment SHA1 a été retiré par la NIST suite à la découverte de vulnérabilités, et ne serait donc pas sécurisé pour générer de bons HMAC. Aussi, ces algorithmes dépendent de clés, et celles-ci doivent donc être choisies judicieusement pour éviter toute compromission, avec une longeur suffisante. Enfin, SHA peut être coûteux en calcul, et donc une application dans l'embarqué nécessiterait probablement une optimisation des algorithmes et du support matériel. Quant à MD5, il est vulnérable aux attaques par collision [88] et est considéré obsolète aujourd'hui. Il est encore utilisé dans certains cas, et en embarqué nous pourrions bénéficier de son faible coût computationnel, mais il semble plus sage d'utiliser SHA. Avec l'authentification, nous bloquons toutes les attaques d'impersonnation et d'injection de message, et éventuellement de rejeu si un compteur est pris en compte, mais les autres attaques restent possibles.

Pour les chiffrements, la technologie la plus courante est AES, avec une clé de taille variable. Si celle-ci est d'au moins 128 bits, nous pouvons considérer l'algorithme comme sûr, cependant il est primordial de choisir un mode de chiffrement adapté selon notre besoin, puisque certains peuvent laisser paraître les motifs répétitifs. Aussi, AES nécessitant des clés cryptographiques, le principal risque réside dans la gestion de ces clés au sein du réseau. AES peut être assez coûteux en ressources, mais un bon support matériel le rend utilisable en embarqué. Avec le chiffrement, nous empêchons l'écoute passive, les attaques de l'homme au milieu, et toutes les attaques de manipulation des trames sur le réseau, mais certaines attaques comme le déni de service restent possibles.

Nous voyons donc que ces défenses sont efficaces pour bon nombres d'attaques, mais certaines restent possibles. De plus, elles impliquent malheureusement un besoin en calcul et donc coût supplémentaire pour chaque ECU les implémentant, qui peut s'avérer être un frein dans le cas d'une adoption de l'industrie automobile, chaque véhicule contenant des dizaines d'ECUs de ce type.

IDS

Afin de détecter les attaques non mitigées par les mécanismes vus précédemment, nous allons procéder à un bref état de l'art des IDS Ethernet, notamment de ceux spécialisés pour les

véhicules.

Moldenhauer et al. [89] ont proposé d'adapter les méthodes ML appliquées au bus CAN à Ethernet, afin de potentiellement créer un IDS commun. La détection se base donc principalement sur la spécification et l'analyse temporelle. Leur travail a été évalué sur plusieurs bases de données, et ils obtiennent moins de 1% de faux positifs, cependant, les vrais positifs ne sont pas donnés et nous ne pouvons donc pas conclure sur la performance réelle.

B. C. Douss et al. [86] ont établi un état de l'art des IDS pour Ethernet véhiculaire parmi leurs pistes de défenses possibles. Nombre d'entre eux se basent sur l'apprentissage machine et proposent donc des IDS basés sur les anomalies, mais leur état de l'art met en évidence le manque d'évaluation de la détection de la part des auteurs. De plus, certains des IDS étudiés sont spécifiques à un protocole particulier, et ne permettrait donc pas de sécuriser tout le réseau Ethernet de celui-ci, mais seulement une partie. Le manque d'information ne nous permet donc malheureusement pas de conclure sur la faisabilité de tels IDS, cette technologie demandera plus de travail pour atteindre la maturité. Nous remarquons cependant une approche intéressante soulignée par leur état de l'art, concernant le placement de ces IDS. Plusieurs articles proposent la mise en place d'un IDS sur les gateways inter-réseaux, afin d'analyser plusieurs sources à la fois, et d'éviter la propagation des attaques. La validité de cette approche avait déjà été mise en évidence par notre état de l'art sur le bus CAN, et elle se confirme ici.

Zihan et al. [90] ont suivi une approche différente, en proposant un IDS basé sur les règles. En se basant sur les IDS existants sur les réseaux classiques, ils ont eu pour idée d'adopter cette architecture afin de pouvoir réutiliser les connaissances agrégées au fil du temps par les très nombreux utilisateurs. Ils proposent donc un algorithme de traduction des règles utilisées pour Snort [91] et Suricata, les deux IDS open-source les plus répandus. Snort et Suricata sont nourris par une communauté d'utilisateurs et d'experts en cybersécurité pour obtenir un jeu de règles régulièrement mis à jour, permettant de détecter les attaques les plus récentes et sophistiquées. Les règles sont composées de 3 composantes majeures [92] :

- Action : 8 choix possibles :
  - alert
  - block
  - drop
  - log
  - pass
  - react
  - reject

    - rewrite
- Header : 6 critères à renseigner
    - Protocole
    - Adresse A
    - Port A
    - Direction (->, <- ou <->)
    - Adresse B
    - Port B
- Options : Si besoin

Ainsi, chaque règle définit une action à effectuer pour un type de communication donnée, spécifiée par le header. Les composantes du header peuvent utiliser des symboles spéciaux pour accepter un champ de valeurs, ou même toutes les valeurs. Les options précisent les données liées à l'alerte comme un message par exemple, ou bien donnent des détails sur le contenu du message pour appliquer ou non l'action associée. En réutilisant les règles existantes et en en créant de nouvelles spécifiques aux réseaux véhiculaires, les auteurs obtiennent un taux de détection moyen de 97.67% selon les attaques, ce qui est supérieur à Suricata qui obtient 88.37%. Cette approche simple mais astucieuse, leur permet d'obtenir une détection précise des attaques à moindre coût de développement. De plus, le jeu de règles ne fait que quelques kb, et l'usage du CPU pour application sur une Raspberry Pi n'est que de quelques pourcents, cette technologie est donc parfaitement adaptée au contexte embarqué.

Pour conclure, nous avons pu voir que les mécanismes d'authentification et de chiffrement assurent une bonne protection du système, mais qu'un IDS est nécessaire en complément pour assurer l'intégrité d'un dispositif critique comme la voiture. De plus, nous avons vu que dans le cas de l'Ethernet automobile, les approches par ML sont encore trop peu matures, et qu'une détection par règles semble plus adaptée.

2.5 Autres IDS

Outre les approches classiques énumérées plus haut, certains chercheurs ont tenté de faire de la détection d'intrusion par des méthodes innovantes. Nous proposons donc d'analyser certaines d'entre elles.

Taylor et al. [93] ont proposé un système de détection d'intrusion faisant appel aux réseaux de neurones. Pour leur travail, les chercheurs ont implémenté 5 scénarios d'anomalies, correspondant pour la plupart à des attaques reportées dans la littérature. Ils ont utilisé une très grande base de données pour l'entraînement, et extrait une base de données d'évaluation

de la même source. Ils ont obtenu des résultats variables selon les anomalies et les sources analysées, mais certains sont très prometteurs. En paramétrant de sorte à n'avoir aucun faux positif, ils obtiennent jusqu'à 100% de vrais positifs selon les IDs, avec 40% de détection pour pire performance. Nous pouvons donc voir que la technologie n'est pas encore mature, mais qu'avec un travail d'approfondissement, elle pourrait probablement être utilisée dans le futur.

La société Palitronica a développé un IDS [94] basé sur les analyses par canaux auxiliaires. Comme nous l'avons vu, beaucoup d'IDS nécessitent de modifier les ECUs existants, et leur efficacité peut être mitigée par un attaquant méticuleux. Ils ont donc choisi d'utiliser des informations comme la consommation énergétique, ou le bruit d'un dispositif, pour reconnaître un fonctionnement normal, ou une anomalie. Ainsi, nous n'avons plus besoin de modifier les ECUs mais simplement de mettre un capteur dessus, et les attaquants ne pourront pas mitiger leurs traces, de par la nature incorruptible des phénomènes physiques qui sont mesurés. Les détails de leur technologie ne sont pas donnés, mais l'approche semble prometteuse, et mériterait d'être approfondie.

Enfin, Cosson et al. [95] ont proposé un IDS basé sur les métriques du noyau des acteurs d'un réseau IoT. En effet, la grande majorité des attaques impliquent une modification du comportement de l'ECU victime. Si du code malicieux est injecté, celle-ci ne se comportera plus normalement, si elle est victime d'un DoS, sa consommation énergétique sera plus élevée pour gérer les trames entrantes, etc. De cette manière, ils ont pu obtenir 95% de vrais positifs, pour seulement 2% de faux positifs dans leur meilleur résultat. Cette piste pourrait être adaptée avec quelques changements au cas des réseaux intra-véhiculaires, qui partagent des contraintes de ressources similaires.

CHAPITRE 3 PROPOSITION D'IDS

Dans cette section, nous allons présenter l'approche choisie pour mettre en place un système de détection d'intrusion pour réseaux intra-véhiculaires. Nous commencerons pour cela par introduire l'architecture des véhicules afin de choisir un emplacement de l'IDS au sein de ce réseau. Ensuite, nous présenterons la base du support matériel utilisé pour effectuer la détection. Dans un troisième temps, nous indiquerons les méthodes de détection choisies, en les classifiant selon leur type. Finalement, nous ferons une analyse de la manière dont l'IDS devra réagir en cas d'alertes.

3.1 Architecture et placement au sein du véhicule

Afin de définir précisément l'emplacement de notre IDS, il est nécessaire de bien comprendre l'architecture classique d'un véhicule moderne.

Comme nous l'avons vu précédemment, les voitures sont de nos jours équipées d'un ou plusieurs bus CAN. Les bus CAN utilisent un câble central partagé entre l'ensemble des nœuds, sur lequel passent toutes les communications.

Les plus critiques gèrent les ECUs comme le moteur ou la direction. Ces bus de communication ont pour contrainte principale la latence de transmission de l'information, puisqu'une commande de freinage par exemple ne peut pas être décalée ne serait-ce que des quelques centièmes de seconde, sans risque d'accident grave. Pour cette raison, ils sont souvent connectés à un nombre relativement faible d'ECUs, afin d'assurer une marge de fonctionnement en tous temps. Ces bus sont souvent appelés **bus CAN à haute priorité**.

D'autres bus CAN sont souvent présents également, pour connecter d'autres systèmes, comme par exemple le système de signalisation du véhicule. Ces bus, qui gèrent par exemple les clignotants ou les phares, peuvent tolérer une latence légèrement plus élevée, puisqu'ils ne concernent pas les composants critiques des ECUs. Cependant, ces fonctionnalités restent essentielles à un bon comportement du véhicule, raison pour laquelle ils n'utilisent pas Ethernet. Ces bus sont souvent appelés **bus CAN à priorité moyenne**.

Enfin, certains véhicules utilisent aussi des bus CAN pour des fonctionnalités non essentielles, comme la climatisation par exemple. Ces bus peuvent tolérer plus de latence, puisqu'ils ne concernent que des fonctionnalités de confort. Ces bus sont aussi bien souvent plus accessibles physiquement, afin de permettre l'ajout de composants optionnels vendus par le fabricant, ce qui représenterait une vulnérabilité s'ils étaient connectés aux ECUs critiques. Ces bus sont

souvent appelés **bus CAN à faible priorité**.

En complément du réseau CAN, nous avons vu que les nouvelles générations de véhicules autonomes utilisent Ethernet pour transmettre des données plus lourdes telles que de la vidéo par exemple. Les réseaux Ethernet utilisent une topologie en étoile : au centre du réseau se trouve le switch, qui va distribuer les communications entre les nœuds branchés chacun sur leur propre port. La taille du réseau est donc uniquement limitée par le nombre de ports du switch, et parfois des switchs sont interconnectés entre eux pour obtenir une topologie en arbre.

Ces bus sont parfois essentiels au bon fonctionnement du véhicule lorsque des capteurs sont connectés dessus par exemple, auquel cas ils sont isolés des autres réseaux tels que ceux de divertissement. Nous les appellerons **bus Ethernet critiques**.

Enfin, les véhicules utilisent des bus pour les fonctionnalités de divertissement, comme la radio ou la connexion avec les smartphones. Historiquement, des protocoles spécifiques comme MOST (Media Oriented Systems Transport) étaient utilisés, mais récemment les manufacturiers utilisent de plus en plus des bus Ethernet, plus flexibles. Puisque cette recherche concerne les nouvelles générations de véhicules, nous considérerons ici qu'Ethernet sera la technologie utilisée pour ces systèmes. Nous les appellerons **bus Ethernet non-critiques**.

Afin d'assurer toutes les fonctionnalités, ces bus sont interconnectés entre eux via des passerelles. Un utilisateur qui configure son véhicule sur l'écran central, enverra une trame Ethernet, qui passera par ces passerelles afin de rejoindre le bus CAN où elle sera transmise à l'ECU cible par exemple. De la même manière, des trames pourraient venir du bus Ethernet critique, ou bien d'un autre bus CAN. À l'inverse, les bus critiques pourraient remonter des messages vers le système d'infodivertissement pour signaler un danger au conducteur par exemple. Nous voyons donc qu'une bonne interconnexion est essentielle, et que la configuration de ces passerelles jouera un grand rôle dans la sécurité du véhicule, comme cela a été souligné par notre état de l'art.

La figure 3.1 ci-dessous montre une structure assez classique, dans laquelle nous avons les bus mentionnés plus haut, et une passerelle entre ceux-ci.

Nous voyons clairement que la passerelle est le composant central pour gérer les communications au sein du véhicule. Celle-ci est connectée à l'ensemble des sous-réseaux du véhicule, quelle que soit leur nature, et elle peut donc voir passer toutes les communications avec la bonne configuration des switchs. De plus, elle permet d'isoler les communications, ce qui pourrait permettre une réponse efficace dans le cas de certaines attaques. Cette position stratégique est donc idéale pour un IDS, est c'est d'ailleurs ce qui a été suggéré par la littérature.

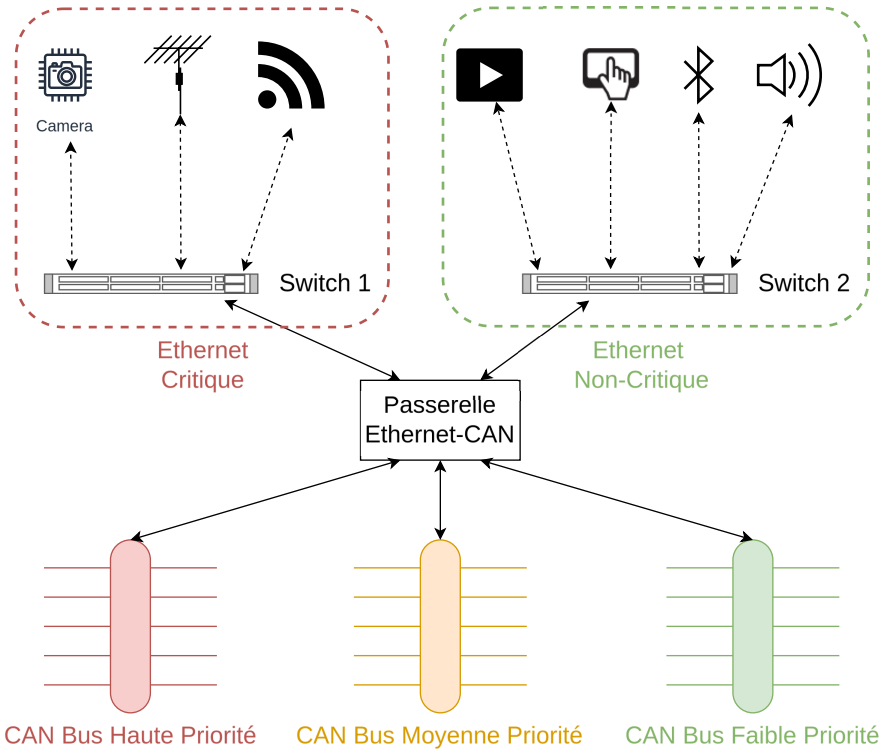

FIGURE 3.1 Architecture réseau d'un véhicule moderne.

De plus, ce composant étant déjà critique pour le véhicule, y placer un IDS permet de ne pas agrandir la surface d'attaque, et il bénéficie souvent d'une sécurité accrue. L'ensemble de ces raisons font de cet emplacement l'emplacement idéal, aussi bien d'un point de vue technique qu'économique, et nous nous proposons d'y implémenter notre IDS.

3.2 Support matériel

Notre IDS devra être capable de monitorer plusieurs réseaux en parallèle, Ethernet ou bus CAN, ainsi que d'avoir une connexion spécifique permettant de remonter les informations de sécurité et de consommation de ressources. Pour cela, il devra disposer d'une variété de connectiques différentes, que nous allons détailler ici.

Tout d'abord, afin d'avoir un maximum de libertés sur la programmation du système, ainsi que la possibilité de mettre en place des modules d'optimisation par la suite, nous avons décidé de baser notre recherche sur les solutions FPGA, et notamment les FPGAs hybrides. Les FPGAs hybrides sont une famille de FPGAs, composés à la fois d'un processeur classique (PS), et d'une partie logique programmable (PL) permettant d'implémenter une grande variété de circuits logiques. Dans notre cas, le processeur classique facilitera le développement

de l'IDS tout en offrant une puissance de calcul suffisante pour supporter un système d'exploitation avec des applications logicielles. La partie logique programmable nous permettra de mettre en place les modules de communication CAN, et dans le futur d'ajouter des modules d'optimisation, comme par exemple de cryptographie.

Afin de communiquer avec le bus CAN, il devra nécessairement être équipé d'un adaptateur. En effet, un module de communication CAN est divisé en deux composants essentiels : le contrôleur et le transmetteur (voir figure 3.2).

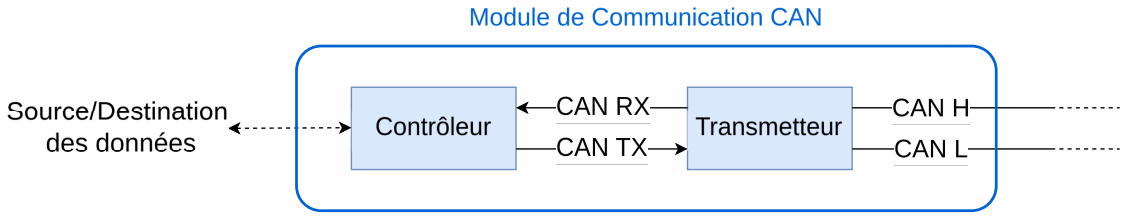

FIGURE 3.2 Architecture d'un module CAN.

Le contrôleur va permettre de préparer la structure des trames d'un point de vue logiciel, et va également gérer la communication avec le reste du système, que ça soit un capteur, ou un IDS dans notre cas. Le transmetteur va quant à lui gérer la couche physique, en introduisant les bits de parités, en gérant la négociation des envois sur le bus selon la priorité, et en mesurant la différence de voltage pour la traduire en bit. Ce module de communication devra être ajouté au FPGA afin de lui permettre de communiquer avec le bus CAN.

Afin de communiquer avec le réseau Ethernet, notre FPGA devra être muni d'une connectique réseau RJ45. Bien que les véhicules utilisent souvent des connectiques différentes adaptées au contexte embarqué, nous utiliserons une connectique plus classique par soucis de compatibilité avec le reste de nos équipements, puisque cela n'influence aucunement notre IDS.

Finalement, à des fins de journalisation, notre IDS devra être équipé d'un lecteur de carte SD, qui servira de boîte noire. Cela permettra notamment d'effectuer des analyses forensiques [1] afin de mieux comprendre les attaques, qu'elles soient détectées ou non.

Avec l'ensemble de ces composants, nous obtenons un FPGA capable de communiquer avec les réseaux qui l'entourent, et de supporter un système d'analyse en temps réel. La figure 4.2 résume l'interaction de ces composants dans notre implémentation finale pour le bus CAN (sans connexion Ethernet via RJ45).

1. Analyse forensique : recherche de preuves numériques suite à la détection d'une anomalie

3.3 Méthodes de détection

Après avoir défini la base matérielle, nous allons présenter dans cette section comment nous avons créé notre système de détection d'intrusion. Puisque celui-ci est hybride, en analysant à la fois le bus CAN et Ethernet, ces deux parties seront présentées séparément.

3.3.1 CANBus : Règles

Tout d'abord, nous avons voulu mettre en place un système de détection d'intrusion par règles, puisque le bus CAN est un dispositif dont le fonctionnement est prédéfini à l'avance.

En effet, les réseaux CAN se basent sur un fichier de spécification, appelé Database CAN (DBC) [96]. Ce fichier définit le format des communications entre les ECUs, il est constitué comme suit :

- En début de fichier les ECUs connectés au bus sont listés.
- Ensuite, les messages sont listés selon leur ID, on les nomme pour une meilleure compréhension (par exemple *ABS* pour le système ABS). De plus, la taille du champ de données du message, et le nom de l'ECU qui est supposé l'envoyer sont donnés.
- Pour chaque message, une liste de signaux est dressée, et chacun d'entre eux se voit associé un nom qui définit ce qu'il contient (par exemple *CurrentGear* pour la vitesse actuellement engagée). La structure du signal est donnée grâce aux informations suivantes : bit de début, longueur en bits, boutisme (*endianness* en anglais) et si la valeur contenue est signée ou non. Enfin, dans le but d'obtenir la véritable valeur transmise, un décalage et une échelle sont donnés, et la valeur est donc calculée selon la formule :

$$Valeur_{réelle} = Décalage + \text{É}chelle * Valeur_{message} \tag{3.1}$$

- En complément, les bornes de la valeur (minimale et maximale) sont données, ainsi que l'unité utilisée (par exemple *kmh* pour la vitesse). Cette information supplémentaire permet de comprendre le sens réel du message.
- Finalement, la liste des ECUs qui lisent ce signal est donnée.
- Pour certains signaux, un tableau de valeur peut aussi être donné en fin de document, c'est souvent le cas pour les modes de conduites par exemple, ou un entier ne permet pas de comprendre quel mode lui est associé.

Nous pouvons voir que la construction des trames suit une structure spécifique, pour que les ECUs puissent déchiffrer les différents champs du message. C'est cette particularité du bus CAN que nous exploiterons afin de détecter certaines trames malicieuses. Nous établirons

donc un fichier de règles, dérivé du fichier DBC du véhicule.

Le format des règles de notre IDS est fortement inspiré de celui des règles Snort [92], afin de simplifier la création de celles-ci. Nos règles suivent la structure suivante :

```
action ID_type ID1 ID2(optional) Frame_type Direction_symbol
 (
    # Any allowed options
 )
```

Afin de mieux comprendre le rôle de chacun des champs, nous allons détailler leur utilité dans la règle.

- `action` définit l'action à effectuer si la règle n'est pas respectée.
- `ID_type` définit si la trame suit un ID classique (11 bits) ou étendu (29 bits)
- `ID1` donne les 11 bits de l'ID
- `ID2` donne les 18 bits de l'ID étendu, si celui-ci est activé
- `Frame_type` indique si la trame est de type donnée (D) ou requête (R)
- `Direction_symbol` définit si la règle concerne les trames reçues, émises, ou les deux (->, <- ou <->).

L'ensemble de ces champs définit l'en-tête de la règle, qui permet d'identifier quelles trames sont concernées. Lorsqu'une trame est reçue, elle est comparée avec ce header, puis les options sont analysées une par une en cas de correspondance, pour appliquer l'action. Notre prototype n'implémentera que l'action *alert* dans un premier temps, mais d'autres actions comme celles données par Snort devraient être implémentées, notamment pour les trames transitant entre les réseaux Ethernet et CAN. C'est dans le champ entre parenthèses que nous précisons les options associées à la règle. Nous en avons implémenté 6 :

- *Message* donne le message associé à la règle
- *UpLimit* donne la borne supérieure d'un des signaux du messages. Il faut renseigner quels octets sont utilisés, et quelle est la valeur du seuil.
- *DownLimit* donne la borne inférieure d'un des signaux du messages. Il faut renseigner quels octets sont utilisés, et quelle est la valeur du seuil.
- *Length* donne la longueur du message en octets.
- *Format* donne le format suivi par le message. Il est donné sous forme de masque, qui doit donner 0 une fois appliqué au message avec l'opérateur logique "ET"
- *Contains* donne une suite de bits ne devant pas être contenue dans le message.

A partir de ces options, nous pouvons établir pour chaque ID une règle de conformité à la spécification donnée par le fichier DBC, ce qui nous permet de détecter des trames malfor-

mées. De plus, nous pouvons mettre en place une levée d'alerte dans le cas où aucune règle ne s'applique, puisque cela voudrait dire que la trame reçue a un ID qui n'est pas reconnu par le fichier DBC. L'option *Contains* permet en complément de détecter des données spécifiques, comme par exemple des suites connues pour faire redémarrer un ECU, ou le mettre dans un état qui nuirait à son fonctionnement normal. Enfin, le champ message permet d'associer un message spécifique à chaque option, afin d'obtenir une information précieuse pour une future analyse forensique.

Ces règles sont données ici dans un format aisément compréhensible par un humain, mais elles ne le sont pas pour un ordinateur. Nous pourrions inclure le fichier tel quel et le déchiffrer au fur et à mesure, mais ce système serait très sous performant, puisqu'un FPGA a une puissance de calcul limité, et prendrait donc un certain temps à l'analyser. Par conséquent, nous avons implémenté un analyseur syntaxique, qui prend en entrée le fichier décrit plus haut, ainsi que le fichier d'en-tête de l'IDS par règle, qui contient les structures supportées. À partir de ces deux fichiers, il va générer un fichiers *can_rules.h*, qui contiendra un tableau de règles, dans un format utilisable par l'IDS. Ce fichier contient également une variable qui indique le nombre de règles implémentées. Si une règle du fichier d'origine ne peut pas être traduite par le traducteur, celui-ci envoie un avertissement, et n'inclut pas la règle, ou la partie de la règle qui est mal constituée. Ensuite, ce fichier est inclus dans les fichiers de programmation de l'IDS, et est donc inclus lors de la compilation de celui-ci. Cette méthode nous permet d'optimiser la détection par l'IDS, tout en simplifiant grandement la procédure d'écriture des règles.

Finalement, nous pouvons voir que ces règles nous permettent en premier lieu une analyse statique des messages, en les confrontant à la spécification donnée par le constructeur. Nous pourrons ainsi détecter toutes les attaques de forge de trames, ou de fuzzing, souvent utilisées pour la reconnaissance et l'armement pré-attaque. De plus, avec l'option *contains* nous pouvons détecter certaines attaques connues, utilisant des données spécifiques. Cet ensemble de règles permettra donc une protection de nombreuses attaques parmi les plus basiques.

3.3.2 CANBus : Analyse statistique

En complément du fichier de règles, nous devons mettre en place une analyse dynamique du réseau, afin de détecter des attaques plus poussées, qui utiliseraient des messages valides à des fins malveillantes. Pour cela, nous avons décidé d'adopter une approche par analyse statistique, comme suggéré dans la littérature [34].

Pour cela, nous avons choisi de suivre le modèle des moments statistiques. Ce modèle se base sur deux métriques statistiques : la moyenne et la déviation standard. L'objectif est de

détecter des changements de comportements au sein de la base de données analysées, à l'aide de la formule suivante :

$$Nouvelle\ valeur = Moyenne\ historique + k \times Déviation\ standard\ historique \tag{3.2}$$

Ainsi, en fonction de la valeur choisie pour k, nous allons pouvoir détecter des changements plus ou moins importants au sein de la base de données. Le seul paramètre qui va influencer la détection sera donc k, qui doit être défini empiriquement à la programmation. Afin de trouver la meilleure valeur, une série de tests devra être effectuée, afin de trouver une valeur qui maximise les vrais positifs tout en minimisant les faux positifs.

La figure 3.3 montre la distribution des intervalles de confiance, dans le cas où les données suivent une loi normale. Cette figure montre qu'avec un facteur k de 3, 99% des données seront incluses dans notre intervalle de confiance.

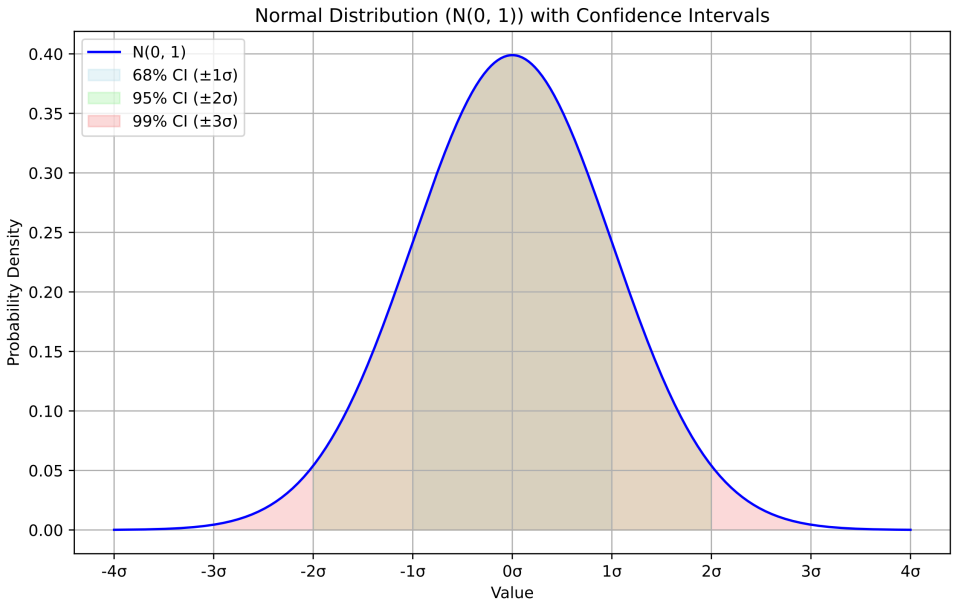

FIGURE 3.3 Distribution des intervalles de confiance pour une loi normale.

Nous voyons donc qu'en choisissant correctement la valeur de notre paramètre k, nous pouvons obtenir une détection des valeurs qui sont très peu probables.

Afin de permettre une analyse de données, nous allons enregistrer un historique du bus CAN au fur et à mesure, qui conserve les 60 dernières secondes de communication. À partir de ces données, nous tirons plusieurs métriques, et celles-ci sont analysées par moment statistique

[34]. En cas d'attaque détectée, il est important de remarquer que les valeurs de moyenne et déviation standard de chaque métrique sont enregistrées, afin de ne pas être modifiées par l'historique des données incluant l'attaque. En effet, sans cette mesure, un attaquant pourrait lancer une attaque qui provoque des changements de comportement pour le bus CAN, et donc qui lève une alerte, comme souhaité. Cependant, cette même attaque va par conséquent modifier les valeurs des métriques considérées, et donc la moyenne et la déviation standard vont évoluer en tenant compte de l'attaque, amenant le système à considérer que ce comportement est en fait normal, au bout de quelques secondes.

Pour notre prototype, nous avons décidé de suivre plusieurs métriques par cette méthode, et de les analyser de manière périodique. Notre système tourne à une période de 1 Hz, mais puisque le besoin en calcul est faible, nous pourrions augmenter cette fréquence pour un meilleur délai de détection.

Tout d'abord, afin de détecter un changement dans le bus CAN en général, nous nous intéresserons à la bande passante, en octets/seconde. Cette métrique nous permettra de détecter tout changement de comportement du bus CAN, et elle servira principalement à la détection d'attaque par déni de service, puisque celles-ci augmentent considérablement la bande passante, jusqu'à saturer le réseau. Aussi, cette mesure nous permettra de détecter des anomalies dans le fonctionnement du bus, en cas de forte baisse de la bande passante. Bien qu'aucune attaque de ce type n'ait été recensée par l'état de l'art, un tel changement présenterait un risque pour l'usager, indiquant l'absence de certaines communications.

Ensuite, nous nous intéresserons, pour chaque ID connu du véhicule (i.e. spécifié dans le fichier DBC utilisé), au nombre de trames reçues durant la dernière seconde. Un tableau historique de nombre de trames reçue à la dernière seconde sur les 60 secondes est maintenu, et nous comparons la nouvelle valeur instantanée aux données historiques, avant de choisir si une attaque a lieu. Cette mesure nous permettra de détecter des attaques comme par exemple le replay, le flooding, ou l'injection de message en cas d'augmentation. De plus, en cas de diminution, nous pouvons détecter des attaques de suspension des ECUs, comme par exemple lorsque leurs trames sont manipulées pour les pousser à se déconnecter du bus, pensant avoir une erreur.

Ainsi, nous pouvons voir que la détection par analyse statistique nous permet de détecter d'autres types d'attaques, dont l'analyse structurelle ne relèverait pas d'erreur.

3.3.3 Ethernet : Snort

Pour la détection d'attaques sur le réseau Ethernet, nous avons décidé de réutiliser des solutions existantes, afin de bénéficier des connaissances accumulées sur ces réseaux à la structure classique. Pour cela nous avons décidé de prendre en compte les deux IDS les plus courants dans le monde de l'open-source : Snort et Suricata.

Ces deux IDS, très similaires dans leur fonctionnement, se basent sur des fichiers de règles pour effectuer de la détection d'attaques connues. En complément de cette fonctionnalité, Suricata propose de l'analyse dynamique, afin de mieux analyser la structure globales des communications au sein d'une session. D'un point de vue performance, Suricata est capable de mieux tirer partie d'une architecture multicœur, et de certaines optimisations, mais cela se fait grâce à une mise en place plus complexe sur le système hôte.

Partant de cette brève analyse, et prenant en compte les contraintes liées au contexte embarqué, nous avons choisi d'utiliser Snort. En effet, nos ressources computationnelles restent notre plus importante contrainte, et un IDS léger sera donc plus adapté. De plus, l'objectif de cette recherche est d'analyser les règles utiles pour une analyse en embarqué, et les fonctionnalités plus poussées de Suricata pour certains protocoles ne pourront pas être exploitées dans notre cas, puisque les protocoles ne sont pas les même qu'en Ethernet classique.

La seconde étape pour la mise en place de Snort au sein de notre réseau, consiste en la création d'un fichier de règles adaptées. Pour cela, nous avons créé un fichier de règles spécifiques aux réseaux Ethernet intra-véhiculaires, utilisant les fonctionnalités existantes de Snort. Les règles Snort sont définies comme suit :

```
alert tcp $EXTERNAL_NET 80 -> $HOME_NET any
(
    msg:"Attack attempt!";
    flow:to_client,established;
    file_data;
    content:"1337 hackz 1337",fast_pattern,nocase;
    service:http;
    sid:1;
)
```

Les champs sont les suivants :

- `alert` donne l'action à effectuer, parmi 8 possibles
- `tcp` définit le protocole concerné par la règle
- `$EXTERNAL_NET 80` donne l'adresse et le port source de la trame concernée

— `->` donne la direction de la trame concernée
— `$HOME_NET any` donne l'adresse et le port destination de la trame concernée
— Le champs entre parenthèses contient les options de la règle. Elles donnent des informations sur quand et comment appliquer la règle, elles sont nombreuses et ne seront pas détaillées ici (c.f. [92]).

L'analyse des trames grâce aux règles de se fait en deux parties : l'analyse par préprocesseur puis l'application des règles. Une fois l'analyse faite, les sorties de Snort sont définies par les systèmes d'alertes et de journalisation, et notamment par la présence de modules optionnels, permettant des interconnexions par exemple. Le flux d'analyse de Snort est donné en figure 3.4.

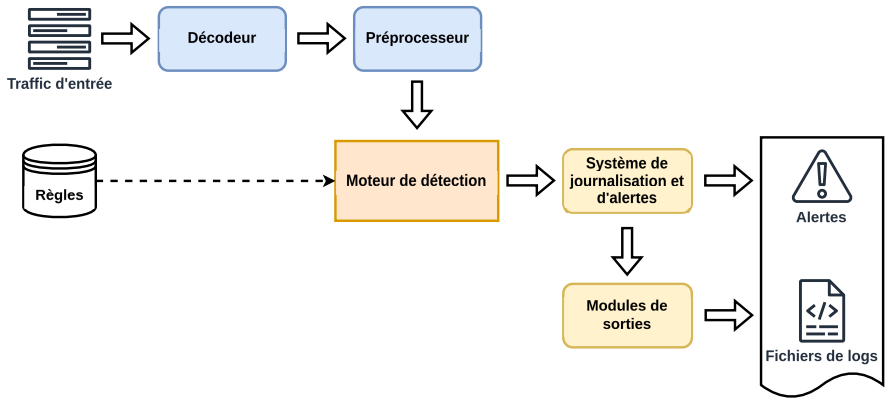

FIGURE 3.4 Flux d'analyse Snort.

Préprocesseur

Le préprocesseur est un module qui permet à Snort de mieux comprendre les communications dans leur contexte. Il intervient avant la détection par règles, en reliant les différentes communications entre elles afin de permettre une analyse comportementale du réseau. Il permet ainsi de détecter des attaques plus complexes, qui nécessitent une corrélations de plusieurs trames pour parvenir à leur objectif. Pour ce préprocesseur, nous avons implémenté deux mécanismes pour aider la détection.

Le premier mécanisme servira à détecter les attaques par analyse de ports. Ces attaques, qui permettent à un hacker de trouver des vulnérabilités dans les ports ouverts des différents composants du réseau, sont difficilement détectables si nous regardons les trames individuellement, mais facilement détectables lorsque l'on regarde le flux global et que l'on observe une répétition de certaines requêtes spécifiques. Pour cela, nous utilisons la règle suivante :

```
preprocessor sfportscan: proto{all} memcap{10000000}
   sense_level{high}
```

Cette règle indique au module de détection principal de faire appel aux analyses du préprocesseur, en suivant tous les prototypes d'alertes (*all*), avec une capacité mémoire de 10 MB pour le préprocesseur, et un niveau de sensibilité élevé (*high*). Ensuite, les prototypes de règles doivent être donnés dans le fichier de configuration de préprocesseur. Ces prototypes sont nombreux, et ne seront pas détaillés ici, mais ils sont inclus dans l'annexe A. Grâce à cette règle de préprocesseur, nous serons capable de détecter les attaques par scan de ports.

Le second mécanisme servira à détecter les attaques d'ARP spoofing. Nous incluons dans les règles du préprocesseur les 4 règles suivantes :

```
alert ( msg: "TESTARPSPOOF_UNICAST_ARP_REQUEST"; sid: 1; gid: 112;
   rev: 1; metadata: rule-type preproc ;
   classtype:protocol-command-decode; )
alert ( msg: "TESTARPSPOOF_ETHERFRAME_ARP_MISMATCH_SRC"; sid: 2;
   gid: 112; rev: 1; metadata: rule-type preproc ;
   classtype:bad-unknown; )
alert ( msg: "TESTARPSPOOF_ETHERFRAME_ARP_MISMATCH_DST"; sid: 3;
   gid: 112; rev: 1; metadata: rule-type preproc ;
   classtype:bad-unknown; )
alert ( msg: "TESTARPSPOOF_ARP_CACHE_OVERWRITE_ATTACK"; sid: 4;
   gid: 112; rev: 1; metadata: rule-type preproc ;
   classtype:bad-unknown; )
```

La première règle lance une alerte chaque fois qu'une requête ARP est lancée en mode *unicast*, puisqu'en temps normal, ce genre de requête se fait en *broadcast*. Si ce n'est pas le cas, on en déduit qu'un nœud spécifique est ciblé par l'attaque

Les règles 2 et 3 lèvent une alerte en cas d'incohérence entre l'adresse MAC de l'en-tête (respectivement source, et destination), et celle du corps de la trame ARP. Si les deux sont différentes, c'est qu'il y a usurpation d'identité.

La dernière règle lève une alerte si une tentative de réécriture du cache ARP d'un nœud est détectée. Au sein d'un réseau véhiculaire, les associations IP-MAC sont fixées à l'avance, et nous pouvons donc fixer des règles qui permettraient de détecter les spoof ARP, si les trames contiennent une correspondance non renseignée à l'avance. Nous incluons donc dans le fichier de règle Snort les règle suivantes :

```
preprocessor arpspoof_detect_host: 192.168.3.68 08:00:27:fd:aa:5e
preprocessor arpspoof_detect_host: 192.168.3.117 EE:0E:09:92:4D:1A
```

Nous n'utilisons que deux règles dans cet exemple, qui correspondent à des cas de tests. Grâce à ces règles, nous allons pouvoir alerter toute tentative de modification des tables ARP ne correspondant pas à la spécification. La règle préprocesseur donnée plus haut et ces deux règles sont indissociables pour un bon fonctionnement, le préprocesseur détecte une tentative de réécriture, transmet cette information au module de détection principal, qui compare la requête à sa table de spécification, avant de lever une alerte ou non.

Ces deux règles de préprocesseur nous permettent donc de détecter deux types d'attaques courantes, que sont le scan de port et l'ARP spoofing. Cela nous permettra de lever des alertes au conducteur du véhicule avant que celle-ci n'ait d'effets concret, en détectant dès la phase de reconnaissance.

Règles

En complément de ces règles de préprocesseur, nous devons également configurer les règles classiques du moteur de détection de Snort. Pour cela, nous avons pris différentes règles assez classiques, qui pourraient être utiles pour un réseau véhiculaire, et nous avons créer notre propre fichier de configuration. Nous avons procédé ainsi car les fichiers de base de Snort sont lourds, et incluent de nombreuses règles spécifiques à certains protocoles, comme SMTP pour les mails par exemple, qui n'existent pas dans un réseau véhiculaire, il aurait donc été plus long de supprimer les règles qui ne nous intéressent pas. Dans cette section, nous allons résumer les règles ajoutées sans les donner afin de faciliter la lecture, mais le fichier de règles est donné en annexe B.

Les premières règles que nous avons implémentée concernent les DOS et DDOS. Nous avons considéré 5 sources de déni de service possibles se basant sur TCP pour lesquelles nous avons mis en place de la détection :

— TCP SYN
— TCP FIN
— TCP ACK
— TCP RST
— TCP PSH

Pour chacune de ces sources, nous avons ajouté une ou deux règles (pour les cas DOS et DDOS) levant des alertes avec un message spécifique permettant de comprendre la source de l'attaque. Ensuite, nous avons fait de même pour les déni de service se basant sur UDP. Les attaques sur UDP sont plus simples, puisque UDP n'utilise pas les mêmes signaux de gestion de session que TCP. Par conséquent, nous avons utilisé 2 règles comptant le nombre

de trames UDP sur un intervalle de temps donné, qui lèvent une alerte si le seuil est dépassé. Nous avons implémenté des règles pour les DOS par envoi de ping, qui utilisent le protocole ICMP. Deux règles sont utilisées, la première pour analyser le nombre de ping envoyés, la seconde pour analyser les Ping of Death, qui utilisent une taille de trames très grande pour provoquer un crash de la cible. Toutes ces règles sont trouvables dans les sections *#DOS ATTACK DETECTION* et *#DDOS ATTACK DETECTION* de l'annexe B.

En complément des dénis de services, nous avons implémenté la détection de scan de ports, par TCP et UDP. Nous avons créé 5 règles pour cela, une pour UDP, et 4 pour des variantes de scan TCP. Nous avons par exemple le scan XMAS, qui utilise des trames TCP avec les flags FIN, PSH, et URG activés pour essayer de contourner certains pare-feux, ou un scan n'utilisant que des trames FIN. Les règles utilisées pour cela sont trouvables dans la section *#Detecting Various Kind of Port Scanning* de l'annexe B.

Nous avons ajouté une règle de détection d'attaque Land, qui utilise des paquets ayant la même adresse source et destination, avec pour objectif de geler la cible. Cette règle est trouvable dans la section *#Land attack* de l'annexe B.

Finalement, nous avons implémenté deux règles de détection des attaques par force brute. La première concerne les attaques par force brute sur SSH, et la seconde sur Telnet, et les deux se basent sur un nombre de requêtes sur un intervalle de temps donné. Ces règles sont trouvables dans la section *#Brute force* de l'annexe B.

Grâce à l'ensemble de ces règles, nous avons un système capable de détecter les attaques les plus classiques connues pour les réseaux Ethernet, qui nous permettra de détecter la phase de reconnaissance, ou bien d'attaque par déni de service et par force brute.

3.4 Journalisation et alertes

Le dernier élément manquant à notre IDS est le système de journalisation et d'alerte au conducteur. Afin de faciliter une analyse forensique, toutes les communications du bus CAN sont enregistrées en temps réel sur une carte SD, au format JSON. Chaque trame est représentée de la manière suivante :


```
{
    "timestamp": 12180132,
    "status": "ok",
    "direction": "received",
    "can_id": "0x180",
    "extended_id": "0x0F25A",
```

```
    "ide": 0,
    "rtr": 0,
    "dlc": 8,
    "data": ["0x00", "0x00", "0x00", "0x00", "0x00", "0x00", "0
       x00", "0x00"]
},
```

Le champ *timestamp* est exprimé en microsecondes, et est un temps relatif depuis le démarrage de l'IDS. Le statut(*status*) correspond au résultat de l'analyse par règles de la trame, et indique si elle est conforme (*"OK"*) ou non, auquel cas le message associé à la règle est indiqué. Le champ *direction* indique si la trame a été reçue ou émise par l'IDS, puisque celui-ci sert aussi de passerelle entre les deux réseaux. Les 6 derniers champs correspondent au contenu de la trame, tel que déchiffré par le contrôleur CAN.

Pour le trafic Ethernet, nous avons utilisé le système de journalisation de Snort, qui n'enregistre pas l'intégralité des trames reçues mais uniquement celles ayant levé des alertes, dans le format pcap. Celles-ci sont stockées dans leur intégralité, ce qui permet une analyse en profondeur des données suspectes. L'analyse du contexte sera plus limitée, mais le trafic Ethernet peut vite représenter beaucoup de données, et un tel tri est nécessaire dans un contexte embarqué.

De plus, afin de remonter les alertes au conducteur, l'IDS CAN retransmet en temps réel ses métriques par un port série, ainsi que chaque alerte qu'il lève. Les métriques incluent la bande passante, sa déviation standard, et le nombre de paquets reçu lors de la dernière seconde. Les alertes incluent une catégorie d'attaque, comme DOS, Flooding, etc. et éventuellement l'ID concerné par l'attaque si pertinent. Cette transmission permet de remonter et d'enregistrer les alertes sur un PC séparé, et donc d'alimenter un système d'affichage d'alertes dans un cas d'usage réel par exemple.

De même, Snort remonte ses alertes via un port série, en utilisant un module additionnel configuré via la commande suivante :

```
output alert_CSV:/dev/ttyPS0 timestamp,msg,proto,src,srcport,dst,
   dstport,tcpflags
```

A chaque alerte, le système connecté au port série pourra récupérer l'horodatage, le message de l'alerte, qui indique le type d'attaque suspecté, ainsi que les informations de base sur la trame qui a levé l'alerte. De la même manière qu'avec le bus CAN, cela pourrait alimenter un système d'affichage des alertes au conducteur dans un cas réel.

En complément, Snort enregistre ses alertes dans des fichiers spécifiques, un au format faci-

lement lisible par un humain, et le second dans un fichier au format Unified2, qui contient tous les détails de l'alerte, et quelques informations sur la trame qui l'a provoquée également.

La configuration exacte des plugins de sortie de Snort est donnée en annexe C.

L'ajout de ce système de journalisation et de levée d'alertes conclus l'architecture de notre prototype d'IDS, qui analyse en temps réel les réseaux CAN et Ethernet d'un véhicule, à l'aide d'analyse statistique ainsi que par règles.

CHAPITRE 4 IMPLÉMENTATION ET VALIDATION DU PROTOTYPE

Dans cette section, nous présenterons l'implémentation concrète de notre système de détection d'intrusion, avant de mener différentes mesures de performance, aussi bien pour la détection d'intrusion que pour la consommation de ressources.

4.1 Mise en place de l'IDS sur FPGA

Tout d'abord, notre implémentation commence par un choix de support matériel. Puisque le base de notre système est un FPGA, et que nous souhaitons qu'il soit hybride, nous avons opté pour les plateformes Pynq. Cette gamme de FPGA a été pensée pour proposer des solutions de développement simple avec les outils de Xilinx®, tels que la suite Vitis. Ces FPGA sont souvent utilisés dans le milieu scolaire, et nous disposions donc de deux modèles distincts : le PYNQ-Z2 [97], et le PYNQ-ZU [98]. Les deux modèles sont équivalents en terme d'architecture, mais diffèrent en connectiques et surtout en puissance de calcul. Nous utiliserons les deux pour évaluer les différences de performance et les besoins en ressource de notre système. Les spécifications des deux FPGA sont résumées dans le tableau 4.1.

Nous pouvons voir que le Pynq ZU est équipé d'un processeur beaucoup plus puissant que le Z2, ce qui devrait lui permettre d'être plus efficace, et de mieux gérer la répartition des tâches. Il est d'ailleurs équipé d'un ventilateur pour mieux dissiper la chaleur, ce qui peut s'avérer utile lorsque le système reste longtemps en fonctionnement, pour éviter toute dégradation de performance. De plus, il dispose d'une mémoire vive plus importante, ce qui peut être utile pour stocker les informations instantanées du système analysé. Enfin, nous pouvons voir que la partie PL de la ZU est également plus grande, mais notre projet ne dépendant que peu de cette partie, nous ne tirerons pas avantage de cette caractéristique.

La suite de cette section présentera en détail les IDS CAN puis Ethernet implémentés sur ces plateformes.

IDS CAN

Tout d'abord, nous allons présenter les différentes connectiques nécessaires à l'implémentation de notre IDS CAN.

Pour remonter les alertes et certaines métriques de l'IDS, nous aurons besoin d'une connexion par port série, et nous opterons donc pour l'usage du port JTAG des FPGA, grâce à l'UART intégré.

Tableau 4.1 Comparatif des Pynq.

	PYNQ Z2	**PYNQ ZU**
Mémoire	512MB DDR3	4GB DDR4
Processeur	Dual-Core Cortex A9	Quad core ARM Cortex-A53, Dual R5 processors
Ethernet	Oui	Non
Pmod	2	2
Système de refroidissement	Non	Ventilateur
Logique Programmable	13,300 logic slices four 6-input LUTs per slice 8 flip-flops per slice 630 KB of fast block 220 DSP slices On-chip analog-to-digital converter (XADC)	256,200 system logic cells 234,240 Flip-Flops 117,120 LUTs 5.1 Mb total block RAM, 18.0 Mb ultra RAM 1,248 DSP slices On-chip analog-to-digital converter (XADC)

La journalisation de communications se fera via la carte Micro SD intégrée dans le FPGA. Cette carte sert habituellement à contenir le programme de celui-ci, mais nous n'en avons pas besoin pour notre prototype.

De manière à communiquer avec le bus CAN, les FPGA seront équipés d'un Pmod CAN de Digilent® [99]. Ce composant s'intègre facilement via une connectique propriétaire de 12 pins, et dispose d'une connectique DB9 pour le côté bus CAN. La connectique DB9 est une connectique à 9 pins couramment utilisée pour le bus CAN, et nous permettra une adaptation facile aux autres composants. Ce transmetteur nous permettra donc de gérer la couche physique du bus CAN, et nous devrons configurer le FPGA afin de pouvoir utiliser le contrôleur associé.

Le logiciel que nous utiliserons pour configurer notre FPGA sera la suite Vitis de Xilinx® (version 2022), qui regroupe un vaste ensemble d'outils pour la programmation embarquée. Dans notre cas, nous utiliserons deux d'entre eux : Vivado pour la programmation matérielle et Vitis SDK pour la programmation logicielle.

Vivado est un logiciel de design pour la partie logique programmable du FPGA, à l'aide d'une GUI. Il permet de déployer rapidement des architectures FPGA et d'exporter le fichier de configuration de la partie logique programmable, appelé *bitstream*. Dans notre cas, nous avons dû proposer deux designs différents, un pour chaque FPGA utilisé, mais les deux sont quasiment identiques.

Pour chacun, nous avons commencé par inclure le processeur classique (respectivement Zynq 7 pour la Z2 et Zynq Ultrascale+ pour la ZU), auquel nous avons ajouté ensuite le composant HDL Pmod CAN, que nous avons relié au processeur principal via une interface AXI. Nous avons également ajouté un système de gestion des réinitialisations, qui permet un reset global du système. Enfin, nous avons connecté la sortie du contrôleur Pmod aux pins physiques du FPGA, en indiquant manuellement l'interconnexion Pin Physique-Pin HDL du contrôleur. Finalement, nous avons terminé le design en connectant le processeur à la mémoire DDR.

Ainsi, nous obtenons une conception simple qui contient tous les éléments nécessaires au bon fonctionnement de notre IDS. La conception est donnée en figure 4.1, nous pouvons y trouver les éléments externes comme les pins du Pmod ou la mémoire DDR, ainsi que le contrôleur CAN, les composants d'interconnexion, et le processeur central. Les sorties Fixed IO sont générées automatiquement par Vivado afin d'assurer le bon fonctionnement du processeur.

Nous avons effectué une analyse de l'usage des ressources, que nous avons résumée dans les tableaux 4.2 et 4.3.

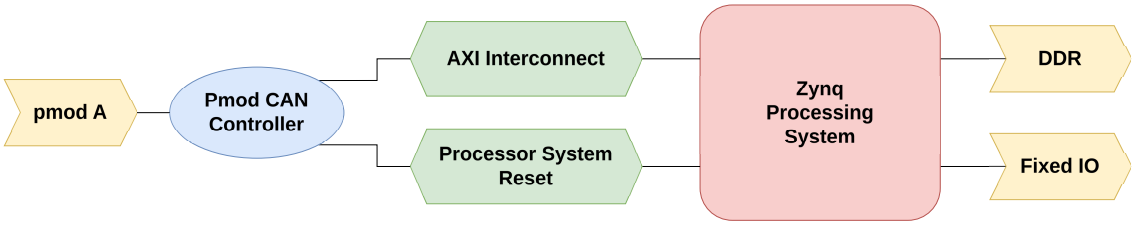

FIGURE 4.1 Conception de la section PL du FPGA.

FPGA	Ressource	Utilisation	Disponible	Utilisation (%)
Pynq Z2	LUT	941	53,200	**1.77**
	LUTRAM	74	17,400	**0.43**
	FF	1,353	106,400	**1.27**
Pynq ZU	LUT	2964	117120	**2.53**
	LUTRAM	195	57,600	**0.34**
	FF	3,501	234,240	**1.49**

TABLEAU 4.2 Utilisation des Ressources PL

La consommation en termes de ressources est, sans surprise, plus élevée pour la Pynq ZU, à cause du processeur plus gros, qui a besoin d'une plus grosse part de la logique programmable. Cependant, le point le plus important de cette synthèse est la faible utilisation des ressources globales, de l'ordre de quelques % seulement. Cela nous indique que la majorité de la logique programmable est encore disponible pour y intégrer des modules de notre choix dans le futur, comme par exemple des modules d'accélération cryptographique.

Caractéristique	Pynq Z2	Pynq ZU
Total On-Chip Power	**1.407 W**	**3.178 W**
Junction Temperature	**41.2 °C**	**32.4 °C**
Worst Negative Slack	**3.345 ns**	**6.294 ns**

TABLEAU 4.3 Rapport énergétique et temporel

Pour ce qui est de la consommation énergétique, c'est une nouvelle fois la Pynq ZU qui sera plus demandante, ce qui est conforme au résultat précédent. La consommation est plus de deux fois supérieure, mais elle reste très faible, de l'ordre de quelques Watts. La température des deux systèmes est similaire, et reste dans une zone parfaitement acceptable. Enfin, la WNS est elle aussi assez faible, bien qu'elle soit quasiment deux fois plus élevée pour la Pynq ZU, cela ne posera aucun problème. Nous n'avions pas de contraintes d'horloge, et nous avons donc pris la fréquence suggérée par Vivado de 100 MHz.

Après avoir mis en place le support matériel, nous sommes passés à la programmation logi-

cielle de notre système, grâce au logiciel Vitis IDE. Ce logiciel est fait pour être utilisé avec Vivado, et il permet donc de créer un projet se basant sur un design exporté de Vivado.

Nous sommes donc partis de notre design matériel, et nous avons créé deux domaines, un pour chaque cœur de notre Pynq Z2, ou pour les cœurs 0 et 1 de la Pynq ZU. Nous avons fait ce choix car nos premières expérimentations à un seul cœur se sont très vite retrouvées limitées par la puissance de calcul, les trames CAN arrivant à haut débit (1 kHz en considérant tous les ECUs), et les fonctionnalités d'écriture sur le port série ou la carte SD sont elles assez lentes (de l'ordre de quelques ms pour le port série, très aléatoire pour la SD).

Pour chacun des cœurs, nous avons utilisé le système d'exploitation FreeRTOS. qui est un système d'exploitation en temps réel destiné à l'embarqué. Nous avons fait ce choix afin de permettre de faire appel aux fonctionnalités d'un OS, sans pour autant perdre en performance. De plus, le choix d'un OS temps réel permet de respecter des délais maximum au besoin, ce qui pourrait être nécessaire pour le fonctionnement d'un gateway CAN-Ethernet. C'est d'ailleurs l'OS utilisé par le gateway des Tesla Model S, comme l'ont découvert Rogers et al. [50]. De plus, FreeRTOS est le système d'exploitation le mieux supporté par nos outils logiciels, et il supporte notamment bien LwIP que nous utiliserons plus tard.

Nous avons donc utilisé cette architecture à deux cœurs, afin de consacrer le cœur 1 exclusivement à la réception, l'horodatage, et la transmission des trames au cœur 0. Cette méthode nous permet d'obtenir un timestamp très précis, à la microseconde près, qui ne sera pas perturbé par les autres calculs que le cœur pourrait faire en parallèle. Afin de faire transiter les informations entre les cœurs, nous allons mettre en place une mémoire partagée sur la DDR verra transiter les trames CAN et quelques autres données de petite taille. L'architecture des interconnexions du prototype est donnée en figure 4.2.

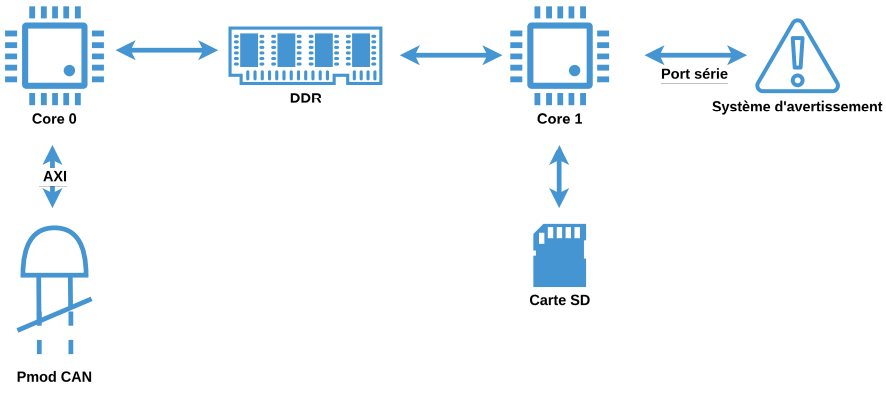

FIGURE 4.2 Interconnexions des composants de l'IDS CAN.

Afin d'obtenir notre horodatage, nous avons dû utiliser deux systèmes complémentaires. Tout d'abord, nous avons utilisé le compteur de ticks de FreeRTOS, qui sert de "métronome" pour le système d'exploitation. Le nombre de ticks par seconde est donné dans les paramètres de configuration de FreeRTOS (100 dans notre cas, la valeur par défaut), il s'agit d'un paramètre très important, car c'est lui qui régit les changements de contexte entre les différentes tâches. Nous avons par exemple fait des essais en le passant à 1000, qui rendaient le système sous-performant, à cause de changements de contexte permanents. Le compteur de tick nous permet donc d'avoir une précision à 10 ms près, mais c'est très insuffisant pour obtenir une bonne analyse temporelle du bus CAN. Pour répondre à ce problème, nous avons dû faire appel à du support matériel, à savoir le compteur de cycles de notre processeur. Ce compteur de cycles est disponible sur les processeurs de type Cortex A les plus récents, via la Performance Monitoring Unit (PMU). En comptant les cycles, nous pouvons les traduire en un temps avec la fréquence de fonctionnement du processeur. Grâce à la PMU, nous remettons à zéro le compteur de cycles via la fonction *vApplicationTickHook* qui est appelée à chaque tick. Ainsi, l'horodatage est calculé par la formule suivante :

$$Timestamp_{microsecondes} = Ticks \times 10^4 + \frac{Cycles}{Frequence\ Processeur_{Hertz}} \tag{4.1}$$

Une fois l'horodatage effectué, le cœur 1 écrit la trame avec ses métadonnées sur un buffer circulaire sur la DDR, et met ensuite à jour la tête du buffer de façon atomique, afin d'informer le cœur 0 si des trames sont disponibles ou non. Lorsque le cœur 0 le demande, le cœur 1 peut aussi lui envoyer son horodatage actuel, afin de mener des analyses de sécurité. Cela sert à avoir une horloge commune sans mettre en place des processus de synchronisation pouvant être coûteux. Grâce à ces fonctionnalités limitées, le cœur 1 est capable de gérer des données CAN arrivant à grande vitesse, sans perte. Nous avons effectué plusieurs tests, et avons remarqué une réception de 100% des trames, tant que celles-ci ont un délai d'au moins 500 ns entre chaque, ce qui est largement suffisant pour notre prototype. Au-delà, notre système d'envoi des trames était le facteur limitant, et non le FPGA. Cette rapidité d'exécution nous permet de détecter quand l'IDS, tournant sur le cœur 0, n'arrive plus à analyser en temps réel, en comptant les trames perdues.

Le cœur 1, quant à lui, va gérer toutes les fonctionnalités de détection d'intrusion, de journalisation, et de levée d'alertes.

Pour cela, nous avons développé un programme en C, qui fait appel à des threads pour gérer les tâches d'analyse de sécurité et de journalisation en parallèle.

Tout d'abord, 2 threads servent à la journalisation des trames CAN reçues sur la carte SD.

En effet, l'accès à la carte SD prend du temps et est bloquant, ce qui fait qu'un seul thread ne permettrait pas de gérer l'écriture sur la carte SD et la réception des messages du bus CAN. Les trames sont donc reçues par un thread, qui les analyse en utilisant le système de règles présenté en 3.3.1, lève une alerte au besoin, et stocke la trame dans l'historique du système de sécurité, qui est un buffer circulaire de 10 000 trames. Ensuite, il enregistre la trame au format JSON dans un buffer, avec le message donné par l'IDS par règle. Une fois le buffer plein (dans notre cas, 1000 trames), le thread passe en mode écriture, et libère le verrou qui lui donne l'accès exclusif à la DDR. Le second thread prend donc le relais pendant que le premier écrit et ainsi de suite. Cette distribution des tâches permet une analyse par règles en temps réel, tout en assurant une journalisation de l'ensemble des trames.

En complément, un minuteur logiciel est mis en place à l'aide des fonctionnalités de FreeRTOS, qui effectue à chaque seconde une analyse des trames nouvellement reçues, et les compare aux données historiques des 60 dernières secondes. Cette analyse suit la méthode indiquée dans 3.3.2 et transmet les métriques importantes ainsi que les messages d'erreur via le port série. Pour implémenter la détection par moment statistique, nous avons choisi les valeurs de k par analyse empirique. Nous avons finalement gardé les valeurs de k=5 pour la détection du DoS, qui regarde les métriques globales du bus CAN, et de k=4 pour les attaques flooding, suspend et replay, qui elles analysent les métriques de chaque ID individuellement. Ce choix se justifie par le fait que chaque ID est plus stable que le système global, impacté par les instabilités de tous les signaux.

Ce dernier ajout conclut la mise en place de notre IDS pour le bus CAN, qui est donc capable de recevoir, analyser et enregistrer les communications du bus en temps réel, tout en levant des alertes via son port série.

Snort

Pour ce qui est de l'IDS Snort, le développement a été plus simple, puisque les Pynq viennent avec un large support logiciel, permettant une prise en main via un Notebook Jupyter notamment. Nous avons donc gardé la configuration de base de la Pynq, qui utilise un OS Pynq spécifique, qui se base sur une noyau Linux. Nous avons décidé de ne pas utiliser d'OS spécialisé pour l'embarqué comme FreeRTOS dans ce cas, puisque Snort nécessite un grand nombre de dépendances du noyaux Linux.

Afin de permettre la communication avec le réseau Ethernet, nous utiliserons le port Ethernet de la Pynq Z2, et le port USB Composite de la ZU, puisque celle-ci ne dispose par de port RJ45. Le port USB est moins performant en terme de bande passante, mais il sera suffisant pour faire nos simulations.

La connexion par le Notebook nous a donné accès à un terminal root, et nous avons donc pu installer Snort avec toutes ses dépendances sur le FPGA. Ensuite, nous avons utilisé la configuration présentée en 3.3.3 pour activer les règles qui nous intéressent, et pour envoyer les alertes via le port série. Finalement, nous n'avons plus qu'à lancer Snort et la détection se lance après quelques secondes. Avant de procéder à la détection d'intrusion et à la validation du prototype, nous avons commencé par faire une analyse de la bande passante que notre FPGA était capable de gérer, afin de valider qu'il est suffisamment performant. Pour cela, nous avons effectué des tests avec UDP et TCP, et nous avons obtenu les résultats suivants : Une bande passante maximale de $1\,\mathrm{Gbit\,s^{-1}}$ en TCP, au-delà, une perte de quelques pourcents est remarquée. Une bande passante maximale de $500\,\mathrm{Mbit\,s^{-1}}$ en UDP, au-delà une perte de quelques pourcents est remarquée. Cette capacité est plus faible que ce que nous aurions aimé, mais compte-tenu que notre système ne simule pas l'envoi de données lourdes comme des images ou des vidéos, nous n'avons pas tenté d'améliorer ce résultat. En complément, nous avons évalué l'usage en CPU et mémoire de Snort au repos (sans trafic Ethernet à analyser), afin de voir s'il était possible de faire tourner d'autre systèmes en parallèle sur le FPGA. Les résultats obtenus sont donnés dans le tableau 4.4.

Tableau 4.4 Consommation de ressources de Snort au repos.

FPGA	**PYNQ-ZU**	**PYNQ-Z2**
CPU	Environ 2%	Environ 5%
RAM	10%	40%
Délai Lancement IDS	2 secondes	20 secondes
Charge Lancement IDS CPU & RAM	Moyenne	Élevée

Ainsi, nous pouvons voir que la PYNQ ZU dispose de ressources largement suffisantes pour faire tourner l'IDS, mais que la Z2 utilise déjà 40% de sa RAM au repos, ce qui veut dire qu'elle ne supportera qu'une petite charge de travail. Par conséquent, nous avons utilisé la Pynq ZU uniquement pour faire notre détection d'intrusion sur Ethernet.

Cette analyse conclut notre implémentation de Snort sur FPGA, qui nous servira pour la détection d'intrusion sur le réseau Ethernet.

Architecture complète

Finalement, l'architecture complète de notre système, avec les différentes connexions est donnée en figure 4.3.

Nous retrouvons donc la Pynq Z2, qui peut être remplacée par la ZU au besoin, qui est reliée

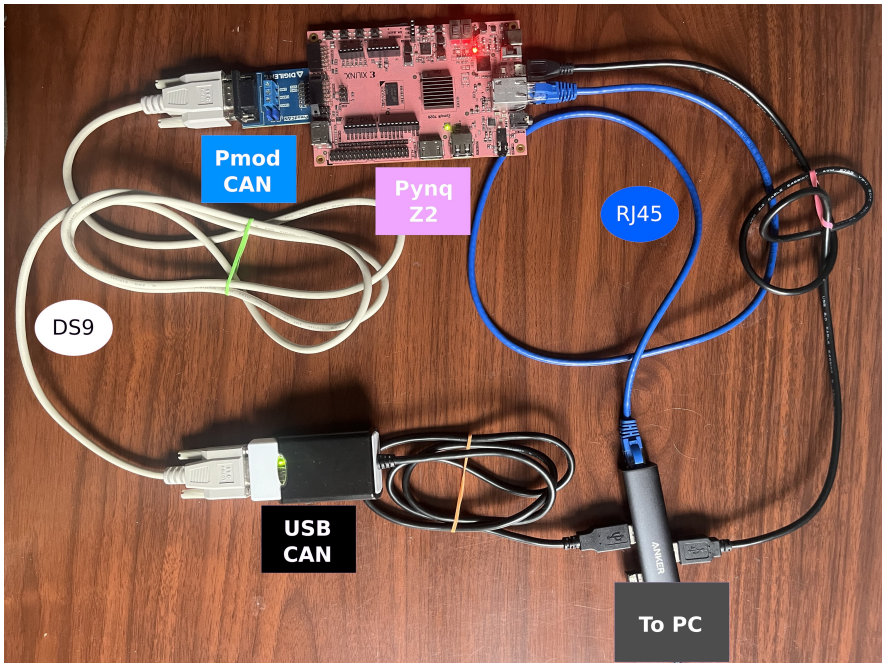

FIGURE 4.3 Setup du banc d'essai CAN.

au bus CAN via le Pmod CAN. Elle est également reliée par un port série au PC hôte pour remonter les alertes et par le câble RJ45 au réseau Ethernet. Dans ce cas, toute la simulation est faite sur le même PC hôte, et la connexion de l'ensemble du système se fait donc via un hub USB. Bien que cette figure montre un exemple où l'IDS est connecté aux deux réseaux à la fois, il convient de rappeler que les deux architectures logicielles sont différentes et que les deux ne fonctionnent pas en même temps sur le FPGA. Le FPGA peut être démarré de différentes manières pour passer d'un mode de fonctionnement à l'autre.

4.2 Simulation des réseaux intra-véhiculaires

Afin de tester les fonctionnalités de détection de notre système, nous avons mis en place un simulateur de communications sur le bus CAN. En effet, disposer d'un banc d'essai réaliste et adaptable est crucial pour expérimenter et valider les solutions de sécurité du bus CAN dans des conditions qui imitent de près les environnements réels. Un banc d'essai bien conçu nous permettra de simuler divers scénarios d'attaque, d'évaluer l'efficacité des mesures de sécurité et de générer des ensembles de données précieux pour une analyse ultérieure. Il offre un environnement contrôlé pour comprendre l'impact de différentes attaques et l'efficacité des contre-mesures, facilitant le développement de solutions de sécurité plus robustes et évolutives pour les AVs.

Tout d'abord, nous avons sélectionné l'architecture d'un véhicule en particulier pour simuler ses communications internes. Pour cela, nous avons recherché dans la base des données OpenDBC [100], et nous avons choisi l'Opel Omega de 2001. Nous avons fait ce choix car nous souhaitions implémenter un véhicule à l'architecture simple dans un premier temps, et qui contient tout de même tous les signaux les plus importants du véhicules (vitesse, angle du volant, etc.). Ce fichier définit 11 ECU/TCU [1], qui sont résumés dans le tableau 4.5.

Nous pouvons voir que le bus CAN contient les signaux les plus importants de la voiture, que nous pourrons générer avec notre simulateur, ainsi que d'autres signaux moins importants, et plus stables, comme les modes de conduite, ou le régulateur de vitesse. Nous avons établi pour chacun de ces signaux une période de 10 ms, qui correspond au délai minimal entre deux messages d'un même ECU dans notre état de l'art. En pratique, certains ECUs envoient moins souvent leurs données, mais nous voulions être sûr que notre IDS peut fonctionner sous le flow de données maximal.

Afin de générer les données du véhicule, nous avons utilisé deux simulateurs différentes : une solution de base sur mesure codée par nos soins, et le simulateur de conduite open-source CarlaSim [101].

Pour notre simulateur sur mesure, nous avons codé en C un programme qui génère un thread par ECU simulé et qui envoie les données périodiquement, toutes les 10 ms. Les valeurs des différents capteurs sont générées par des scénarios aléatoires, qui prennent des décisions sur les trajectoires et les vitesses du véhicule. Une fois qu'un angle et une vitesse ont été choisis, une durée de la manœuvre est déterminée aléatoirement, et les threads calculent la valeur instantanée en suivant la fonction suivante :

$$Valeur_{ECU} = (1 - x^2) \times Valeur_{Objectif} \tag{4.2}$$

où x évolue ainsi :

$$x = -1 + 2 \times \text{É}tape/(Durée + 1); \tag{4.3}$$

où *Étape* est incrémentée à chaque mise à jour de la valeur (toutes les 10 ms ici).

De cette manière, nous obtenons des valeurs qui évoluent de manière progressive, jusqu'à atteindre la valeur souhaitée, puis qui reviennent progressivement à la normale. Pour les modes de conduites, et autres signaux de ce type, une valeur a été fixée arbitrairement à l'implémentation, et elle reste la même. Pour les signaux qui ne s'activent qu'en cas de problèmes comme l'ABS ou l'ESP, ils restent en Off. Pour les signaux moteur, ils dépendent directement de la valeur de la pédale d'accélérateur et de la vitesse engagée, sauf pour les températures

1. Pour une meilleure compréhension des signaux, se référer à la table d'abréviations.

Tableau 4.5 Opel Omega 2001 : Tableau des signaux simulés

ID	ECU	Signal
0x110	TCU_Data1	TorqueRequest1 TorqueRequest2 OutputShaftSpeed
0x120	ESP_Data1	ABD_Active TorqueRequestFast TorqueRequestSlow
0x180	SAS_Data	SteeringAngle SteeringSpeed
0x1a0	ECU_Data1	RPM TorqueResponse TorqueLost APP TorqueRequest
0x1c0	ECU_Data2	TPS
0x280	ECU_Data3	CruiseActive KickdownActive BrakeActive
0x2e0	TCU_Data2	TOT InputShaftSpeed
0x300	ABS_WheelSpeed	FrontLeftWheelErrorFlag FrontLeftWheelSpeed FrontRightWheelErrorFlag FrontRightWheelSpeed RearLeftWheelErrorFlag RearLeftWheelSpeed RearRightWheelErrorFlag RearRightWheelSpeed
0x318	ESP_Data2	ABS_Active ESP_Active ESP_Off
0x3e0	TCU_Data3	CurrentGear SelectorPosition SportModeActive WinterModeActive AutoNeutralActive TCC_State
0x5c0	ECU_Data4	ECT IAT

qui varient légèrement autour d'une valeur nominale. Une fois ces valeurs générées, elles sont envoyées sur une interface physique SLCAN pour être lues par le FPGA.

Le second simulateur se base sur CarlaSim, qui a été créé pour faire des tests de conduite autonome en environnement logiciel. Il utilise Python et propose un ajout facile de divers modules pour affiner la simulation ou générer des données. Nous avons pu utiliser cette fonctionnalité afin de mettre en place un système de pilotage, composé d'un volant et d'un pédalier, qui nous permet de contrôler le véhicule en direct dans la simulation. De plus, nous avons ajouté via Python une interface qui permet un envoi des données générées et une lecteur des nouvelles valeurs sur le bus CAN. Cette étape est faite grâce à une interface réseau virtuelle créée spécifiquement pour notre simulation, à l'aide de VirtualCAN, qui est implémentée par défaut dans le noyau Linux. CarlaSim se connecte à cette interface, y envoie ses données CAN, et lit ensuite ce même réseau pour mettre à jour le véhicule dans la simulation. En l'absence d'attaques, il lit donc exclusivement ses données, et le comportement de la voiture est normal. Afin que ces trames puissent être lues par l'IDS, qui est sur FPGA, nous avons ensuite mis en place une copie de l'ensemble des communications de cette interface virtuelle sur une interface physique SLCAN.

L'interface SLCAN est créée à l'aide de socketCAN, également implémenté nativement dans le noyau Linux, qui permet de mettre en place des sockets de communication faciles d'usage. Ainsi, les trames sont envoyées sur cette interface, qui est physiquement connectée à un adaptateur USB vers CAN (dans notre cas le USB-CAN de Titan Electronics®), lui-même connecté au Pmod CAN via un câble DB9. Ce passage par une couche physique permet de simuler la procédure d'arbitration usuelle du bus CAN.

Pour mettre en place nos attaques, nous allons envoyer sur le réseau CAN les paquets souhaités, en plus de ceux du simulateur. Pour notre simulateur sur mesure, on envoie directement sur SLCAN. Pour CarlaSim, cela se fait en injectant sur l'interface virtuelle, pour que le simulateur mette à jour le véhicule en prenant en compte les données de l'attaquant, ce qui perturbera la conduite dans la simulation. Pour nos mesures, nous avons mis en place les scénarios d'attaque suivants :

— Déni de service : Envoi de trames avec ID 0 toutes les millisecondes
— Flooding : Envoi de trames sur ID de notre choix toutes les 5 ms
— Suspend : Plus d'envoi de trames sur l'interface réseau pour un ID donné
— Replay : Rejoue le dernier message reçu au moment de l'activation, toutes les 10 ms
— Fuzzing : Envoi de trames au contenu aléatoire, toutes les 10 ms

Ces différents scénarios vont nous permettre de tester la capacité de détection de notre IDS ainsi que les conséquences de celles-ci sur le véhicule.

La deuxième partie de la simulation concerne le réseau Ethernet, mais CarlaSim ne propose malheureusement à ce jour aucune fonctionnalité de ce type. Afin de pallier à ce problème, nous avons utiliser l'outil *tcpreplay* afin de rejouer une base de données de communication Ethernet enregistrée au format PCAP. Nous avons choisi la base de données de Kim et al. [102] car celle-ci est open-source, et contient plusieurs dizaines de minutes de communication au total. Cette base de données contient des fichiers avec et sans attaque, mais nous n'utiliserons que le fichiers sans attaque dans ce prototype. En effet, leurs attaques concernent le contenu des trames, en rejouant en boucle une même image au lieu de la mettre à jour. Notre IDS n'implémentant que des règles de bases, pour détecter du scan de ports, du DOS etc., il ne sera pas capable de détecter ces attaques.

Afin d'ajouter des attaques, nous avons fait appel à *nmap* et *hping3* afin d'injecter des trames malicieuses sur l'interface réseau, avec les commandes suivantes :

Scan NMAP `nmap -sV 192.168.3.1/24 -e enx12bf21575a01`

TCP SYN Flood `sudo hping3 --flood --rand-source -S -p 80 192.168.3.1`

TCP FIN Flood `sudo hping3 --flood -F -p 80 192.168.3.1`

TCP RST Flood `sudo hping3 --flood -R -p 80 192.168.3.1`

TCP ACK Flood `sudo hping3 --flood -A -p 80 192.168.3.1`

TCP PSH Flood `sudo hping3 --flood -P -p 80 192.168.3.1`

Land Attack `sudo hping3 -S -a 192.168.3.1 -k -s 80 -p 80 --flood 192.168.3.1`

UDP DoS `sudo hping3 --flood --rand-source --udp -p 53 192.168.3.1`

ICMP Flood `sudo hping3 --flood --icmp 192.168.3.1`

Ping of Death `hping3 --flood --icmp --data 65495 192.168.3.1`

Scan XMAS `sudo hping3 --flood -F -P -U -p 80 192.168.3.1`

ARP Spoof `sudo arpspoof -i enx12bf21575a01 -t 192.168.3.1 192.168.3.68`

En complément de la génération des données, le simulateur CarlaSim lit ses ports série, connectés aux IDS, pour obtenir les informations d'alertes et les afficher dans le simulateur. Le simulateur les affiche dans un bandeau à droite, avec un code couleur : Vert pour aucune attaque, Orange pour reconnaissance (Scan de port Ethernet ou fuzzing sur bus CAN), et Rouge pour compromission (injection de trames ayant un impact sur le comportement du véhicule, ARP spoofing et autres). Le rendu graphique du simulateur est donné en figure 4.4.

Finalement, nous obtenons l'architecture donnée en figure 4.5.

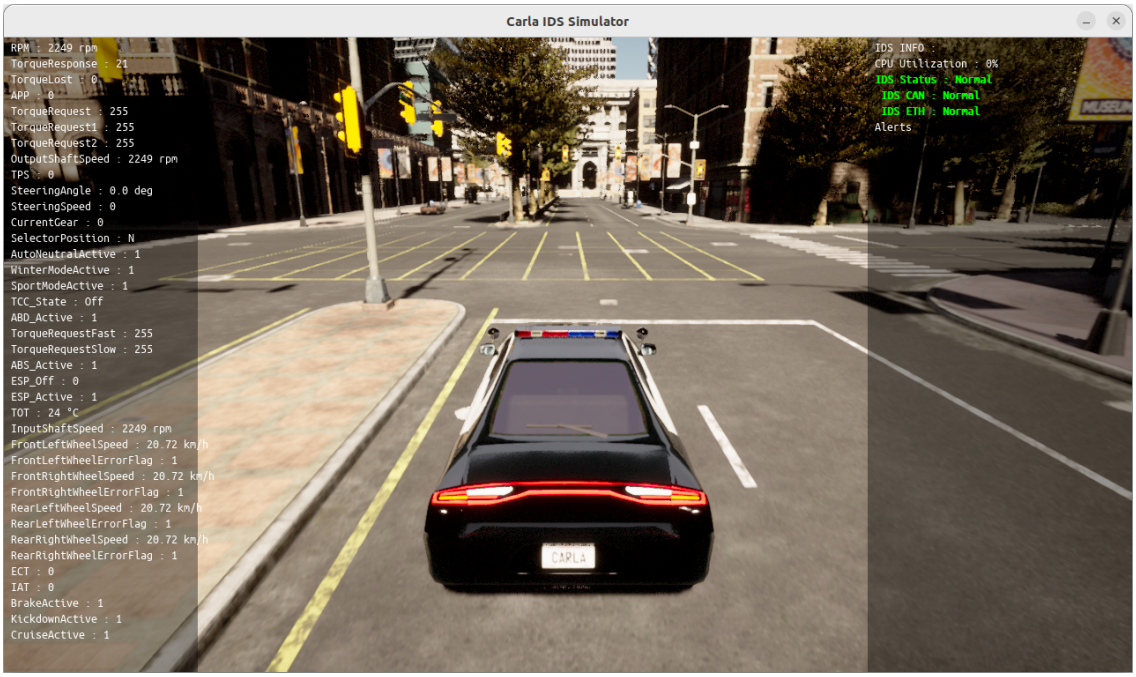

FIGURE 4.4 Capture d'écran du simulateur.

4.3 Analyse de performance

Dans cette section, nous allons présenter les mesures de performance obtenues avec nos deux IDS, et s'intéressant tout d'abord à la détection d'intrusion, et ensuite à la performance en terme de ressources.

4.3.1 Détection d'intrusions

Pour la détection d'intrusion, nous allons analyser le nombre de vrais et faux positifs, ainsi que les vrais et faux négatifs. Les formules de ces métriques sont données ci-dessous.

Taux de vrais positifs

$$TPR = \frac{TP}{TP + FN}$$

Taux de faux positifs

$$FPR = \frac{FP}{FP + TN}$$

Taux de vrais négatifs

$$TNR = \frac{TN}{TN + FP}$$

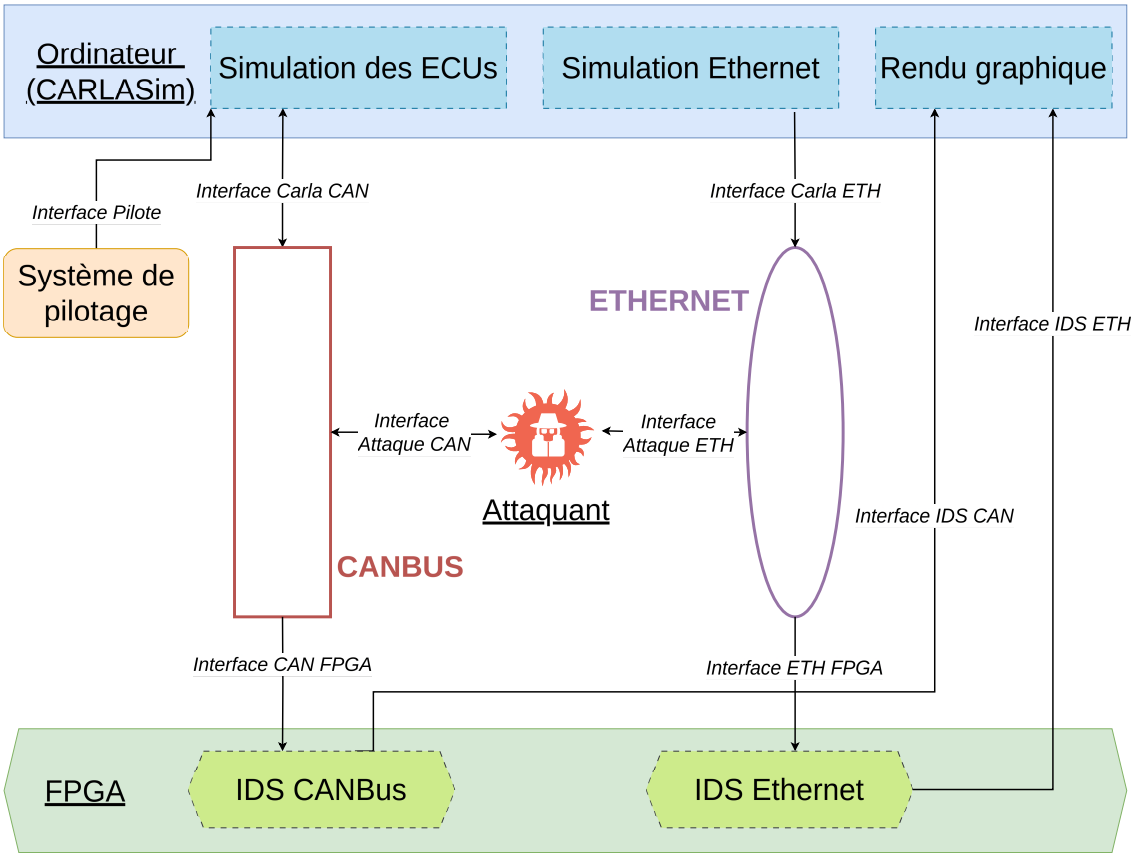

FIGURE 4.5 Architecture du banc d'essai complet.

Taux de faux négatifs

$$FNR = \frac{FN}{TP + FN}$$

Précision globale de l'IDS

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

où les sigles ont le sens suivant :

- **TP** : Vrais positifs
- **FP** : Faux positifs
- **TN** : Vrais négatifs
- **FN** : Faux négatifs

À l'aide de ces métriques, nous pourrons déterminer le niveau de fiabilité de nos IDS.

CAN Bus

Pour la détection d'attaques sur le bus CAN, nous avons fait 3 séries de tests, pour une durée totale de 20 min45 s. Le premier test, d'une durée de 12 min58 s, sert à mesurer la performance de l'IDS en fonctionnement normal, afin notamment d'obtenir une mesure fiable des faux positifs. Le 2e test, d'un durée de 4 min38 s, contient du fonctionnement normal, et des attaques de type suspend, flooding, et DoS. Le flooding dure 28 s, le DoS dure 21 s et le suspend dure 30 s. Le 3e test, d'une durée de 3 min9 s, contient 32 s d'attaque replay.

Les résultat des détection de chaque attaque est donné dans le tableau 4.6. Nous avons indiqué pour chaque attaque, le nombre de vrais et faux positifs, ainsi que les taux correspondants. Nous ne donnons par les faux négatifs, puisque notre simulation n'en a donné aucun, compte tenu des seuils choisis, qui cherchent à maximiser la détection, quitte à avoir des faux positifs.

Tableau 4.6 IDS CAN : Résultats

Attaque	**Vrais Positifs**	**Faux Positifs**	**TPR (%)**	**FPR (%)**
DoS	21	8	100	0,64
Flooding	28	3	100	0,24
Suspend	30	13	100	1,04
Replay	32	0	100	0

En plus de ces mesures de détection, nous avons aussi mesuré le nombre de trames perdues lors de la simulation, et nous en avons compté 85, sur un total de 1 245 000 de trames, ce qui nous donne une perte de 0,007% seulement. Nous avons remarqué que seules 3 trames ont été perdues en fonctionnement normal, une perte négligeable, le reste est dû aux attaques qui augmentent la quantité de données à analyser. Nous en avons notamment perdu 2 durant le flooding, soit une perte quasi nulle, et 80 durant le DoS, soit 0,004%.

Finalement, nous avons donc obtenu une précision globale de 98,1%.

Dans un second temps, nous avons testé notre détection par règles, en injectant des trames générées par fuzzing. Nous avons obtenu une détection de 100% des trames malformées, pour aucun faux positif, ce qui étai attendu, puisqu'il s'agit d'une analyse statique.

Ethernet

Pour la détection d'attaques Ethernet, nous avons mis en place une série de règles décrites en 3.3.3, et nous avons récupéré les logs d'alertes de Snort, que nous avons analysé avec les logs d'attaques afin de valider ou non l'alerte. Pour chaque attaque, nous avons rejoué la base de données de Kim et al. [102] en fond, puis au bout d'une minute, nous avons lancé l'attaque. Pour les attaques de scan de port, nous avons attendu la fin du scan, puis patienté quelques instants pour analyser le comportement post-attaque de l'IDS. Pour les autres attaques, nous avons lancé l'attaque pendant 30 secondes, nous avons attendu la fin de l'analyse des paquets par Snort, puis nous avons patienté une trentaine de secondes pour voir le comportement post-attaques. Les résultats, incluant le taux de détection et la latence (écart entre la fin de l'attaque et la fin de l'analyse par Snort), sont donnés dans le tableau 4.7.

Nous pouvons voir sur ce tableau que l'ensemble des attaques considérées sont détectées, bien que celles-ci ont un impact sur la performance de Snort. De plus, nous précisons que Snort n'a pas donné de faux positifs, sur aucune des alertes que nous avons implémenté.

4.3.2 Consommation des ressources

Pour le bus CAN, nous avons effectué une estimation de la consommation des ressources directement depuis Vitis, qui nous a permis un profilage en temps réel. Nos résultats ont montré un usage constant des cœurs du CPU, autour de 100%, que ce soit en comportement normal ou sous un scénario d'attaque. Pour le cœur 1, qui gère exclusivement la réception des trames, nous avons une moyenne de 0,02 instructions par cycle (IPC), ce qui indique qu'une bonne marge de progression est encore possible, nous pouvons donner plus de tâches à ce

Tableau 4.7 IDS Ethernet : Résultats

Attaque	**Détection**	**Latence (secondes)**
Scan NMAP	100 %	0
TCP SYN Flood	100 %	30-60
TCP FIN Flood	100 %	0
TCP ACK Flood	100 %	0
TCP RST Flood	100 %	0
TCP PSH Flood	100 %	0
Land Attack	100 %	120
UDP DoS	100 %	0
ICMP Flood	100 %	0
Ping of Death	100 %	10-20
Scan XMAS	100 %	30-60
ARP Spoof	100 %	0

cœur. Pour le cœur 0, qui gère l'analyse par l'IDS et la journalisation des communications, nous avons obtenu une moyenne de 0,82 IPC, qui est donc beaucoup plus élevé. Le Cortex A9 a un maximum théorique de 2 IPC, donc nous pouvons estimer qu'il nous reste un peu de marge, mais la limite de 2 IPC n'est atteignable que dans certaines circonstances, et notre mesure correspond donc à un fort usage des ressources disponibles.

Pour l'usage mémoire, nous avons un faible nombre de cycles de stall pour le cœur 0, avec 0 en écriture et 0,14 en lecture seulement, ce qui indique une bonne gestion de la mémoire. Pour le cœur 1, nous avons toujours 0 cycles en écriture, mais nous obtenons 0,94 stall cycles par instruction en lecture, ce qui indique que la gestion de la mémoire pourrait être optimisée pour ce cœur. Nous le voyons également par une moyenne de 93,9% d'échecs lors de l'accès au cache L1.

Pour évaluer la consommation de ressources de Snort sur notre matériel, nous avons conduit une simulation de 15 min 30 s, au cours de laquelle nous allons lancer les différents scénarios d'attaques. Cette simulation n'a été menée que sur la Pynq ZU, puisque Snort est assez demandant en ressources, et nous avons estimé que la Z2 ne serait pas bien adaptée pour le supporter. La distribution temporelle des différents scénarios d'attaque est donnée en tableau 4.8.

Ensuite, nous avons pris à chaque seconde l'usage du CPU à l'aide d'appels aux commandes adaptées du noyau Linux, et nous avons établi le graphe 4.6.

Sur ce graphe, nous voyons que l'usage du CPU reste globalement autour des 4% en temps normal, et sous une bonne partie des attaques, mais nous pouvons aussi observer un certain nombre de pics de consommations, qui atteignent voire dépassent les 30%. Nous allons tenter

Attaque	Période (s)
Scan nmap	30-90
TCP SYN Flood	130-145
TCP FIN Flood	180-195
TCP ACK Flood	300-315
TCP RST Flood	345-360
TCP PSH Flood	390-415
Land Attack	435-450
UDP DoS	630-645
ICMP	660-675
Ping of Death	705-720
Scan XMAS	735-750
ARP Spoofing	855-870

TABLEAU 4.8 Distribution temporelle des scénarios d'attaque

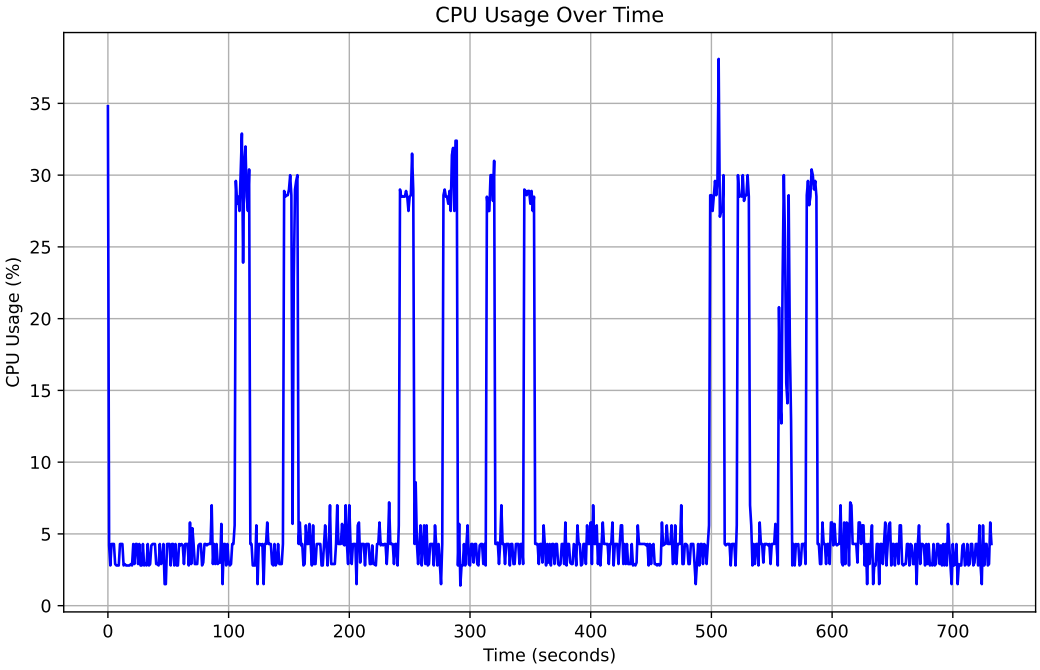

FIGURE 4.6 Courbe d'usage CPU pour Snort sur Pynq ZU.

d'expliquer brièvement le sens de chacun de ces pics.

Tout d'abord, la courbe commence par une consommation de 35% à t=0 s, qui correspond au lancement de Snort et au chargement des fichiers de règles. Ensuite, le pic autour de t=110 s correspond à un comportement normal, et nous supposons qu'il s'agit d'une conséquence

du scan nmap effectué peu avant, qui fait appel au préprocesseur, qui garde donc un usage mémoire sur une plus longue durée. Le 3e pic, à t=150 s, correspond à la fin de l'attaque TCP SYN Flood. Les 4e et 5e pics à t=250-285 s correspondent à l'attaque TCP FIN Flood, qui à été mise en place plus tôt mais dont l'envoi et l'analyse des trames dure bien après la fin de l'attaque, jusqu'à environ t=285 s. Le 6e pic, autour de t=315 s correspond à la fin du TCP ACK Flooding. Le 7e pic, autour de t=350 s correspond à l'attaque TCP RST Flood. Les 4 derniers pics, allant de t=500 s à t=580 s, correspondent tous à la Land Attack, qui est une attaque très demandante en analyse, et dont l'analyse dure bien après la fin de l'attaque.

Cette analyse conclut notre section de validation de notre IDS, qui a montré une bonne performance de détection, et ce pour un coût computationnel raisonnable.

CHAPITRE 5 DISCUSSION

Dans cette section, nous résumerons nos travaux, et nos contributions au domaine, avant d'analyser nos résultats sur les performances de détection. Ensuite, nous présenterons les limitations de notre projet, et les améliorations futures pouvant y être apportées afin de le rendre plus efficace.

5.1 Contributions de la recherche

Nos travaux ont apporté des contributions au domaine de la sécurité automobile grâce à deux fonctionnalités principales : un simulateur réaliste, et un système de détection d'intrusion efficace.

Tout d'abord, nos simulateurs sont mis en œuvre avec un PC Ubuntu, un adaptateur USB <-> CAN, un câble DS9 et une plateforme Zynq (en l'occurrence une PYNQ-Z2 ou ZU). Le PC permettant la simulation du comportement du véhicule est le seul élément nécessitant un gros besoin en ressources, les autres composants sont assez simples et notre plateforme peut donc être aisément implémentée à faible coût. De plus, notre second simulateur offre plusieurs avantages par rapport aux solutions existantes :

- l'utilisation de CarlaSim [101], un logiciel de simulation open-source,
- la génération rapide de nouveaux simulateurs à partir de fichiers «Database CAN» (DBC) [96],
- une simulation réaliste des composantes physiques du véhicules possible grâce à l'intégration de PyChrono [103],
- la possibilité de créer des bases de données CAN à partir du simulateur,
- l'intégration d'attaques par GUI avec CANdevStudio [104],
- la duplication des communications CAN vers un système physique (ensemble USB-CAN, Cable DS9, et Pynq Z2),
- l'intégration des alertes IDS dans le simulateur directement, afin de tester différentes manières de remonter les alertes.

Notre système est donc très souple, il permet de facilement changer d'architecture, d'implémenter de nouvelles attaques ou de faire des tests sur les composantes physiques du véhicule. Cet aspect nous démarque des simulateurs de la littérature, qui sont habituellement propriétaires, très coûteux et peu modulables, avec des fonctionnalités limitées. La génération de données est un autre aspect primordial, car les bases de données bus CAN sont encore

aujourd'hui peu répandues et ne contiennent que peu de données, puisque les constructeurs cherchent à garder un secret industriel. Dans une ère où le ML occupe une place exponentiellement grandissante, la clé de nos solutions futures se cache dans les bases de données à notre disposition. Enfin, l'intégration d'une interface visuelle permet la démonstration et la sensibilisation des acteurs de l'industrie, pour qui la compréhension technique peut être un facteur limitant. Nos tests ont montré l'étendue de l'impact des attaques considérés, en voyant les perturbations sur la conduite, jusqu'à obtenir un contrôle complet du véhicule. Cette fonctionnalité permet aussi de faire des tests sur les méthodes de remontée des alertes, qui sont encore méconnues, et seront primordiales pour obtenir les réactions appropriées des occupants du véhicule.

D'un point de vue cybersécurité, notre système contribue au domaine grâce aux fonctionnalités suivantes :

— détection en temps réel d'attaques sur bus CAN et réseau Ethernet,
— analyse statistique dynamique du bus CAN permettant la détection d'attaques, même Zero-Day [1],
— analyse dynamique des trames CAN à l'aide d'un fichier de règles,
— la possibilité de créer de nouvelles règles de détection suivant un format classique «Snort-like» [91, 92],
— analyse du placement stratégique de l'IDS sur la passerelle CAN-Ethernet,
— création de logs de fonctionnement du réseau interne, permettant une analyse forensique,
— une étude de l'architecture de l'IDS : support matériel, système d'exploitation et dépendances,
— une analyse de la performance de l'IDS, en terme de détection, ainsi que d'utilisation des ressources.

Bien que l'analyse statistique ait été explorée auparavant, il n'existe à notre connaissance aucune architecture qui la couple avec une détection par règle pour une meilleure détection. De plus, notre travail s'est intéressé à d'autres aspects du développement d'un IDS, en prenant en compte le placement au sein du véhicule et les besoins de ressource, ce qui rapproche notre prototype d'une adoption potentielle par l'industrie.

Les résultats de performance de nos IDS sont prometteurs, et valident la faisabilité de mise en place d'un IDS Hybride analysant à la fois des sous-réseaux CAN et Ethernet.

1. ZeroDay : Nouvelles attaques encore jamais recensées, sur lesquelles nous n'avons donc pas d'information.

Pour le réseau CAN, nous avons obtenu une précision globale de 98,1%, ce qui est prometteur pour notre solution. Les vrais positifs sont à 100%, nous avons donc une bonne capacité de détection, cependant nos faux positifs sont le facteur qui fait baisser notre score. Nos essais nous ont permis de remarquer que ces faux positifs sont principalement dus à la détection des DoS et des attaques suspend. Nous suspectons que la latence de notre simulateur est la principale cause de nos faux positifs d'attaques suspend, puisque les signaux peuvent se décaler légèrement, et nous obtenons donc parfois une fenêtre d'analyse dans laquelle une trame est manquante. Pour les attaques par déni de service, une analyse approfondie serait nécessaire, pour comprendre la source. Nous avons essayé d'augmenter la valeur du facteur k pour cette détection, mais cela nous a amené à obtenir des faux négatifs, ce que nous souhaitions éviter. La détection par règle nous a quant à elle donné une précision de 100%, ce qui était attendu puisqu'il s'agit d'une analyse de conformité statique des trames. Dans le futur, des règles plus complètes devront être ajouté, en utilisant par exemple l'option *contains* pour détecter des motifs connus dans les attaques, et ainsi permettre une détection plus large que celle des trames malformées.

Au niveau de l'utilisation des ressources nous avons vu que notre usage est déjà élevé et laisse une marge limitée, notamment pour le cœur 0. Cependant, nos tests ont été effectués sur la Pynq Z2, qui dispose d'un petit processeur, comparé à la ZU. Par conséquent, nous estimons que lors d'un usage sur la ZU, nous aurions un usage bien plus faible, qui permettrait donc une implémentation en parallèle de Snort, ou des fonctionnalités de passerelle.

Pour le réseau Ethernet, nous avons obtenu une détection de 100% pour les scénarios considérés, et ce pour un usage du CPU ne dépassant pas les 35%. Cela nous montre que l'analyse par règles de Snort, habituellement destinée aux réseaux Ethernet classiques d'entreprise, peut aussi être appliquée aux réseaux embarqués. En complément, le faible usage CPU nous garantie que notre FPGA peut exécuter d'autres fonctions en parallèle, comme par exemple celle d'une passerelle Ethernet-CAN, ou l'intégration de notre IDS CAN. Nous remarquons cependant que notre FPGA est parfois perturbée par ces mêmes attaques, et que les règles que nous avons mises en place pourraient être améliorées pour éviter une compromission de l'IDS lui-même. Certaines attaques provoquent un fort usage CPU juste après leur fin, et bien que celui-ci ne soit pas problématique en l'état, il faudra trouver la raison de cet usage excessif afin de pallier d'éventuels problèmes dans le futur. D'autres attaques impliquent de fortes latences, et nous suspectons qu'un attaquant bien informé puisse exploiter cet aspect afin de couvrir une autre attaque plus dangereuse par exemple.

5.2 Limitations de la solution proposée

Malgré ses contributions, notre prototype a plusieurs limitations, qui doivent être considérées pour relativiser nos résultats et en déduire des améliorations futures.

Tout d'abord, notre prototype manque d'une vraie simulation matérielle. Bien que nous utilisions des composants physiques pour tester notre IDS, notre système actuel se base sur la duplication du bus CAN virtuel *vcan* sur lequel se base le simulateur, sur le support physique. Par conséquent, l'ensemble des communications partent du PC de simulation, via le USB CAN, ce qui veut dire que les deux seuls nœuds du réseau qui participent au protocole d'arbritation des communications sont cet adaptateur et l'IDS. Puisque l'IDS écoute mais n'envoie pas de messages en fonctionnement normal, toute l'arbitration se fait en logiciel sur le simulateur, et donc la latence n'est pas la même. Cela empêche aussi de simuler certaines vulnérabilités, comme l'attaque par bus-off de [42] ou l'inversion de priorité [43]. Cela implique aussi que nous ne prenions pas en compte les attaques et défenses basées sur le matériel directement, comme l'analyse par canaux auxiliaires. Notre système manque donc d'un HITL plus réaliste pour une analyse plus fine.

Ensuite, nous avons montré que CarlaSim permet une bonne simulation physique du véhicule et donc une bonne génération de données CAN, mais il ne permet pas de générer tous les signaux liés aux fonctionnalités de sécurité comme l'ABS, l'ESP, et autres. Une simulation des différents logiciels propriétaires aux constructeurs permettrait donc plus de réalisme.

De plus, CarlaSim ne permet pas de simuler un réseau Ethernet véhiculaire, dont les données sont principalement du contenu multimédia. Nous avons par conséquent expérimenté notre système avec un seul jeu de données Ethernet, et nous ne pouvons pas conclure sur la capacité de l'IDS dans un environnement différent. D'ailleurs, notre simulation physique Ethernet se base sur une connectique qui n'existe en réalité pas, puisque nous utilisons un câble USB et non un RJ45 ou autre connectique Ethernet classique. Cela est dû à l'absence de bonne connectique sur notre matériel. Enfin, comme pour le bus CAN, tout le trafic part du simulateur vers l'IDS, et la simulation manque donc d'un réseau plus réaliste, avec notamment des switch et d'autres nœuds.

Notre prototype manque également de la fonctionnalité de passerelle, puisque c'est sur ce composant existant que nous proposons de l'implémenter. Nous nous sommes concentré sur l'aspect sécurité, mais notre analyse de performance bénéficierait de l'ajout de cette fonction pour plus de réalisme. Il convient notamment de remarquer que nos IDS sont évalués séparément, et qu'une implémentation des deux sur le même matériel manque à nos mesures.

Enfin, notre simulateur manque de détection spécifique aux protocoles Ethernet automobiles.

Nous n'avons considéré que des attaques sur les protocoles des couches 3 et 4 du modèle OSI (TCP et UDP), mais d'autres protocoles déviés d'Ethernet sont importants comme AVB, et d'autres protocoles de plus haut niveau comme SOME/IP devraient être soumis à des règles spécifiques.

5.3 Perspectives d'amélioration

En partant des limitations présentées ci-dessus, nous allons proposer quelques améliorations futures à apporter au projet.

Tout d'abord, nous aimerions mettre en place une meilleure simulation matérielle. Pour cela, nous souhaiterions mettre chaque ECU sur son propre support matériel, avec par exemple des cartes Arduino ou Raspberry, et les interconnecter avec un véritable câble CAN automobile. Nous pourrions donc obtenir plus de réalisme, et l'implémentation d'autres attaques. Pour générer les données, nous pourrions continuer à utiliser CarlaSim, mais il faudrait donc transmettre les valeurs aux ECUs, qui les enverraient sur le bus CAN, qui serait lu par le simulateur, afin d'avoir une boucle fermée. Une autre méthode serait que chaque ECU génère ses propres données, par simulateur ou lecture physique de métriques, avec un modèle physique par exemple. Il serait aussi intéressant d'ajouter d'autres réseaux CAN, afin de voir la performance en décuplant les quantités de données, et d'analyser la propagation des attaques au sein des différents réseaux.

De plus, nous aimerions améliorer la simulation des métriques mécaniques, avec l'aide d'un simulateur matériel. Nous avons fait des test avec PyChrono, mais le fonctionnement avec CarlaSim est instable, et nous n'avons pas donc pu l'exploiter totalement. De même, la simulation de situations plus rares comme quand lorsque l'ABS ou l'ESP s'activent serait bénéfique, pour tester la résilience de l'IDS.

Pour la partie Ethernet, nous aimerions implémenter un véritable système multimédia avec quelques Raspberry, afin de pouvoir interagir avec et générer des données, comme nous l'avons fait pour CarlaSim. Nous aimerions ajouter une console centrale qui permettrait de configurer le véhicule, une connexion à un smartphone pour évaluer cette nouvelle surface d'attaque, et également une connexion aux autres véhicules et aux infrastructures. Ces 3 exemples sont ceux qui présentent le plus gros risque car ils ne nécessitent pas un accès physique à la voiture. Aussi, nous aimerions ajouter des caméras, des RADAR et des LIDAR afin de générer des données du monde réel, ou à minima nourrir les cartes simulant ces composants avec des données récoltées lors de l'entraînement de voitures autonomes par exemple.

Pour ce qui est de la détection, nous aimerions mettre en place d'autres méthodes, à des fins

de comparatifs.

En premier lieu, pour le bus CAN nous aimerions développer une base de données de règles bien plus grosse. Actuellement nous n'avons utilisé que quelques règles de conformité pour l'Opel Omega 2001, mais nous aimerions ajouter d'autres véhicules, et pour cela il faudrait implémenter un logiciel dérivant ces règles du fichier DBC. Ensuite, nous souhaiterions ajouter la détection de payloads dangereuses, en se basant sur une analyse des attaques sur les ECUs par envoi de trames spécifiques, par fuzzing par exemple. Pour l'analyse statistique, nous aimerions ajouter d'autres méthodes que les moments statistiques, comme par exemple la somme cumulative (CUSUM), ou les chaînes de Markov. Ces méthodes plus poussées permettraient potentiellement de limiter les faux positifs, et à minima nous aurions une comparaison de ces méthodes. Ensuite, nous aimerions ajouter d'autres métriques du réseau, pour voir lesquelles sont les plus pertinentes, selon les attaques.

En second lieu, nous aimerions améliorer la détection Ethernet en implémentant des règles spécifiques aux protocoles automobiles. Pour cela, nous aimerions faire un état de l'art de ces protocoles et de leurs vulnérabilités, pour en détecter l'exploitation. Dans un second temps, nous aimerions implémenter des attaques sur ces protocoles, pour tester notre détection.

En général, nous aimerions augmenter la capacité d'action de nos IDS, en leur donnant les moyens de mitiger certaines attaques. Nous aimerions commencer par l'IDS bus CAN, en lui ajoutant les actions manquantes pour avoir les mêmes fonctionnalités que Snort. Dans un second temps, nous aimerions implémenter des fonctions de sécurité sur la passerelle CAN-Ethernet, pour bloquer la propagation d'une attaque par exemple.

Finalement, nous aimerions mettre en place un système de collecte et d'analyse des alertes des deux IDS. Nous pourrions ainsi prendre de la hauteur sur les attaques, et mieux suivre la démarche des attaquants au sein du véhicule en général, et non au sein de chaque sous-réseau. Ainsi, nous pourrions limiter les faux positifs, ou augmenter la sensibilité de la détection.

CHAPITRE 6 CONCLUSION

Cette recherche nous a permis de répondre à plusieurs questions et besoins en termes de cybersécurité des véhicules modernes.

Nous avons commencé par conduire une analyse de risque approfondie des nouvelles architectures intra-véhiculaires, en considérant les réseaux CAN et Ethernet. Dans un second temps, nous avons proposé une architecture d'IDS hybride CAN-Ethernet, à l'aide de diverses méthodes de détection. Pour tester ce système, nous avons implémenté un simulateur de communications intra-véhiculaires pour ces deux réseaux, incluant des outils de cybersécurité. Enfin, nous avons évalué notre architecture vis-à-vis de ce simulateur, afin de prouver le réalisme de notre prototype.

Notre analyse de risque nous a montré que les nouvelles attaques sur les véhicules sont une véritable menace, puisque le passage aux véhicules autonomes leur donne un impact décuplé sur leurs usagers et leur environnement. L'utilisation du bus CAN, qui n'implémente aucune fonctionnalité de sécurité, représente le plus gros risque, en permettant de prendre le contrôle de tous les composants critiques, sans aucune contre-mesure possible. De plus, la nouvelle génération de véhicules connectés augmente considérablement la surface d'attaque en permettant un accès à distance, sans avoir à modifier la structure du véhicule. Cet accès se fait le plus souvent en infiltrant le réseau Ethernet du véhicule, puis en se déplaçant latéralement jusqu'à atteindre le bus CAN. C'est pourquoi nous avons proposé de faire de la détection d'intrusion sur ces réseaux en premier lieu.

Pour cela, nous avons mis en place un premier IDS, pour le bus CAN, se basant sur une analyse par règles ainsi qu'une analyse statistique. Ce modèle nous permet de détecter à 100% toute attaque non conforme à la spécification du véhicule, en déviant des règles du fichier de configuration DBC. De plus, notre analyse statistique nous a permis d'obtenir un taux de détection de 98,1%, afin de prévenir des attaques sophistiquées répondant aux contraintes de conformité du véhicule.

Pour sécuriser le réseau Ethernet, nous avons proposé l'adoption de Snort, qui bénéficie de sa communauté pour établir une détection la plus fine possible. Nous avons implémenté une série de règles de base, qui nous ont permis d'obtenir 100% de détection pour 12 scénarios d'attaque différents, dont les scans de port, l'ARP Spoofing ou des attaques par flooding et DoS.

Ces IDS se basent sur FPGA, et sont peu demandant en ressource, ce qui devrait permettre de

les ajouter au gateway CAN-Ethernet déjà existant, pour un faible coût. Cette considération permet de réduire l'écart de réalité entre les domaines académiques et industriels, pour une meilleure adoption de notre solution dans le futur.

Afin de valider nos IDS et de tester diverses attaques, nous avons également mis en place un simulateur de conduite basé sur CarlaSim, qui permet une mise en place à très faible coût pour des analyses de cybersécurité. Ce simulateur intègre des outils d'attaque, de génération de données CAN, et permet une meilleure compréhension de leurs conséquences et moyen de détection. Il inclus une génération logicielle des signaux, et une duplication des communications sur un canal physique pour plus de réalisme.

Notre approche améliore la compréhension du domaine, puisqu'il s'agit à notre connaissance de la première architecture d'IDS hybride entre CAN et Ethernet, qui permet d'obtenir un véritable IDS intra-véhiculaire, non limité à un des sous-réseaux. Cette architecture hybride permet aussi une centralisation des informations de sécurité, qui permettront de meilleures analyses forensiques.

De plus, notre IDS CAN est le premier à croiser les analyses par règles et statistiques, ce qui permet une détection aussi bien des attaques connues que Zero-Day. Cette méthode permet également de gagner en souplesse, puisqu'une adaptation du jeu de règles peut être faite directement depuis un fichier DBC pour l'appliquer à un véhicule.

Enfin, notre simulateur est le premier à permettre de générer des données à partir d'un pilotage via une poste de conduite physique, tout en intégrant des attaques dans la boucle. Il est également le premier à intégrer la levée d'alertes informatiques par support visuel, ce qui permet une étude future de cette fonctionnalité aux enjeux critiques.

Toutefois, notre projet reste théorique, et est soumis à diverses limitations, qui ouvrent la voie à une recherche plus approfondie.

Tout d'abord, la simulation matérielle est une duplication, et bénéficierait de l'ajout de HITL afin d'obtenir une boucle fermée entre l'IDS et le simulateur, pour des analyses plus réalistes et approfondies. La simulation d'Ethernet aussi bénéficierait d'un support matériel plus réaliste, d'autant plus que notre simulateur ne génère pas ces données, mais se base sur des enregistrements faits par la littérature sur des véhicules réels, les communications des deux réseaux sont donc décorrélés.

Notre système de détection, bien que performant selon nos analyses, n'a été testé que pour une architecture de véhicule, et devra être testé dans d'autres contextes pour valider son fonctionnement. Nous souhaiterions également tester diverses méthodes de détection complémentaires, afin de couvrir la plus grande diversité d'attaques possibles.

Finalement, nos IDS ont été développés séparément, et bien que nos résultats semblent montrer qu'une fusion des deux est possible, celle-ci devra être implémentée pour confirmer cette hypothèse. Une telle fusion ouvrirait également la porte à la création d'une entité de détection qui croiserait les deux source d'information pour une sensibilité de détection accrue.

RÉFÉRENCES

[1] B. of Transportation Statistics, “Freight shipments by mode,” 2024. [En ligne]. Disponible : https://www.bts.gov/topics/freight-transportation/freight-shipments-mode

[2] T. M. Kersten Heineke, Nicholas Laverty et F. Ziegler, “The future of mobility,” 2023. [En ligne]. Disponible : https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/the-future-of-mobility-mobility-evolves

[3] J. Motavalli, “The dozens of computers that make modern cars go (and stop),” 2010. [En ligne]. Disponible : https://www.nytimes.com/2010/02/05/technology/05electronics.html?smid=url-share

[4] “Véhicules routiers — gestionnaire de réseau de communication (CAN),” International Standard Organization, Rapport technique, 2024. [En ligne]. Disponible : https://www.iso.org/fr/standard/86384.html

[5] “Véhicules routiers — gestionnaire de réseau de communication (CAN),” International Standard Organization, Rapport technique, 2024. [En ligne]. Disponible : https://www.iso.org/fr/standard/85120.html

[6] D. Zax, “Many cars have a hundred million lines of code,” 2012. [En ligne]. Disponible : https://www.technologyreview.com/2012/12/03/181350/many-cars-have-a-hundred-million-lines-of-code/

[7] S. Mazloom, M. Rezaeirad, A. Hunter et D. McCoy, “A security analysis of an In-Vehicle infotainment and app platform,” dans *10th USENIX Workshop on Offensive Technologies (WOOT 16)*. Austin, TX : USENIX Association, août 2016. [En ligne]. Disponible : https://www.usenix.org/conference/woot16/workshop-program/presentation/mazloom

[8] C. Valasek et C. Miller, “Remote exploitation of an unaltered passenger vehicle,” IOActive, Rapport technique, 2015. [En ligne]. Disponible : https://ioactive.com/remote-exploitation-of-an-unaltered-passenger-vehicle/

[9] R. Research, “United states autonomous vehicles market report,” 2024. [En ligne]. Disponible : https://www.renub.com/united-states-autonomous-vehicles-market-p.php

[10] “Sae levels of driving automation,” Society of Automotive Engineers, Rapport technique, 2021. [En ligne]. Disponible : https://www.sae.org/blog/sae-j3016-update

[11] G. D. L. Torre, P. Rad et K.-K. R. Choo, “Driverless vehicle security : Challenges and future research opportunities,” *Future Generation Computer Systems*, vol. 108,

p. 1092–1111, 2020. [En ligne]. Disponible : https://www.sciencedirect.com/science/article/pii/S0167739X17315066

[12] T. R. Aljoscha Lautenbach, Tomas Olovsson, "A state-of-the-art report on vehicular security," *Chalmers HoliSec*, 2017. [En ligne]. Disponible : https://autosec.se/wp-content/uploads/2018/04/1.2-holisec-state-of-the-art.pdf

[13] "Foresight cybersecurity threats for 2030," Agence de l'Union Européenne pour la Cybersécurité (ENISA), Rapport technique, 2024. [En ligne]. Disponible : https://www.enisa.europa.eu/publications/foresight-cybersecurity-threats-for-2030-update-2024-extended-report

[14] P. Sharma et J. Gillanders, "Cybersecurity and forensics in connected autonomous vehicles : A review of the state-of-the-art," *IEEE Access*, vol. 10, p. 108 979–108 996, 2022.

[15] "Cyber security and resilience of smart cars," Agence de l'Union Européenne pour la Cybersécurité (ENISA), Rapport technique, 2017. [En ligne]. Disponible : https://www.enisa.europa.eu/publications/cyber-security-and-resilience-of-smart-cars

[16] X. Sun, F. R. Yu et P. Zhang, "A survey on cyber-security of connected and autonomous vehicles (cavs)," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, n°. 7, p. 6240–6259, 2022.

[17] K. Kim, J. S. Kim, S. Jeong, J.-H. Park et H. K. Kim, "Cybersecurity for autonomous vehicles : Review of attacks and defense," *Computers & Security*, vol. 103, p. 102150, 2021.

[18] R. Hussain et S. Zeadally, "Autonomous cars : Research results, issues, and future challenges," *IEEE Communications Surveys & Tutorials*, vol. 21, n°. 2, p. 1275–1313, 2019.

[19] M. Amin et Z. Tariq, "Securing the car : How intrusive manufacturer-supplier approaches can reduce cybersecurity vulnerabilities," *Technology Innovation Management Review*, vol. 5, p. 21–25, 01 2015.

[20] A. Cabigiosu, F. Zirpoli et A. Camuffo, "Modularity, interfaces definition and the integration of external sources of innovation in the automotive industry," *Research Policy*, vol. 42, n°. 3, p. 662–675, 2013. [En ligne]. Disponible : https://www.sciencedirect.com/science/article/pii/S0048733312002107

[21] D. M. V. Cheri Lofy, "Demistifying platform cyber-resilience," BAE Systems, Rapport technique, 2019. [En ligne]. Disponible : https://www.baesystems.com/en-us/product/cyber-resilience

[22] S. Tuohy, M. Glavin, C. Hughes, E. Jones, M. Trivedi et L. Kilmartin, “Intra-vehicle networks : A review,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, n°. 2, p. 534–545, 2015.

[23] “Enisa good practices for security of smart cars,” Agence de l’Union Européenne pour la Cybersécurité (ENISA), Rapport technique, 2019. [En ligne]. Disponible : https://www.enisa.europa.eu/publications/smart-cars

[24] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham et S. Savage, “Experimental security analysis of a modern automobile,” dans *2010 IEEE Symposium on Security and Privacy*, 2010, p. 447–462.

[25] S. Hartzell, C. Stubel et T. Bonaci, “Security analysis of an automobile controller area network bus,” *IEEE Potentials*, vol. 39, n°. 3, p. 19–24, 2020.

[26] T. A. L. John D. Howard, “A common language for computer security incidents,” SANDIA National Laboratories, Rapport technique, 1998.

[27] C. Schmittner, Z. Ma, E. Schoitsch et T. Gruber, “A case study of FMVEA and CHASSIS as safety and security co-analysis method for automotive cyber-physical systems,” dans *Proceedings of the 1st ACM Workshop on Cyber-Physical System Security*, ser. CPSS ’15. New York, NY, USA : Association for Computing Machinery, 2015, p. 69–80. [En ligne]. Disponible : https://doi.org/10.1145/2732198.2732204

[28] J. Cui et B. Zhang, “Vera : A simplified security risk analysis method for autonomous vehicles,” *IEEE Transactions on Vehicular Technology*, vol. 69, n°. 10, p. 10 494–10 505, 2020.

[29] M. Hamad et V. Prevelakis, “Savta : A hybrid vehicular threat model : Overview and case study,” *Information*, vol. 11, n°. 5, 2020. [En ligne]. Disponible : https://www.mdpi.com/2078-2489/11/5/273

[30] I. Studnia, V. Nicomette, E. Alata, Y. Deswarte, M. Kaâniche et Y. Laarouchi, “Survey on security threats and protection mechanisms in embedded automotive networks,” dans *2013 43rd Annual IEEE/IFIP Conference on Dependable Systems and Networks Workshop (DSN-W)*, 2013, p. 1–12.

[31] V. L. Thing et J. Wu, “Autonomous vehicle security : A taxonomy of attacks and defences,” dans *2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 2016, p. 164–170.

[32] S. Satam, “Autonomous vehicle security framework (AVSF),” Thèse de doctorat, University of Arizona, 2022.

[33] "The stride threat model," Microsoft, Rapport technique, 2009. [En ligne]. Disponible : https://learn.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v=cs.20)?redirectedfrom=MSDN

[34] A. Qayyum, M. Islam et M. Jamil, "Taxonomy of statistical based anomaly detection techniques for intrusion detection," dans *Proceedings of the IEEE Symposium on Emerging Technologies, 2005.*, 2005, p. 270–276.

[35] J. Cui, L. S. Liew, G. Sabaliauskaite et F. Zhou, "A review on safety failures, security attacks, and available countermeasures for autonomous vehicles," *Ad Hoc Networks*, vol. 90, p. 101823, 2019, recent advances on security and privacy in Intelligent Transportation Systems. [En ligne]. Disponible : https://www.sciencedirect.com/science/article/pii/S1570870518309260

[36] M. M. Islam, A. Lautenbach, C. Sandberg et T. Olovsson, *A Risk Assessment Framework for Automotive Embedded Systems*. New York, NY, USA : Association for Computing Machinery, 2016, p. 3–14. [En ligne]. Disponible : https://doi.org/10.1145/2899015.2899018

[37] S. Rizvi, J. Willet, D. Perino, S. Marasco et C. Condo, "A threat to vehicular cyber security and the urgency for correction," *Procedia Computer Science*, vol. 114, p. 100–105, 2017, complex Adaptive Systems Conference with Theme : Engineering Cyber Physical Systems, CAS October 30 – November 1, 2017, Chicago, Illinois, USA. [En ligne]. Disponible : https://www.sciencedirect.com/science/article/pii/S187705091731815X

[38] S. Woo, H. J. Jo et D. H. Lee, "A practical wireless attack on the connected car and security protocol for in-vehicle can," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, n°. 2, p. 993–1006, 2015.

[39] L. Pan, X. Zheng, H. Chen, T. Luan, H. Bootwala et L. Batten, "Cyber security attacks to modern vehicular systems," *Journal of Information Security and Applications*, vol. 36, p. 90–100, 2017. [En ligne]. Disponible : https://www.sciencedirect.com/science/article/pii/S2214212616301429

[40] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner et T. Kohno, "Comprehensive experimental analyses of automotive attack surfaces," dans *20th USENIX Security Symposium (USENIX Security 11)*. San Francisco, CA : USENIX Association, août 2011. [En ligne]. Disponible : https://www.usenix.org/conference/usenix-security-11/comprehensive-experimental-analyses-automotive-attack-surfaces

[41] J. Liu, S. Zhang, W. Sun et Y. Shi, "In-vehicle network attacks and countermeasures : Challenges and future directions," *IEEE Network*, vol. 31, n°. 5, p. 50–58, 2017.

[42] K.-T. Cho et K. G. Shin, “Error handling of in-vehicle networks makes them vulnerable,” dans *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’16. New York, NY, USA : Association for Computing Machinery, 2016, p. 1044–1055. [En ligne]. Disponible : https://doi.org/10.1145/2976749.2978302

[43] K. Tindell, “Can priority inversion,” CanisLabs, Rapport technique, 2020. [En ligne]. Disponible : https://kentindell.github.io/2020/06/29/can-priority-inversion/

[44] P. Carsten, T. R. Andel, M. Yampolskiy et J. T. McDonald, “In-vehicle networks : Attacks, vulnerabilities, and proposed solutions,” dans *Proceedings of the 10th Annual Cyber and Information Security Research Conference*, ser. CISR ’15. New York, NY, USA : Association for Computing Machinery, 2015. [En ligne]. Disponible : https://doi.org/10.1145/2746266.2746267

[45] H. Lee, K. Choi, K. Chung, J. Kim et K. Yim, “Fuzzing can packets into automobiles,” dans *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*, 2015, p. 817–821.

[46] C. M. Chris Valasek, “Adventures in automotive networks and control units,” IOActive, Rapport technique, 2014. [En ligne]. Disponible : https://ioactive.com/pdfs/IOActive_Adventures_in_Automotive_Networks_and_Control_Units.pdf

[47] ——, “Can message injection : Og dynamite edition,” IOActive, Rapport technique, 2016. [En ligne]. Disponible : https://illmatics.com/can%20message%20injection.pdf

[48] N. Sen, L. Ling et D. Yuefeng, “Free-fall : Hacking tesla from wireless to can bus,” Keen Security Lab of Tencent, Rapport technique, 2016.

[49] T. K. S. Lab, “Experimental security assessment on lexus cars,” Tencent Keen Security Lab, Rapport technique, 2020. [En ligne]. Disponible : https://keenlab.tencent.com/en/2020/03/30/Tencent-Keen-Security-Lab-Experimental-Security-Assessment-on-Lexus-Cars/

[50] M. Rogers et K. Mahaffey, “How to hack a tesla model s,” DEF CON 23, 2023. [En ligne]. Disponible : https://www.youtube.com/watch?v=KX_0c9R4Fng

[51] Upstream, “Autothreat® intelligence cyber incident repository,” latest publicly reported automotive cyber incidents targeting the smart mobility ecosystem. [En ligne]. Disponible : https://upstream.auto/research/automotive-cybersecurity/?id=4720

[52] A.-I. Radu et F. D. Garcia, “Leia : A lightweight authentication protocol for can,” dans *Computer Security – ESORICS 2016*, I. Askoxylakis, S. Ioannidis, S. Katsikas et C. Meadows, édit. Cham : Springer International Publishing, 2016, p. 283–300.

[53] Q. Wang et S. Sawhney, “Vecure : A practical security framework to protect the can bus of vehicles,” dans *2014 International Conference on the Internet of Things (IOT)*, 2014, p. 13–18.

[54] P. Mundhenk, A. Paverd, A. Mrowca, S. Steinhorst, M. Lukasiewycz, S. A. Fahmy et S. Chakraborty, “Security in automotive networks : Lightweight authentication and authorization,” *ACM Trans. Des. Autom. Electron. Syst.*, vol. 22, nº. 2, mars 2017. [En ligne]. Disponible : https://doi.org/10.1145/2960407

[55] P. Mundhenk, S. Steinhorst, M. Lukasiewycz, S. A. Fahmy et S. Chakraborty, “Lightweight authentication for secure automotive networks,” dans *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2015, p. 285–288.

[56] J.-N. Luo, C.-M. Wu et M.-H. Yang, “A can-bus lightweight authentication scheme,” *Sensors*, vol. 21, nº. 21, 2021. [En ligne]. Disponible : https://www.mdpi.com/1424-8220/21/21/7069

[57] B. Groza, S. Murvay, A. V. Herrewege et I. Verbauwhede, “Libra-can : Lightweight broadcast authentication for controller area networks,” *ACM Trans. Embed. Comput. Syst.*, vol. 16, nº. 3, avr. 2017. [En ligne]. Disponible : https://doi.org/10.1145/3056506

[58] A. Herrewege, D. Singelée et I. Verbauwhede, “Canauth - a simple, backward compatible broadcast authentication protocol for can bus,” *ECRYPT Workshop on Lightweight Cryptography*, p. 7, 01 2011. [En ligne]. Disponible : https://www.researchgate.net/publication/235323481_CANAuth_-_A_Simple_Backward_Compatible_Broadcast_Authentication_Protocol_for_CAN_bus

[59] V. Verendel, D. K. Nilsson, U. E. Larson et E. Jonsson, “An approach to using honeypots in in-vehicle networks,” dans *2008 IEEE 68th Vehicular Technology Conference*, 2008, p. 1–5.

[60] A. Muhammad, D. Ayavoo et M. J. Pont, “A novel shared-clock scheduling protocol for fault-confinement in can-based distributed systems,” dans *2010 5th International Conference on System of Systems Engineering*, 2010, p. 1–6.

[61] W. A. Farag, “Cantrack : Enhancing automotive can bus security using intuitive encryption algorithms,” dans *2017 7th International Conference on Modeling, Simulation, and Applied Optimization (ICMSAO)*, 2017, p. 1–5.

[62] T. Zhang, H. Antunes et S. Aggarwal, “Defending connected vehicles against malware : Challenges and a solution framework,” *IEEE Internet of Things Journal*, vol. 1, nº. 1, p. 10–21, 2014.

[63] S. Jin, J.-G. Chung et Y. Xu, “Signature-based intrusion detection system (ids) for in-vehicle can bus network,” dans *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021, p. 1–5.

[64] J. Khan, D.-W. Lim et Y.-S. Kim, “Intrusion detection system can-bus in-vehicle networks based on the statistical characteristics of attacks,” *Sensors*, vol. 23, n°. 7, 2023. [En ligne]. Disponible : https://www.mdpi.com/1424-8220/23/7/3554

[65] H. Lee, S. H. Jeong et H. K. Kim, “Otids : A novel intrusion detection system for in-vehicle network by using remote frame,” dans *2017 15th Annual Conference on Privacy, Security and Trust (PST)*, 2017, p. 57–5709.

[66] U. E. Larson, D. K. Nilsson et E. Jonsson, “An approach to specification-based attack detection for in-vehicle networks,” dans *2008 IEEE Intelligent Vehicles Symposium*, 2008, p. 220–225.

[67] M. Weber, S. Klug, E. Sax et B. Zimmer, “Embedded Hybrid Anomaly Detection for Automotive CAN Communication,” dans *9th European Congress on Embedded Real Time Software and Systems (ERTS 2018)*, Toulouse, France, janv. 2018. [En ligne]. Disponible : https://hal.science/hal-01716805

[68] A. Taylor, N. Japkowicz et S. Leblanc, “Frequency-based anomaly detection for the automotive can bus,” dans *2015 World Congress on Industrial Control Systems Security (WCICSS)*, 2015, p. 45–49.

[69] M. Gmiden, M. H. Gmiden et H. Trabelsi, “An intrusion detection method for securing in-vehicle can bus,” dans *2016 17th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA)*, 2016, p. 176–180.

[70] M. R. Moore, R. A. Bridges, F. L. Combs, M. S. Starr et S. J. Prowell, “Modeling inter-signal arrival times for accurate detection of can bus signal injection attacks : a data-driven approach to in-vehicle intrusion detection,” dans *Proceedings of the 12th Annual Conference on Cyber and Information Security Research*, ser. CISRC ’17. New York, NY, USA : Association for Computing Machinery, 2017. [En ligne]. Disponible : https://doi.org/10.1145/3064814.3064816

[71] K.-T. Cho et K. G. Shin, “Fingerprinting electronic control units for vehicle intrusion detection,” dans *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX : USENIX Association, août 2016, p. 911–927. [En ligne]. Disponible : https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/cho

[72] M. Müter et N. Asaj, “Entropy-based anomaly detection for in-vehicle networks,” dans *2011 IEEE Intelligent Vehicles Symposium (IV)*, 2011, p. 1110–1115.

[73] C. Ling et D. Feng, “An algorithm for detection of malicious messages on can buses,” dans *Proceedings of the 2012 National Conference on Information Technology and Computer Science*. Atlantis Press, 2012/11, p. 627–630. [En ligne]. Disponible : https://doi.org/10.2991/citcs.2012.161

[74] M. R. Ansari, W. T. Miller, C. She et Q. Yu, “A low-cost masquerade and replay attack detection method for can in automobiles,” dans *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2017, p. 1–4.

[75] S. Abbott-McCune et L. A. Shay, “Intrusion prevention system of automotive network can bus,” dans *2016 IEEE International Carnahan Conference on Security Technology (ICCST)*, 2016, p. 1–8.

[76] B. Lampe et W. Meng, “Ids for can : A practical intrusion detection system for can bus security,” dans *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*, 2022, p. 1782–1787.

[77] M. Marchetti et D. Stabili, “Anomaly detection of can bus messages through analysis of id sequences,” dans *2017 IEEE Intelligent Vehicles Symposium (IV)*, 2017, p. 1577–1583.

[78] S. N. Narayanan, S. Mittal et A. Joshi, “Obd_securealert : An anomaly detection system for vehicles,” dans *2016 IEEE International Conference on Smart Computing (SMARTCOMP)*, 2016, p. 1–6.

[79] A. Wasicek et A. Weimerskirch, “Recognizing manipulated electronic control units,” University of California, Berkeley and University of Michigan, Rapport technique, 2014. [En ligne]. Disponible : https://ptolemy.berkeley.edu/projects/chess/pubs/1111/autoids_v2_preprint1.pdf

[80] A. Tomlinson, J. Bryans, S. A. Shaikh et H. K. Kalutarage, “Detection of automotive can cyber-attacks by identifying packet timing anomalies in time windows,” dans *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2018, p. 231–238.

[81] C. Wang, Z. Zhao, L. Gong, L. Zhu, Z. Liu et X. Cheng, “A distributed anomaly detection system for in-vehicle network using htm,” *IEEE Access*, vol. 6, p. 9091–9098, 2018.

[82] J. Zhang, F. Li, H. Zhang, R. Li et Y. Li, “Intrusion detection system using deep learning for in-vehicle security,” *Ad Hoc Networks*, vol. 95, p. 101974, 2019. [En ligne]. Disponible : https://www.sciencedirect.com/science/article/pii/S1570870519304354

[83] T. Moulahi, S. Zidi, A. Alabdulatif et M. Atiquzzaman, “Comparative performance evaluation of intrusion detection based on machine learning in in-vehicle controller area network bus,” *IEEE Access*, vol. 9, p. 99 595–99 605, 2021.

[84] J. Li, “Cansee : An automobile intrusion detection system,” Qihoo 360, Rapport technique, 2016. [En ligne]. Disponible : https://archive.conference.hitb.org/hitbsecconf2016ams/sessions/cansee-an-automobile-intrusion-detection-system/#

[85] N. Nowdehi, W. Aoudi, M. Almgren et T. Olovsson, “Casad : Can-aware stealthy-attack detection for in-vehicle networks,” 2019. [En ligne]. Disponible : https://arxiv.org/abs/1909.08407

[86] A. B. C. Douss, R. Abassi et D. Sauveron, “State-of-the-art survey of in-vehicle protocols and automotive ethernet security and vulnerabilities,” *Mathematical Biosciences and Engineering*, vol. 20, n°. 9, p. 17 057–17 095, 2023. [En ligne]. Disponible : https://www.aimspress.com/article/doi/10.3934/mbe.2023761

[87] M. De Vincenzi, G. Costantino, I. Matteucci, F. Fenzl, C. Plappert, R. Rieke et D. Zelle, “A systematic review on security attacks and countermeasures in automotive ethernet,” *ACM Comput. Surv.*, vol. 56, n°. 6, janv. 2024. [En ligne]. Disponible : https://doi.org/10.1145/3637059

[88] Z. E. Rasjid, B. Soewito, G. Witjaksono et E. Abdurachman, “A review of collisions in cryptographic hash function used in digital forensic tools,” *Procedia Computer Science*, vol. 116, p. 381–392, 2017, discovery and innovation of computer science technology in artificial intelligence era : The 2nd International Conference on Computer Science and Computational Intelligence (ICCSCI 2017). [En ligne]. Disponible : https://www.sciencedirect.com/science/article/pii/S1877050917321221

[89] J. E. Peter Moldenhauer, “Automotive ethernet cyberattack defense in ground vehicles,” National Defense Industrial Association Michigan, Rapport technique, 2022. [En ligne]. Disponible : https://www.ndia-mich.org/images/events/gvsets/2022/Papers/cgs/Automotive%20Ethernet%20Cyberattack%20Defense%20in%20Ground%20Vehicles.pdf

[90] Z. Zihan, C. Lirong, Z. Haitao et Z. Fan, “Research on intrusion detection technology based on embedded ethernet,” dans *2021 18th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*, 2021, p. 587–600.

[91] M. Roesch, “Snort : Lightweight intrusion detection for networks.” dans *LISA*, D. W. Parter, édit. USENIX, 1999, p. 229–238. [En ligne]. Disponible : http://dblp.uni-trier.de/db/conf/lisa/lisa1999.html#Roesch99

[92] *Snort 3 Rule Writing Guide*, Cisco. [En ligne]. Disponible : https://docs.snort.org/rules/

[93] A. Taylor, S. Leblanc et N. Japkowicz, “Anomaly detection in automobile control network data with long short-term memory networks,” dans *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, 2016, p. 130–139.

[94] “Palisade,” Palitronica, Rapport technique. [En ligne]. Disponible : https://www.palitronica.com/products/palisade

[95] A. Cosson, A. K. Sikder, L. Babun, Z. B. Celik, P. McDaniel et A. S. Uluagac, “Sentinel : A robust intrusion detection system for iot networks using kernel-level system information,” dans *Proceedings of the International Conference on Internet-of-Things Design and Implementation*, ser. IoTDI ’21. New York, NY, USA : Association for Computing Machinery, 2021, p. 53–66. [En ligne]. Disponible : https://doi.org/10.1145/3450268.3453533

[96] Vector Informatik GmbH, *DBC File Format Specification*, Accessed : 2024-06-22. [En ligne]. Disponible : http://socialledge.com/sjsu/index.php/DBC_Format

[97] *Pynq Z2 User Manual*, TUL Embedded Systems. [En ligne]. Disponible : https://dpoauwgwqsy2x.cloudfront.net/Download/PYNQ_Z2_User_Manual_v1.1.pdf

[98] *Pynq ZU User Manual*, TUL Embedded Systems. [En ligne]. Disponible : https://xilinx.github.io/PYNQ-ZU/pdf/PYNQ-ZU_ReferenceUser_Manual_v1_2.pdf

[99] *Pmod CAN User Manual*, Digilent. [En ligne]. Disponible : https://digilent.com/reference/_media/reference/pmod/pmodcan/pmodcan-rm.pdf

[100] CommaAI, “Opendbc,” 2024. [En ligne]. Disponible : https://github.com/commaai/opendbc?tab=MIT-1-ov-file#readme

[101] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez et V. Koltun, “CARLA : An open urban driving simulator,” dans *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, p. 1–16.

[102] S. Jeong, B. Jeon, B. Chung et H. K. Kim, “Automotive ethernet intrusion dataset,” 2021. [En ligne]. Disponible : https://dx.doi.org/10.21227/1yr3-q009

[103] ProjectChrono, “Pychrono : A multi-physics simulation engine,” 2024. [En ligne]. Disponible : https://projectchrono.org/pychrono/

[104] GENIVI, “Candevstudio,” 2024. [En ligne]. Disponible : https://github.com/GENIVI/CANdevStudio

ANNEXE A PROTOTYPES DE RÈGLES PRÉPROCESSEUR POUR LE SCAN DE PORT

```
alert ( msg: "PSNG_TCP_PORTSCAN"; sid: 1; gid: 122; rev: 1;
   metadata: rule-type preproc ; classtype:attempted-recon; )
alert ( msg: "PSNG_TCP_DECOY_PORTSCAN"; sid: 2; gid: 122; rev: 1;
   metadata: rule-type preproc ; classtype:attempted-recon; )
alert ( msg: "PSNG_TCP_PORTSWEEP"; sid: 3; gid: 122; rev: 1;
   metadata: rule-type preproc ; classtype:attempted-recon; )
alert ( msg: "PSNG_TCP_DISTRIBUTED_PORTSCAN"; sid: 4; gid: 122;
   rev: 1; metadata: rule-type preproc ; classtype:attempted-recon;
   )
alert ( msg: "PSNG_TCP_FILTERED_PORTSCAN"; sid: 5; gid: 122; rev:
   1; metadata: rule-type preproc ; classtype:attempted-recon; )
alert ( msg: "PSNG_TCP_FILTERED_DECOY_PORTSCAN"; sid: 6; gid: 122;
   rev: 1; metadata: rule-type preproc ;
   classtype:attempted-recon; )
alert ( msg: "PSNG_TCP_PORTSWEEP_FILTERED"; sid: 7; gid: 122; rev:
   1; metadata: rule-type preproc ; classtype:attempted-recon; )
alert ( msg: "PSNG_TCP_FILTERED_DISTRIBUTED_PORTSCAN"; sid: 8; gid:
   122; rev: 1; metadata: rule-type preproc ;
   classtype:attempted-recon; )
alert ( msg: "PSNG_IP_PORTSCAN"; sid: 9; gid: 122; rev: 1;
   metadata: rule-type preproc ; classtype:attempted-recon; )
alert ( msg: "PSNG_IP_DECOY_PORTSCAN"; sid: 10; gid: 122; rev: 1;
   metadata: rule-type preproc ; classtype:attempted-recon; )
alert ( msg: "PSNG_IP_PORTSWEEP"; sid: 11; gid: 122; rev: 1;
   metadata: rule-type preproc ; classtype:attempted-recon; )
alert ( msg: "PSNG_IP_DISTRIBUTED_PORTSCAN"; sid: 12; gid: 122;
   rev: 1; metadata: rule-type preproc ; classtype:attempted-recon;
   )
alert ( msg: "PSNG_IP_FILTERED_PORTSCAN"; sid: 13; gid: 122; rev:
   1; metadata: rule-type preproc ; classtype:attempted-recon; )
alert ( msg: "PSNG_IP_FILTERED_DECOY_PORTSCAN"; sid: 14; gid: 122;
   rev: 1; metadata: rule-type preproc ;
   classtype:attempted-recon;)
alert ( msg: "PSNG_IP_PORTSWEEP_FILTERED"; sid: 15; gid: 122; rev:
```

```
   1; metadata: rule-type preproc ; classtype:attempted-recon; )
alert ( msg: "PSNG_IP_FILTERED_DISTRIBUTED_PORTSCAN"; sid: 16; gid:
   122; rev: 1; metadata: rule-type preproc ;
   classtype:attempted-recon; )
alert ( msg: "PSNG_UDP_PORTSCAN"; sid: 17; gid: 122; rev: 1;
   metadata: rule-type preproc ; classtype:attempted-recon; )
alert ( msg: "PSNG_UDP_DECOY_PORTSCAN"; sid: 18; gid: 122; rev: 1;
   metadata: rule-type preproc ; classtype:attempted-recon; )
alert ( msg: "PSNG_UDP_PORTSWEEP"; sid: 19; gid: 122; rev: 1;
   metadata: rule-type preproc ; classtype:attempted-recon; )
alert ( msg: "PSNG_UDP_DISTRIBUTED_PORTSCAN"; sid: 20; gid: 122;
   rev: 1; metadata: rule-type preproc ; classtype:attempted-recon;
   )
alert ( msg: "PSNG_UDP_FILTERED_PORTSCAN"; sid: 21; gid: 122; rev:
   1; metadata: rule-type preproc ; classtype:attempted-recon; )
alert ( msg: "PSNG_UDP_FILTERED_DECOY_PORTSCAN"; sid: 22; gid: 122;
   rev: 1; metadata: rule-type preproc ; classtype:attempted-recon;
   )
alert ( msg: "PSNG_UDP_PORTSWEEP_FILTERED"; sid: 23; gid: 122; rev:
   1; metadata: rule-type preproc ; classtype:attempted-recon; )
alert ( msg: "PSNG_UDP_FILTERED_DISTRIBUTED_PORTSCAN"; sid: 24;
   gid: 122; rev: 1; metadata: rule-type preproc ;
   classtype:attempted-recon; )
alert ( msg: "PSNG_ICMP_PORTSWEEP"; sid: 25; gid: 122; rev: 1;
   metadata: rule-type preproc ; classtype:attempted-recon; )
alert ( msg: "PSNG_ICMP_PORTSWEEP_FILTERED"; sid: 26; gid: 122;
   rev: 1; metadata: rule-type preproc ; classtype:attempted-recon;
   )
alert ( msg: "PSNG_OPEN_PORT"; sid: 27; gid: 122; rev: 1; metadata:
   rule-type preproc ; classtype:attempted-recon; )
alert ( msg: "FRAG3_IPOPTIONS"; sid: 1; gid: 123; rev: 1; metadata:
   rule-type preproc ; classtype:protocol-command-decode; )
```

ANNEXE B FICHIER DE RÈGLES SNORT

```
#Land Attack
alert tcp $HOME\_NET 80 <> $HOME\_NET 80 (msg:"LAND ATTACK
   DETECTED";sid:10000001;rev:3;)

#Detecting Various Kind of Port Scanning
alert tcp $EXTERNAL_NET any -> $HOME_NET 80 (threshold: type
   threshold, track by_dst, count 20, seconds 60; msg:"TCP SCAN
   DETECTED";sid:1000004;rev:2;)

alert tcp $EXTERNAL_NET any -> $HOME_NET 80 (msg:"TCP FIN Scan
   Detected"; flags:F; threshold:type threshold, track by_src,
   count 2, seconds 60;classtype:attempted-recon; sid:10000005;
   rev:1;)
alert tcp $EXTERNAL_NET any -> $HOME_NET 80 (msg:"Null Scan
   Detected"; flags:0; threshold:type threshold, track by_src,
   count 20, seconds 60; classtype:attempted-recon; sid:1000006;
   rev:1;)
alert tcp $EXTERNAL_NET any -> $HOME_NET 80 (msg:"Xmas Scan
   Detected"; flags:UPF; threshold:type threshold, track by_src,
   count 2, seconds 6; classtype:attempted-recon; sid:100007;
   rev:1;)
alert udp $EXTERNAL_NET any -> $HOME_NET any (msg:"UDP SCAN
   DETECTED"; threshold:type threshold, track by_dst, count 1000,
   seconds 3; classtype:attempted-recon; sid:10000008;rev:1;)


#DOS ATTACK DETECTION
alert tcp $EXTERNAL_NET any -> $HOME_NET any (flags: S;
   msg:"Possible SYN DoS"; flow: stateless; threshold: type
   threshold, track by_dst, count 1000, seconds 3; sid:10009;rev:1;)
alert tcp $EXTERNAL_NET any -> $HOME_NET any (flags: S;
   msg:"Possible SYN DoS"; flow: stateless; threshold: type both,
   track by_dst, count 1000, seconds 3; sid:10009;rev:1;)
alert tcp $EXTERNAL_NET any -> $HOME_NET any (flags: A;
   msg:"Possible ACK DoS"; flow: stateless; threshold: type both,
```

```
   track by_dst, count 1000, seconds 3; sid:10010;rev:1;)
alert tcp $EXTERNAL_NET any -> $HOME_NET any (flags: R;
   msg:"Possible RST DoS"; flow: stateless; threshold: type both,
   track by_dst, count 1000, seconds 3; sid:100011;rev:1;)
alert tcp $EXTERNAL_NET any -> $HOME_NET any (flags: F;
   msg:"Possible FIN DoS"; flow: stateless; threshold: type both,
   track by_dst, count 1000, seconds 3; sid:100012;rev:1;)
alert udp $EXTERNAL_NET any -> $HOME_NET any (msg:"Possible UDP
   DoS"; flow: stateless; threshold: type both, track by_dst, count
   1000, seconds 3; sid:10013;rev:1;)
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"Possible ICMP
   DoS"; threshold: type both, track by_dst, count 250, seconds 3;
   sid:10014;rev:1;)
alert icmp any any -> 192.168.137.2 any (msg:"threshold";
   threshold: type threshold, track by_dst, count 100, seconds 10;
   sid:10014;rev:1;)
alert icmp any any -> $HOME_NET any (msg:"detection filter";
   sid:1000001; rev:1; classtype:icmp-event; detection_filter:track
   by_dst, count 100, seconds 10;)


#DDOS ATTACK DETECTION
alert tcp $EXTERNAL_NET any -> $HOME_NET any (flags: S;
   msg:"Possible SYN DDoS"; flow: stateless; threshold: type both,
   track by_dst, count 100000, seconds 10; sid:100015;rev:1;)
alert tcp $EXTERNAL_NET any -> $HOME_NET any (flags: A;
   msg:"Possible ACK DDoS"; flow: stateless; threshold: type both,
   track by_dst, count 100000, seconds 10; sid:100016;rev:1;)
alert tcp $EXTERNAL_NET any -> $HOME_NET any (flags: R;
   msg:"Possible RST DDoS"; flow: stateless; threshold: type both,
   track by_dst, count 100000, seconds 10; sid:100017;rev:1;)
alert tcp $EXTERNAL_NET any -> $HOME_NET any (flags: F;
   msg:"Possible FIN DDoS"; flow: stateless; threshold: type both,
   track by_dst, count 100000, seconds 10; sid:100018;rev:1;)
alert udp $EXTERNAL_NET any -> $HOME_NET any (msg:"Possible UDP
   DDoS"; flow: stateless; threshold: type both, track by_dst,
   count 100000, seconds 10; sid:100019;rev:1;)
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"Possible ICMP
```

```
   DDoS"; threshold: type both, track by_dst, count 100000, seconds
   10; sid:100020;rev:1;)

#PING OF DEATH DETECTION
alert icmp any any -> $HOME_NET any (msg:"Possible Ping of Death";
   dsize: > 10000; sid:100021;rev:1;)

#BRUTE FORCE & scan connexion
alert tcp $EXTERNAL_NET any -> $HOME_NET 22 (msg:"Possible SSH
   Brute Force Attack"; threshold:type both, track by_src, count
   10, seconds 60; sid:100029; rev:1;)
alert tcp $EXTERNAL_NET any -> $HOME_NET 22 (msg:"Possible Scan
   port 22"; threshold: type both, track by_src, count 5, seconds
   60; sid:100032; rev:1;)
alert tcp $EXTERNAL_NET any -> $HOME_NET 23 (msg:"Possible Telnet
   Brute Force Attack"; threshold:type both, track by_src, count 5,
   seconds 60; sid:100030; rev:1;)

alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"Ping detected";
   sid:100031; rev:1;)
```

ANNEXE C CONFIGURATION DES PLUGINS DE SORTIE SNORT

```
##################################################
# Step #6: Configure output plugins
# For more information, see Snort Manual, Configuring Snort - Output
    Modules
##################################################

# unified2
# Recommended for most installs
# output unified2: filename merged.log, limit 128, nostamp,
   mpls_event_types, vlan_event_types
output unified2: filename snort.log, limit 128, nostamp,
   mpls_event_types, vlan_event_types

# Additional configuration for specific types of installs
# output alert_unified2: filename snort.alert, limit 128, nostamp
output alert_unified2: filename snort.alert, limit 128, nostamp
# output log_unified2: filename snort.log, limit 128, nostamp

output alert_CSV:/dev/ttyPS0 timestamp,msg,proto,src,srcport,dst,
   dstport,tcpflags
#output alert_CSV:stdout timestamp,msg,proto,src,srcport,dst,dstport
   ,tcpflags

# syslog
# output alert_syslog: LOG_AUTH LOG_ALERT

# pcap
output log_tcpdump: tcpdump.log

# Fast alert logging for the daily cron script in Debian
output alert_fast: snort.alert.fast

# metadata reference data.  do not modify these lines
include classification.config
include reference.config
```