UNIVERSITÉ DE MONTRÉAL

DÉTECTION D'INTRUSION SUR LES OBJETS CONNECTÉS PAR ANALYSE COMPORTEMENTALE

ROBIN GASSAIS
DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
AOÛT 2018

© Robin Gassais, 2018.

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

DÉTECTION D'INTRUSION SUR LES OBJETS CONNECTÉS PAR ANALYSE COMPORTEMENTALE

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

Mme NICOLESCU Gabriela, Doctorat, présidente
M. DAGENAIS Michel, Ph. D., membre et directeur de recherche
M. FERNANDEZ José M., Ph. D., membre et codirecteur de recherche
M. BARRERA David, Ph. D., membre

DÉDICACE

À mes parents,
plus que ma réussite, je leur dois la vie...
À Sophie,
merci de m'avoir accompagné pendant ces deux magnifiques années au Canada ...

REMERCIEMENTS

Je tient tout d'abord à remercier le professeur Michel Dagenais qui a supervisé ce projet de recherche. La confiance et l'autonomie qu'il m'a accordées, son soutien constant, la qualité de son suivi et sa rapide compréhension des défis techniques auxquels j'ai été confronté m'ont été très précieux tout au long de cette maîtrise. Je le remercie des moyens matériels qu'il a su mettre rapidement à ma disposition lorsque j'en avais besoin.

Je remercie également le professeur José Fernandez, qui a co-dirigé ce projet de recherche. Sa capacité à vulgariser des concepts avancés de sécurité informatique et son expertise dans ce domaine m'ont grandement inspiré dans mon travail.

Je tiens également à remercier mes parents, qui m'ont permis de mener à bien mes études même à l'autre bout de la Terre. Ils m'ont toujours accompagné dans la réussite de mes études et ont mis en place un environnement propice à cela.

Je remercie également ma compagne, Sophie, qui a eu la patience de m'attendre parfois tard à l'école, qui m'a constamment soutenu durant le projet de recherche mais surtout qui a sû égayer ma vie pendant ces deux belles années.

Je souhaite également remercier chaleureusement mes collègues et amis du laboratoire DORSAL et du laboratoire SecSI. Ils m'ont donné de précieux conseils techniques et méthodologiques et ont grandement contribué à mon bien-être durant ces dernières années. Merci pour les parties de squash, de badbinton, de wallyball, merci pour les gâteaux et merci pour toutes ces merveilleuses pauses-café.

Je remercie également les enseignants qui ont sû attiser ma curiosité pendant mes études. Je n'ai sans doute pas beaucoup compter dans leur vie, mais eux ont certainement joué un rôle déterminant dans la mienne.

Enfin, je tiens à souligner la participation financière d'Ericsson, d'EfficiOS, du Conseil de recherches en sciences naturelles et en génie du Canada (CRSNG), de Prompt et de Ciena, qui m'ont permis de mener ce projet à bien.

RÉSUMÉ

Alors que la course à l'innovation des fabricants d'objets connectés s'accélère, de plus en plus d'attaques informatiques impliquant de tels objets, ou même les ciblant, sévissent. Ainsi, des campagnes de déni de service distribué comme celle de Mirai ont mis à mal des infrastructures informatiques gigantesques. En outre, le nombre de vulnérabilités découvertes dans l'écosystème des objets connectés ne cesse de grandir. La sécurité physique des utilisateurs d'objet connectés peut également être menacée par l'insécurité de ces plateformes. Dans le même temps, les solutions de sécurité proposées sont souvent spécifiques à un des nombreux protocoles de communication utilisés par les objets intelligents, ou s'appuient sur les caractéristiques matérielles et logicielles d'un type d'objet. De plus, peu de constructeurs permettent des mises à jour des micrologiciels de ces objets, augmentant ainsi les menaces contre les usagers de tels objets.

Dans ce contexte, il devient nécessaire de fournir des solutions de sécurité applicables à l'ensemble des objets proposés sur le marché. Cela nécessite de parvenir à collecter de l'information de manière efficace sur l'ensemble de ces systèmes et de réaliser une analyse automatique qui ne dépend pas de l'objet surveillé. Ainsi, différentes solutions ont été proposées, se basant essentiellement sur les informations réseau provenant des objets à surveiller, mais peu d'approches se basent sur le comportement même de l'objet.

En conséquence, nous proposons dans ce mémoire une solution de détection d'intrusion se basant sur les anomalies des objets surveillés. Les différents outils que nous avons développés permettent de collecter des informations relatives au comportement des objets surveillés de manière efficace et à différents niveaux, comme le mode usager ou directement dans le noyau du système d'exploitation. Pour ce faire, nous nous appuyons sur des techniques performantes de traçage. L'envoi des traces générées ainsi que leur traitement produira un jeu de données qui sera labellisé automatiquement. Ensuite, différents algorithmes d'apprentissage automatique permettront de détecter les anomalies sur le système de manière automatique et totalement indépendante du type d'objet surveillé.

Notre solution introduit très peu de baisse de performance sur les objets connectés surveillés, et montre d'excellents résultats pour détecter divers types d'attaques qui ont été implémentées durant les travaux de recherche. Différents algorithmes ont été étudiés, et les techniques à base d'arbre ont montré des résultats bien plus élevés que des réseaux de neurones profonds. De plus, les outils développés pendant ce projet de recherche permettent d'utiliser les librairies les plus populaires d'apprentissage automatique sur des traces au format CTF,

ouvrant ainsi la voie à la prédiction de performance d'un système ou à des analyses de traces plus automatiques et puissantes.

ABSTRACT

While vendors are creating more and more connected devices, the rate of cyberattacks involving or targeting such devices keeps increasing. For instance, some massive distributed denial of services campaigns such as Mirai used poorly secured devices to shut down popular services on the Internet for many hours or IoT malware like Brickerbot are regularly launched. The insecurity of smart devices create many threats for the users, and vulnerabilities on devices are disclosed every day, while only little vendors let their devices being updated. Meanwhile, proposed security solutions often failed at being compatible with all the devices, because of the numerous protocols used in the Internet of Things or the heterogeneity of firmware used in such devices.

This explains why it has become essential to creating a security solution for smart devices that are not specific to the kind of device. The first step to being able to protect a device is being able to detect an intrusion on it. This requires to collect data effectively from the monitored system to launch automated analysis that is not specific to the device. While several solutions matching those criteria have been proposed, most of them studied the network activity of the device, and only little focus on the device behavior.

As a consequence, we developed a solution using a device behavior to detect intrusion on it. We obtained very high detection performances with several attacks that have been implemented. Furthermore, we studied various classification algorithm to highlight that tree-based algorithm performed more than the other techniques, including recurrent deep neural network, with the data we collected effectively with tracing techniques. Moreover, we obtained very little overhead on the monitored device because of the architecture we developed. Finally, our tools also enable users to use traces in CTF binary format to feed the most popular Python machine learning libraries thanks to a whole toolchain of processing data.

TABLE DES MATIÈRES

DÉDICACE iii

REMERCIEMENTS iv

RÉSUMÉ v

ABSTRACT vii

TABLE DES MATIÈRES viii

LISTE DES TABLEAUX xi

LISTE DES FIGURES xii

LISTE DES SIGLES ET ABRÉVIATIONS xiii

CHAPITRE 1 INTRODUCTION 1
- 1.1 Définitions et concepts de base 1
  - 1.1.1 L'Internet des objets 1
  - 1.1.2 Intrusion 2
  - 1.1.3 Traçage logiciel 2
- 1.2 Éléments de la problématique 4
- 1.3 Objectifs de recherche 5
- 1.4 Plan du mémoire 5

CHAPITRE 2 REVUE CRITIQUE DE LA LITTÉRATURE 7
- 2.1 L'Internet des objets connectés dans les maisons 7
  - 2.1.1 La domotique 8
  - 2.1.2 Les protocoles de communication 8
  - 2.1.3 Les protocoles de communication à courte portée 9
  - 2.1.4 Les usages de la maison connectée 11
  - 2.1.5 L'architecture des maisons connectées 12
  - 2.1.6 Les risques de sécurité liés aux maisons connectées 13
- 2.2 Système de détection d'intrusion 16
  - 2.2.1 Détection par règle 19
  - 2.2.2 Détection d'anomalie 20

2.3 Le traçage 26
2.3.1 Traçage, débogage, journalisation et profilage 27
2.3.2 Analyse du mode utilisateur 28
2.3.3 Traçage du noyau 31
2.3.4 Visualisation de traces 35
2.4 Conclusion de la revue de la littérature 39

CHAPITRE 3 MÉTHODOLOGIE 42
3.1 Configuration matérielle 42
3.1.1 Objets connectés 42
3.1.2 Machine d'analyse 44
3.2 Environnement logiciel 44
3.2.1 Traçage 44
3.2.2 Apprentissage machine 45
3.3 Validation des résultats 45

CHAPITRE 4 ARTICLE 1 : MULTI-LEVEL HOST-BASED INTRUSION DETECTION SYSTEM FOR INTERNET OF THINGS 47
4.1 INTRODUCTION 48
4.2 Related Work 49
4.2.1 Smart Home and associated threats 49
4.2.2 Intrusion detection in IoT context 51
4.2.3 Tracing and debugging embedded devices 53
4.3 Proposed solution 54
4.3.1 Architecture 54
4.3.2 Trace collection 55
4.3.3 Data processing 56
4.3.4 Automated analysis 58
4.3.5 Alert raising 59
4.4 Use Case 60
4.4.1 Attacks 60
4.4.2 Resulting dataset 61
4.4.3 Learning methodology 63
4.4.4 Classification algorithm 64
4.5 Evaluation 65
4.5.1 Set up 65
4.5.2 Metrics 66

4.5.3 Tracing overhead 67
4.5.4 Training efficiency 68
4.5.5 Analysis efficiency 69
4.5.6 Limitation 71
4.6 Conclusion and Future Work 72
4.7 ACKNOWLEDGMENT 73

CHAPITRE 5 DISCUSSION GÉNÉRALE 74
5.1 Retour sur les résultats 74
5.1.1 Jeux de données générés 74
5.1.2 Apprentissage machine 75
5.1.3 Efficacité du traçage 75
5.2 Scénarios d'attaque 75

CHAPITRE 6 CONCLUSION ET RECOMMENDATIONS 77
6.1 Synthèse des travaux 77
6.2 Limitations de la solution proposée 78
6.3 Améliorations futures 79

RÉFÉRENCES 80

LISTE DES TABLEAUX

Tableau 3.1 Caractéristiques matérielles du RPi3 43
Table 4.1 Post processing feature list 59
Table 4.2 Enabled tracepoints list 62
Table 4.3 Machine learning algorithm used 64
Table 4.4 CPU overhead according to snapshot frequency 68
Table 4.5 Memory overhead according to snapshot requency 68
Table 4.6 Anomaly detection efficiency of various algorithm 69
Table 4.7 Analysis latency 71

LISTE DES FIGURES

Figure 2.1 Architecture typique d'une maison connectée 14
Figure 2.2 Arbre de décision produit à la suite d'un apprentissage 24
Figure 2.3 Résultat du traçage de ls avec Strace 30
Figure 2.4 Visualisation de trace à l'aide de Babeltrace 36
Figure 2.5 Analyse LTTng sur la latence des entrées/sorties 37
Figure 2.6 Analyse LTTng sur les statistiques d'utilisation des appels systèmes . 38
Figure 2.7 Control Flow View de Trace Compass 39
Figure 2.8 Vue des ressources du système dans Trace Compass 40
Figure 2.9 Vue de statistiques propres au système tracé dans Trace Compass . . 40
Figure 4.1 Smart home architecture 51
Figure 4.2 General Solution Architecture 55
Figure 4.3 Data processing architecture 57
Figure 4.4 Dataset event repartition 63
Figure 4.5 Relevant fields of processed events 70

LISTE DES SIGLES ET ABRÉVIATIONS

IDS	Intrusion Detection System
HIDS	Host-Based Intrusion Detection System
NIDS	Network-Based Intrusion Detection System
IoT	Internet of Things
CTF	Common Trace Format
DT	Decision Tree
RF	Random Forest
GBT	Gradient Boosted Tree
SVM	Support Vector Machine
CTF	Common Trace Format
MLP	Multilayer Perceptron
LSTM	Long Short Term Memory

CHAPITRE 1 INTRODUCTION

Il n'existe pas de système informatique impénétrable, et les attaques sur ces plateformes ne cessent d'augmenter en nombre et en intensité à chaque jour. Dans le même temps, de plus en plus de systèmes sont interconnectés, et ce phénomène s'accentue avec l'avènement de l'Internet des objets, qui est composé d'objets innovants mais n'intégrant que très rarement des protections efficaces contre les attaques des cybercriminels. Elles menacent leurs utilisateurs, qu'ils soient des individus ou des entreprises. Il est donc nécessaire de trouver des solutions capables de sécuriser efficacement ces objets et respectant leurs contraintes matérielles et logicielles.

L'une des premières étapes pour améliorer la sécurité de tels dispositifs est d'être en mesure de détecter les tentatives d'attaque dès qu'elles surviennent. Pour cela, il est nécessaire de s'intéresser aux attaques passées impliquant des objets connectés mais également d'être en mesure de prévenir de nouveaux types d'attaques.

Ce mémoire propose donc une solution capable de détecter de nombreux types d'intrusions sur des objets connectés, en utilisant des techniques performantes de traçage pour recueillir des informations précises sur les systèmes surveillés et en comparant diverses techniques d'apprentissage machine pour parvenir à d'excellentes capacités de détection.

1.1 Définitions et concepts de base

Dans cette section, nous définissons les concepts essentiels à la compréhension du mémoire.

1.1.1 L'Internet des objets

L'Internet des objets connectés fait référence à l'interconnexion entre des objets intelligents et les réseaux informatiques traditionnels comme l'Internet.

Les objets connectés sont des systèmes embarqués qui peuvent prendre différentes formes. Ainsi, dans les maisons, il est désormais possible de trouver des ampoules connectées capables d'adapter leur lumière à la requête des habitants, des enceintes intelligentes capables de répondre aux questions et aux ordres de leurs possesseurs et même des jouets pouvant retransmettre aux parents l'image de leur enfant en train de les utiliser. Ces objets sont capables d'effectuer diverses actions et tendent à se répandre rapidement dans les maisons.

Dans la suite du mémoire, nous réfèrerons à ces objets sous les noms d'objets connectés ou

d'objets intelligents.

1.1.2 Intrusion

On parle d'intrusion lorsqu'une personne pénètre dans un espace qui lui est normalement interdit d'accès. Cet espace peut être physique ou logique. Dans ce mémoire, nous réfèrerons à intrusion toute pénétration dans un système informatique ayant pour but de mettre à mal sa confidentialité, son intégrité ou sa disponibilité.

La détection d'intrusion rassemble toutes les techniques mises en oeuvre pour alerter les utilisateurs du système informatique visé par une intrusion sur le fait qu'ils sont en train d'être ciblés par un attaquant.

1.1.3 Traçage logiciel

Le traçage logiciel, appelé traçage dans la suite du mémoire, est un ensemble de techniques mises en oeuvre qui visent à enregistrer des informations précises sur l'exécution d'un programme ou d'un système. Il peut se faire dans l'espace utilisateur ou au niveau du noyau du système d'exploitation. De plus, il est possible de combiner le traçage dans ces deux modes pour avoir des informations détaillées et précises du fonctionnement du système surveillé. En outre, le traçage ne doit pas modifier le comportement du système lorsqu'il est activé, avec un impact minimal en termes de mémoire ou de temps d'exécution par rapport au comportement sans traçage.

Les applications du traçage sont diverses. Il peut être utilisé pour détecter des problèmes survenus sur un système, pour comprendre la source du goulot d'étranglement lors de l'exécution d'un programme, et ainsi permettre son optimisation, ou encore pour détecter des anomalies sur le système surveillé.

Pour enregistrer les informations, le traçage **instrumente** l'exécution d'un programme ou du noyau du système d'exploitation de manière statique ou dynamique à l'aide de **points de trace**. Lorsque ces derniers seront rencontrés, des **évènements** seront produits et le **traceur** les enregistrera de manière temporellement ordonnée dans une trace.

Instrumentation

L'instrumentation d'un programme consiste à modifier son exécution en y ajoutant des instructions supplémentaires. Le but de l'instrumentation est généralement l'étude ou la mesure du fonctionnement du programme.

Point de trace

Un point de trace est un bout de code ajouté à l'exécution du programme tracé. Lorsqu'il est exécuté, il fait un appel au traceur afin que ce dernier enregistre l'évènement associé au point de trace.

Évènement

Un évènement contient une information sur le système au moment de l'exécution où le point de trace qui lui est associé a été rencontré. Il peut contenir diverses informations mais est toujours estampillé temporellement et contient généralement une information relative au système lorsqu'il a été généré, comme le nom de la fonction où était présent le point de trace.

Traceur

Le traceur est le logiciel chargé d'enregistrer les évènements lorsqu'ils sont créés et de les ordonner temporellement dans une trace

1.2 Éléments de la problématique

La forte concurrence des constructeurs d'objets connectés a créé un marché où de nombreuses technologies se côtoient. Ces objets peuvent prendre des formes diverses, allant des systèmes d'alarmes ou de gestion de la lumière à des fours ou à des jouets connectés. Ils ont des ressources parfois extrêmement limitées, ne permettant pas de mettre en place des solutions de sécurité traditionnelles. La diversité des protocoles existant dans cet écosystème ainsi que l'absence de système d'exploitation ou de micrologiciel commun à ces plateformes rendent les travaux visant à sécuriser de tels objets très dépendants des systèmes étudiés. De plus, peu de constructeurs proposent des mises à jour pour leurs dispositifs, rendant les vulnérabilités découvertes très dangereuses pour les utilisateurs. Ainsi, il devient difficile de proposer des solutions de sécurité applicables à l'ensemble des objets, et les travaux effectués en termes de détection d'intrusion se concentrent essentiellement sur des analyses du réseau, puisqu'il n'y a pas besoin de considérer les ressources matérielles des objets ni même les applications qui sont exécutées dessus.

De plus, bien que les techniques de traçage aient déjà été utilisées pour détecter des anomalies sur différents systèmes, particulièrement avec l'aide des appels système effectués sur la machine surveillée, peu de travaux se sont intéressés à l'ensemble des informations disponibles dans les évènements. Ainsi, les arguments des appels système ou les différentes informations au niveau des paquets réseau ou du processeur n'ont presque pas été utilisées pour détecter des attaques informatiques, alors qu'elles peuvent se révéler précieuses pour améliorer la qualité de la détection. Par exemple, un évènement lié au lancement d'un terminal peut être bénin ou néfaste pour le système selon le processus qui est à l'origine de cet appel système. Il convient donc d'utiliser l'ensemble des informations relevées par les points de trace pour améliorer la qualité de la détection.

En outre, peu de travaux ont proposé des suites d'outils permettant d'exploiter des traces dans un format binaire pour entrainer des algorithmes d'apprentissage machine. En effet, pour parvenir à un tel résultat, plusieurs étapes de transformation des évènements recueillis avec des techniques de traçage doivent être mises en place. Les défis pour parvenir à ce résultat sont multiples. Le premier est de déterminer les champs des évènements utiles pour entrainer les algorithmes d'apprentissage machines, qui peuvent être très différents entre les divers évènements enregistrables par le traceur. Un autre défi est de récolter ces champs de manière efficace depuis les formats où sont stockées les traces, particulièrement les formats binaires, et de créer d'autres métriques utiles à l'apprentissage des algorithmes. Il est aussi nécessaire de convertir toutes ces données en des vecteurs de chiffres qui seront alors utilisables pour l'apprentissage des modèles.

Enfin, il est nécessaire d'étudier l'efficacité des différentes méthodes d'apprentissage machine développées pour améliorer les capacités de détection de la solution proposée. En effet, chaque algorithme peut donner des résultats différents en fonction du jeu de données qu'il reçoit ou de l'utilisation du modèle une fois son apprentissage terminé. Par exemple, certains algorithmes peuvent être les plus performants pour classifier des séquences de texte mais peuvent être beaucoup moins optimaux pour classifier des mots seuls. Il convient ainsi d'étudier de manière critique le comportement de différentes techniques d'apprentissage dans le cas de la détection d'intrusion sur des objets connectés, avec des traces relatives au système lui-même.

1.3 Objectifs de recherche

La principale problématique de ce mémoire est la suivante :

Est-il possible de mettre en oeuvre un détecteur d'intrusion basé sur le comportement des objets connectés efficace et reposant sur des techniques de traçage et d'apprentissage machine ?

Pour parvenir à répondre à cette problématique, il est nécessaire de réaliser les objectifs suivants :

1. Comprendre les caractéristiques de l'écosystème des objets connectés.
2. Développer une architecture permettant le traçage des objets intelligents en tenant compte de leurs caractéristiques.
3. Mettre au point un *framework* capable de transformer les informations contenues dans les traces en informations précises utilisables par des algorithmes d'apprentissage machine.
4. Implémenter et optimiser divers algorithmes d'apprentissage machine capables de classifier les évènements enregistrés.
5. Déployer un système domotique de test en identifiant les architectures classiques des maisons intelligentes.
6. Identifier et implémenter des attaques sur les objets connectés étudiés.
7. Tester la performance de la solution et des différents algorithmes sur ce système.

1.4 Plan du mémoire

Le chapitre 2 du mémoire présente une revue critique de la littérature sur les différents concepts utiles lors de ce travail. Les sujets de l'Internet des objets, des systèmes de détection

d'intrusion et des différentes techniques de traçage seront abordés. Ensuite, le chapitre 3 présentera la méthodologie adoptée dans nos travaux pour mettre en place et valider la solution proposée. Le chapitre 4 présente l'article dans lequel la solution développée est présentée ainsi que la critique de ses performances. Le chapitre suivant présent évalue le succès de réponse à la problématique de recherche. La conclusion du travail ainsi que la discussion des travaux futurs sont présentées dans le chapitre 6.

CHAPITRE 2 REVUE CRITIQUE DE LA LITTÉRATURE

L'Internet n'a jamais connecté autant de systèmes informatiques qu'aujourd'hui, et le nombre d'objets capables de communiquer entre eux par ce réseau ne cesse d'augmenter à chaque jour. Cette tendance tend même à s'accélérer avec l'ajout toujours plus rapide de nouveaux objets qui ne cessent de se diversifier. Bien que la concurrence féroce entre les concepteurs d'objets apporte beaucoup d'innovation, elle oblige également les fabricants à diminuer la durée de développement de leurs produits et donc à réduire la phase de test de sécurité des objets mis sur le marché.

Ainsi, de nombreuses vulnérabilités sont découvertes sur les objets intelligents et les risques associés ne cessent de grandir pour les utilisateurs. Il devient donc nécessaire d'être en mesure de sécuriser ces différents objets, la première étape étant d'être en mesure de détecter ces activités malveillantes. Bien que différents travaux aient été proposés, beaucoup de solutions restent inefficaces ou très spécifiques à un objet.

La revue critique de la littérature qui suit souligne les différentes approches proposées pour détecter des intrusions ainsi que différentes techniques qui pourraient être utilisées pour parvenir à de meilleurs résultats.

2.1 L'Internet des objets connectés dans les maisons

L'Internet des objets connectés, ou IDO (IoT en anglais) fait référence à l'interconnexion de nombreux systèmes embarqués entre eux et avec l'informatique traditionnels. Il y a ainsi une convergence entre les anciens réseaux d'information, l'infonuagique et les nouveaux objets capables de communiquer à travers la planète. Notre travail portant principalement sur les objets connectés dans les maisons, souvent appelés système domotique, la revue de littérature détaillera surtout ces aspects. Il est cependant à noter que de nombreux autres travaux ont été proposés dans ce domaine, que ce soit au niveau des villes connectées (Hollands, 2008), (Nam and Pardo, 2011) ou au niveau des entreprises (Lee and Lee, 2015). De plus, des projets de grande ampleur sont menés dans plusieurs grandes villes du monde comme Toronto avec l'implication de Google, Dublin ou encore Amsterdam qui ont des politiques d'extension rapide, et qui augmentent grandement les interactions entre les objets connectés, les véhicules, les habitants et l'Internet mondial.

Les objets connectés sont essentiellement des systèmes embarqués (Wortmann and Flüchter, 2015) qui ont par conséquent des ressources limitées, en termes de puissance de calcul, d'es-

pace de stockage et même d'alimentation puisque certains fonctionnent à l'aide de batteries.

2.1.1 La domotique

La domotique, ou *smart home*, fait référence à une maison dans laquelle des objets connectés communiquent entre eux, ou avec les habitants. Ainsi, la maison connectée s'inscrit dans la tendance d'expansion des objets connectés et a pleinement sa place dans les projets de smart cities lancés un peu partout à travers la planète. De plus, Gubbi explique que les maisons seront les premiers lieux d'émergence des objets connectés (Gubbi et al., 2013), avant qu'ils ne fassent leur apparition à l'échelle des entreprises ou des villes. Il indique également que ces objets seront utiles pour augmenter le confort des utilisateurs, avoir un meilleur contrôle sur leur consommation énergétique et même améliorer leur santé au travers de divers dispositifs. En outre, un rapport publié par Joshfire sur les *smart home* en 2016 estime que le marché des objets connectés génèrera 7 100 milliards de dollars en 2020, alors que Cisco juge qu'à cette époque, il y aura 50 milliards d'objets intelligents en fonctionnement (Hui et al., 2017).

On ne constate actuellement pas de convergence sur les technologies utilisées pour ces objets dans les maisons, que ce soit au niveau logiciel, matériel ou même au niveau des protocoles de communications. Ainsi, de nombreux constructeurs proposent des solutions variées, parfois compatibles avec des objets d'autres fabricants. Cette disparité matérielle est cependant nuancée par la très grande présence des processeurs ARM qui sont efficaces pour effectuer des calculs, tout en consommant peu de ressources, et par les architectures MIPS qui ont été traditionnellement utilisées dans les systèmes embarqués et qui proposent des performances intéressantes.

De même, les disparités entre les différents firmwares et les différents systèmes d'exploitation sont bien présentes, mais beaucoup de ces logiciels sont basés sur des distributions très légères de Linux. Ainsi, des constructeurs comme Vera s'appuient sur des versions modifiées d'OpenWRT, une distribution Linux très employée dans les routeurs réseau. Cependant, très peu de firmwares ont leur code source libre, et plusieurs constructeurs ont créé leur propre firmware afin d'optimiser la performance de leurs appareils.

Enfin, l'hétérogénéité des différents protocoles de communication, qu'ils soient de longue ou de faible portée, est soulignée dans les paragraphes suivants.

2.1.2 Les protocoles de communication

Alaba a expliqué que les objets connectés créent de nouveaux modes de communications : après les échanges entre les ordinateurs, puis entre les ordinateurs et les humains ou les

humains et les humains, les nouveaux échanges d'information se font entre les objets et les humains et même entre les objets eux-mêmes (Alaba et al., 2017). C'est l'interconnexion de ces objets à cette fin que l'on appelle l'Internet des Objets, ou IoT pour Internet of Things.

Nous pouvons distinguer deux catégories de protocoles de communication utilisés par ces appareils : les protocoles à longue portée, comme le NB-IoT, le LoRAWAN, le Sigfox, les réseaux cellulaires classiques (GSM, 3G, 4G, 5G), et les protocoles à plus courte portée, telle que le Zigbee, le Z-Wave, le 6LoWPAN, le BLE (Bluetooth Low Energy). Puisque nous nous intéressons aux objets connectés dans les maisons, nous détaillerons plus les protocoles de courte portée dans la suite de cette revue de littérature.

Protocoles longue portée

Comme expliqué précédemment, les protocoles longue portée les plus populaires sont le LoRaWAN, le NB-IoT et le SigFox, ainsi que les réseaux cellulaires déjà implémentés et leurs nouvelles versions.

Le LoRa (acronyme de *Long Range*) est un protocole développé par Semtec et qui se base sur une couche physique nommée LoRA. On distingue 3 classes différentes de LoRaWAN d'après (WPL), la classe A (les noeuds parlent à la passerelle dès qu'ils en ont besoin), la classe B (les noeuds écoutent à intervalle régulier dans un fenêtre de temps) et la classe C (les noeuds écoutent en tout temps). Cela permet de nombreuses applications de ce protocole.

Le NB-IoT, pour Narrowband IoT est issu d'une initiative de 3GPP, un organisme qui développe des standards de protocole de télécommunication. Il fonctionne sur le réseau 10MHz LTE et possède différentes implémentations.

Enfin, le SigFox autorise des échanges entre les objets connectés avec de très faibles bandes passantes. Il fonctionne sur les bandes 868 MHz en Europe et 915 MHz en Amérique du Nord, a une portée allant jusqu'à 50 km dans les zones rurales et 10 km dans les zones urbaines. Il fonctionne grâce à des messages de très petite taille (12 octets pour le payload), consommant ainsi très peu de ressources d'énergie et autorisant les communications maître à esclave ou bidirectionnelles (RFW).

2.1.3 Les protocoles de communication à courte portée

Le protocole Z-wave

Z-wave est un protocole développé en 2004 par la société Zensys, fonctionnant sur le principe d'un réseau maillé, où les noeuds du réseau peuvent également s'occuper du routage de la

communication. Ce protocole est supporté par la Z-Wave alliance, un consortium de plus de 300 entreprises qui se sont engagées à développer des produits compatibles Z-Wave et intercompatibles entre eux. De ce fait, ce protocole tend à s'installer comme le protocole leader des objets intelligents au sein de la maison et de plus en plus d'objets sont compatibles avec ce standard.

Bien qu'en théorie ce protocole supporte jusqu'à 232 noeuds dans un même réseau, des travaux affirment que des problèmes semblent survenir lorsqu'on connecte plus de 40 noeuds (WPH). Ce document explique également que tous les équipements Z-Waves ne sont pas compatibles entre eux, puisque la rétrocompatibilité n'est pas assurée entre les anciennes versions de Z-wave et les nouvelles, notamment en raison des standards de sécurité qui ont évolué.

En ce qui concerne ce dernier point, le Z-Wave repose sur un chiffrement AES 128 bits ; seuls les puces Z-Wave supérieurs à la série 400 proposent ce chiffrement (Romera, 2017). Ce livre blanc incite ainsi tous les propriétaires de puces des gammes 200 et 300 à les remplacer pour assurer la sécurité de leur réseau, ainsi qu'à changer régulièrement les clés de chiffrement utilisées.

De plus, le protocole Z-Wave a été élaboré pour faciliter la communication des objets connectés dans les maisons (Zarpelão et al., 2017). Il fonctionne sur les fréquences 908.45 MHz en Amérique du Nord et 868.42 MHz en Europe. La bande passante du signal est de 40 kb/s et deux noeuds consécutifs du réseau doivent être espacés de 30 mètres maximum. Cette relativement faible distance n'est pas une barrière à l'adoption du protocole dans une maison, puisque l'on a de multiples objets connectés dans différentes pièces du logement.

Les objets Z-Waves appartiennent à deux catégories : les contrôleurs et les esclaves. Les contrôleurs peuvent donner des ordres et modifier le comportement des esclaves et s'occupent du routage de l'information, les esclaves pouvant alors renvoyer des informations au contrôleur s'ils possèdent des capteurs.

Fouladi explique qu'il est possible de distinguer quatre couches pour établir une communication Z-wave : la couche physique, composée de transmetteurs et récepteurs radiofréquences, la couche de transport, la couche de routage, avec la gestion du réseau maillé, et la couche applicative avec les différentes classes de commandes Z-Wave (Fouladi and Ghanoun, 2013).

Il existe une implémentation Open Source du protocole, nommée Open Z-Wave, mais l'implémentation de la sécurité n'est pas encore prise en charge. De plus, de nombreux modules permettent de transformer les architectures classiques utilisées en domotique en des architectures compatibles avec le protocole. Par exemple, les Raspberri Pi (Foundation, 3) peuvent

transmettre des signaux Z-Wave à l'aide d'une carte compatible nommée Razberry (Fendall et al., 2015), des modèles similaires étant compatibles avec les cartes Arduino.

Le protocole Zigbee

Le protocole ZigBee repose également sur un réseau maillé, mais est implémenté sur les bandes de fréquences 2,4GHz, comme le Wi-Fi ou certaines bandes du Bluetooth. Le zigbee a une spécification basée sur les standards IEE 802.15.4 et est supporté par la Zigbee Alliance.

Comme le Z-wave, le ZigBee est compatible avec de très nombreux appareils, et sa popularité en fait le concurrent direct du Z-Wave. De plus, ce protocole offre des débits théoriques plus importants que le Z-Wave. De plus en plus de constructeurs rendent leurs produits compatibles avec ces deux standards, à l'instar de Samsung avec sa gamme *SmartThings*.

Le protocole 6LoWPAN

Le protocole 6LoWPAN, pour *IPv6 Low power Wireless Persona Area Network*, est une adaptation des protocoles IPv4 et IPv6 pour les communications impliquant des objets connectés. Il a été mis au point par l'IETF (*Internet Engineering Task Force*) dans le but d'être plus "léger" que les protocoles IP standards. Fonctionnant également sur un modèle de réseau maillé, il supporte pleinement UDP et TCP.

Mulligan affirme que les en-têtes des paquets sont très légers (2 à 11 octets) et peuvent permettre la communication entre 2^{64} noeuds (Mulligan, 2007).

De plus la majorité des travaux sur les IDS dans les objets connectés se sont concentrés sur ce protocole (Zarpelão et al., 2017). À l'instar du ZigBee, le 6LoWPAN fonctionne sur la bande de fréquence de 2,4 GHz, rendant son intégration plus facile en raison des équipements actuels.

Ainsi, l'hétérogénéité des objets connectés au sein des maisons rend les travaux visant à proposer des solutions génériques de sécurisation des objets connectés dans ce contexte bien plus difficile.

2.1.4 Les usages de la maison connectée

L'apparition des objets connectés a créé et continuera de créer de nouveaux besoins et de nouveaux usages. Les principaux usages des objets connectés dans les maisons connectées sont le divertissement, la surveillance et la sécurité, la gestion des ressources et de l'énergie, l'aide au ménage, ainsi que la gestion des lumières et de la santé (WPU).

Cependant, les auteurs ont clairement identifié les causes d'insécurité de tels objets. En effet, il est expliqué que la plupart des vulnérabilités découvertes sur ces nouveaux équipements sont dues au design même des produits, au recours à des protocoles de communication qui ne sont pas sécurisés ou à des méthodes d'authentification inadaptées. En outre, les mauvaises implémentations ou utilisations des objets connectés seraient une cause majeure des vulnérabilités découvertes selon les auteurs.

De plus, le livre blanc souligne le manque de mise à jour ou de correctifs disponibles pour le logiciel embarqué. Padilla et al. ont fortement insisté sur la nécessité de pouvoir mettre à jour les objets connectés (Acosta Padilla et al., 2016a). En plus de proposer des stratégies de mise à jour adaptées aux objets connectés, l'auteur explique qu'il serait judicieux de pouvoir mettre à jour le logiciel embarqué morceau par morceau, plutôt qu'une mise à jour globale de ce logiciel. En outre, il est nécessaire de pouvoir vérifier l'intégrité de ces nouvelles installations.

Selon Fouladi, l'un des enjeux de la domotique est de protéger la confidentialité et l'intégrité des communications au sein et à l'extérieur de la maison (Fouladi and Ghanoun, 2013). En conséquence, des techniques de protection efficaces doivent être implémentées. Ainsi, pendant l'échange de clés cryptographiques, les objets connectés peuvent diminuer leur puissance de transmission pour rendre l'interception de paquets plus difficile et ainsi empêcher un attaquant d'obtenir ces informations cruciales en écoutant les transmissions hors de la maison. Cependant, la sécurité de certains objets semble encore insuffisante, puisque les auteurs affirment que dans certains objets, les clés utiles pour les chiffrements sont écrites directement dans la mémoire des objets, et sans protection de chiffrement particulière.

2.1.5 L'architecture des maisons connectées

Comme expliqué précédemment, les protocoles de communication des objets intelligents sont différents des protocoles utilisés sur l'Internet traditionnel. En conséquence, il est nécessaire d'avoir un objet faisant le lien entre les protocoles des maisons connectées et les protocoles TCP/IP, les plus répandus sur Internet. C'est le rôle du contrôleur domotique, qui a également pour tâche de lancer des actions en fonction des informations qu'il reçoit des différents capteurs présents dans la maison. Il est possible de considérer cet objet comme le "cerveau" du système domotique.

Ainsi, une architecture classique dans une maison connectée est présentée dans la figure 4.1. Dans cette configuration, les objets connectés cohabitent dans le réseau domestique avec les appareils traditionnels, comme les ordinateurs ou les téléphones intelligents. Cependant, certains auteurs préconisent de fractionner le réseau des objets intelligents selon leur utilité.

C'est le cas de Cheng, qui propose une architecture où chaque objet est partitionné selon le protocole de communication qu'il utilise (Cheng et al.). Selon ses critères, il faudrait distinguer les objets utilisant le Bluetooth des objets communicants via Z-Wave. Cependant, certains objets, typiquement des caméras connectées, reçoivent des ordres via des messages Z-Wave, mais peuvent transmettre des données (ici les données vidéos) sur du Wi-Fi traditionnel avec le protocole IP.

2.1.6 Les risques de sécurité liés aux maisons connectées

Le développement rapide du marché des objets connectés force les constructeurs à proposer rapidement aux consommateurs de nombreux produits innovants. Ainsi, les considérations de sécurité des nouveaux objets mis en commercialisation ne sont pas suffisamment explorées, ce qui crée de plus grands risques pour les utilisateurs. De plus, avec l'hétérogénéité des solutions proposées, il est compliqué pour les chercheurs en sécurité d'adresser les problèmes de sécurité pour l'ensemble des objets intelligents.

Conti effectue la distinction entre les challenges de sécurité liés aux objets connectés et ceux liés à leur investigation numérique (Conti et al., 2018). Ainsi, l'auteur explique qu'il est nécessaire d'assurer une authentification, une autorisation et un contrôle des accès. Il est également nécessaire de sécuriser la vie privée des utilisateurs. Selon lui, cela doit se faire avec une architecture sécuritaire. En ce qui concerne l'investigation numérique,il affirme que les enjeux liés à l'identification, la cueillette et la conservation des preuves sont différents des systèmes informatiques traditionnels. Il est nécessaire de réfléchir à ces problématiques dans le cadre des objets connectés pour pouvoir analyser les preuves, les corréler et être en mesure de retracer des attaques survenues.

Ainsi, de nombreux auteurs s'inquiètent du manque de sécurité général des objets mis sur le marché et des menaces que cela introduit chez les utilisateurs. Que ce soit dans les articles de Gubbi (Gubbi et al., 2013) ou d'Hui (Hui et al., 2017), la sécurité des objets et le respect de la vie privée des utilisateurs apparaît comme étant un critère fondamental pour l'adoption de ces objets dans les foyers. Le dernier auteur identifie des dangers liés à l'usage de ces objets, alors que Babar effectue une taxonomie de menaces associées à l'émergence de l'Internet des Objects (Babar et al., 2010). Il identifie cinq secteurs de dangers des objets intelligents :

— Gestion de l'identification : ce domaine regroupe l'ensemble des techniques sécuritaires qui doivent être mises en place pour identifier de manière unique les objets, les utilisateurs ou les sessions. Il est nécessaire, selon l'auteur, de pouvoir garantir un niveau de sécurité élevé des méthodes d'authentification, d'autorisation et de gestion de compte.
— Sécurité des communications : ce principe regroupe l'ensemble des mesures visant à

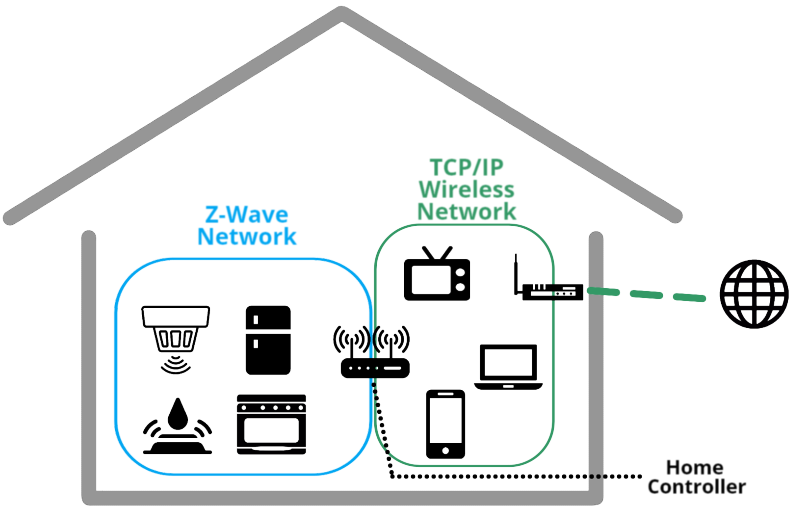

Figure 2.1 Architecture typique d'une maison connectée

assurer la disponibilité des communications. Nous aurions pu rajouter à cela l'intégrité de ces communications, qui ne devraient pas pouvoir être modifiées, ainsi que la confidentialité de celles-ci pour éviter de mettre à risque la vie privée des utilisateurs.

— Menaces physiques : dans cette catégorie, l'auteur explique que l'accès physique aux objets est la plupart du temps très facile, et que cela permet à des attaquants d'effectuer plus facilement des attaques comme de la rétro-ingénierie du système ou des attaques de microprobing permettant de retrouver des fichiers chiffrés (Skorobogatov, 2017). L'auteur n'explique pas que les objets connectés peuvent compromettre la sécurité physique de leurs utilisateurs, ce point étant détaillé ci-après.

— Sécurité des systèmes embarqués : l'auteur affirme que les systèmes embarqués seront sujets à des attaques similaires aux systèmes traditionnels aux niveaux physique et MAC. Par exemple, les attaques par canaux auxiliaires ou contre les environnements sécuritaires pourront être conduites contre ces objets.

— Gestion du stockage d'information : ce principe est indispensable dans la gestion de clés cryptographiques. Il est également important d'assurer l'intégrité et la confidentialité des données présentes sur les objets. Différents travaux ont été proposés pour assurer la sécurité du stockage de données dans le cas des objets connectés, comme celui de Babar qui propose un *framework* global de sécurisation de tels systèmes (Babar et al., 2011).

Peu d'articles traitent des risques introduits par les objets connectés sur les utilisateurs, que ça soit leur sécurité physique ou le respect de leur vie privée.

Ainsi, l'interconnexion de plus en plus d'objets comme des fours connectés, des systèmes d'alarme ou de détection de fumée intelligents, ou même des systèmes de contrôle du gaz capables d'être actionnés à distance se démocratisent. Cependant, si un attaquant obtient un contrôle suffisant sur ces objets, par exemple en acquérant l'accès total au contrôleur domotique, il lui serait alors possible d'ouvrir l'arrivée de gaz puis d'allumer le four brutalement, pouvant ainsi causer une explosion et des dégâts physiques à l'habitat ou à ses habitants.

De même, le respect de la vie privée des usagers des objets intelligents passe par la sécurisation des informations personnelles, comme les pistes audio ou les photos enregistrées par des caméras. Il est également nécessaire de protéger les informations émises par les différents capteurs pouvant se trouver dans les maisons (capteurs de température, de mouvement, de luminosité) puisqu'elles pourraient permettre à un habitant de connaitre les habitudes des habitants de la maison ou même de déterminer leurs activités.

Ces menaces sont fondées sur les différentes vagues d'attaques impliquant des objets connectés. Saxena identifie ainsi plusieurs attaques ciblant les objets intelligents (Saxena et al.), alors que Sikder s'intéresse aux attaques ciblant spécifiquement les capteurs dans les réseaux d'objets connectés (Sikder et al., 2018). De même, Babar effectue une taxonomie des attaques rencontrées (Babar et al., 2010) et Kozlov montre différents scénarios d'attaques en fonction des architectures développées (Kozlov et al., 2012). Nous pouvons résumer les différents travaux de ces auteurs selon les différents principes suivants :

— l'exfiltration de données, dans le but d'espionner les utilisateurs, de voler des informations aux habitants ou de les modifier. Par exemple, l'attaquant pourrait vouloir effectuer du chantage sur la victime, lui voler des documents secrets pour les revendre ou modifier des documents pour tromper sa cible.
— les attaques sur les messages réseau : l'attaquant désire modifier le comportement du système domotique ou même le détruire, pour ce faire il peut envoyer de fausses informations à des objets connectés (que ça soit des données de capteurs erronées ou des ordres à des actionneurs), effectuer des attaques par rejeu où il va capturer des commandes envoyées sur le réseau et les relancer ou utiliser des techniques de personne du milieu.
— l'indisponibilité des systèmes : l'utilisateur malveillant vise ici à rendre inaccessible un système à ses utilisateurs. Ce système peut être un objet connecté en lui-même ou un serveur distant. Un exemple d'une telle attaque où la cible était un objet intelligent est le malware Brickerbot (Wagner et al., 2017), qui a pour but de supprimer tous les fichiers de l'objet qu'il a infecté et de corrompre la mémoire de sorte qu'il ne soit pas en mesure d'être réparé. Une vague d'infection célèbre ciblant un serveur distant est

la campagne du botnet Mirai (Kolias et al., 2017), qui infectait des objets intelligents en tentant des mots de passe par défaut. Une fois les objets infectés, ils pouvaient être contrôlés via un serveur distant pour lancer des tentatives de déni de service distribué contre des cibles distantes, comme les serveurs du fournisseur DNS DYN ou de l'hébergeur OVH. Cela peut même créer des risques financiers pour les possesseurs d'objets connectés, avec des vagues de rançongiciels forçant les consommateurs à payer une rançon pour pouvoir déchiffrer les logiciels de leurs objets infectés et ainsi être en mesure de les utiliser à nouveau.

— l'intrusion dans un réseau : que ça soit un réseau domestique ou d'entreprise, les objets connectés qui y appartiennent sont souvent moins protégés que les systèmes traditionnels, permettant à des attaquants d'avoir sous leur contrôle un premier équipement d'un tel réseau pour pouvoir ensuite en cibler d'autres. Ainsi, des techniques de pivotage peuvent être réalisées une fois qu'un objet est contrôlé, permettant parfois de passer outre des protections réseau comme des pare-feu externes.

2.2 Système de détection d'intrusion

Pour pallier à toutes ces menaces, de nombreux travaux ont tenté de trouver des moyens efficaces pour détecter les intrusions dès qu'elles ont lieu. De tels dispositifs sont nommés IDS, pour Intrusion Detection System, et peuvent repérer des attaques à l'aide de traces réseau, il s'agit alors d'un NIDS (Network-based IDS), ou à l'aide d'information sur le système surveillé, c'est alors un HIDS (Host-based IDS). Les IDS ont été étudiés depuis longtemps, comme l'explique Debar dans sa taxonomie sur les systèmes de détection d'intrusion Debar et al. (1999), et ces systèmes ont été déployés dans tous les types de réseaux comme le montre Anantvalee avec les réseaux mobiles ad hoc (Anantvalee and Wu, 2007).

Zarpelao affirme que les IDS traditionnels ne sont pas adaptés pour l'IoT car les objets ont des ressources limitées qui rendent impossible l'utilisation de technologies de détection d'intrusion trop lourdes. De plus, les protocoles utilisés par ces objets sont nombreux et leur sont spécifiques, et les objets évoluent souvent dans des réseaux maillés, étant à la fois une source de données et un système capable de rediriger les données qu'il reçoit (Zarpelão et al., 2017). Ce sont ces contraintes qui ont amené à de nouvelles recherches sur les IDS dans le contexte des objets intelligents. Cependant, la plupart de celles-ci sont basées sur des informations réseau (NIDS), comme l'explique Sabahi dans sa revue de littérature sur le sujet (Sabahi and Movaghar, 2008). Cela peut être expliqué par le fait que ces informations peuvent être aisément obtenues sans avoir à modifier l'objet surveillé, ce qui peut être compliqué avec des systèmes embarqués voire même diminuer leur performance.

Sabahi explique qu'il y a deux types de détecteurs d'intrusion : ceux basés sur des règles à propos du système, et ceux basés sur la découverte d'anomalies du système (Sabahi and Movaghar, 2008). L'utilisation d'un système expert, qui est un des moyens de réaliser de la détection d'intrusion avec des règles, peut reposer sur des signatures connues de menaces, sur des règles concernant les données collectées ou sur les transitions d'état du système. Ce type de détecteur d'intrusion a l'avantage d'être précis et de très bien fonctionner sur des menaces connues (Zarpelão et al., 2017), mais il est incapable de détecter de nouvelles menaces, contrairement à la deuxième technique.

En effet, les IDS basés sur les anomalies surveillent le comportement du système qu'ils protègent, et tentent de détecter toute variation suspecte de ce comportement. Ils requièrent ainsi de pouvoir caractériser précisément le comportement du système qu'ils surveillent, ce qui peut être parfois compliqué et nécessiter beaucoup de ressources. Les différentes techniques de détection d'anomalies sont explicitées dans la sous-section suivante. Sabahi souligne également que des méthodes hybrides, utilisant à la fois des données issues du trafic réseau et des caractéristiques de l'hôte surveillé, ont été proposées.

L'architecture des détecteurs d'intrusion peut également grandement varier, selon le type de système surveillé, les ressources nécessaires à la détection de l'intrusion ou les données utilisées. Ainsi, Sabahi explique que les IDS centralisés sont les plus courants, mais des architectures distribuées et hybrides ont été proposées. Dans une architecture centralisée, le système qui est surveillé est celui qui est chargé de détecter les anomalies, par exemple dans le cas de SNORT (Roesch et al., 1999) où l'analyse des trames réseau se fait sur un ou des noeuds du réseau, qui sont les systèmes qui vont collecter les informations nécessaires à la détection d'intrusion. En revanche, les architectures distribuées reposent essentiellement sur des agents qui vont aller collecter les informations sur divers hôtes et acheminer cette information jusqu'à un autre système qui se chargera de la détection. Un exemple d'un tel système est DIDMA, la solution proposée par Kannadiga, qui est constituée d'agents statiques sur les hôtes à surveiller, qui transmettent des informations à des agents mobiles à l'aide d'un répartiteur d'agent (Kannadiga and Zulkernine, 2005). Ces agents mobiles transmettent alors les informations à des agents d'alerte sur la console IDS qui va alors lever les alertes lorsque c'est nécessaire.

Comme l'ont souligné les divers articles, les détecteurs d'intrusion sont d'autant plus efficaces que les moyens de collecter les informations dont ils ont besoin le sont. Ainsi, plusieurs travaux sur les HIDS se sont appuyés sur les appels système pour détecter des comportements malveillants, comme les travaux de Hofmeyr (Hofmeyr et al., 1998) ou de Kosoresow (Kosoresow and Hofmeyer, 1997). Cependant, plusieurs articles ont montré que ces informations

n'étaient pas suffisantes pour empêcher les attaquants de réaliser des intrusions sans être remarqués.

Ainsi, les travaux de Wagner montrent comment il est possible d'évader les détecteurs d'intrusion reposant sur des appels systèmes, et particulièrement lorsque les arguments de ces fonctions noyau ne sont pas pris en compte par le détecteur d'intrusion (Wagner and Soto, 2002). Sa méthode consiste à imiter le comportement normal d'un système en y effectuant des actions illégitimes. De même, Ma souligne le fait qu'il est possible d'échapper à la détection de la plupart des HIDS en utilisant des processus fils qui effectueront peu d'actions, mais que la somme de ces actions peut mener à des attaques évoluées (Ma et al., 2012).

Un des moyens utilisés pour acquérir les informations nécessaires à la détection est l'utilisation de techniques de traçage, qui ne permettent pas seulement d'avoir la liste des appels système effectués, mais également leurs arguments et des informations à différents niveaux, lorsque l'on trace le noyau, comme des informations réseau, la séquence des actions de l'ordonnanceur ou même des informations sur l'utilisation du processeur. D'ailleurs, Murtaza explique que les traces au niveau du noyau permettent d'acquérir beaucoup plus d'informations utiles pour la détection d'intrusion (Murtaza et al., 2012a). Les techniques de traçage sont détaillées dans la section correspondante.

Cependant, les usages des systèmes de détection d'intrusion ont évolué avec l'arrivée de nouvelles applications de l'informatique. En ce qui concerne l'infonuagique, Modi décrit de manière très précise les différentes techniques de détection d'intrusion pouvant être employées dans ce contexte (Modi et al., 2013). L'auteur explique que les systèmes traditionnels (HIDS et NIDS) peuvent être utilisés, mais que de nouveaux systèmes sont apparus. C'est le cas des systèmes de détection d'intrusion distribués (DIDS) qui sont composés de plusieurs IDS placés dans un large réseau et qui communiquent entre eux (que ce soit avec un serveur central ou directement entre eux). Ces systèmes peuvent combiner à la fois des HIDS et des NIDS. De plus, des systèmes de détection d'intrusion basés sur les hyperviseurs des machines virtuelles ont également été développés (Bharadwaja et al., 2011). De tels systèmes permettent ainsi de surveiller les échanges d'information entre les machines virtuelles, entre les machines virtuelles et l'hyperviseur et même avec l'hyperviseur du réseau virtuel, d'après l'auteur.

De même, la démocratisation des téléphones intelligents, et particulièrement du système d'exploitation Android, a créé de nouvelles menaces pour les utilisateurs. Cela a justifié différents travaux sur la détection d'intrusion sur de tels dispositifs. Ainsi, Borek a énuméré les différentes techniques de détection d'intrusion employées pour les objets exécutant Android. Il explique ainsi que plusieurs travaux détectent des applications illégitimes d'après leur code

source, comme DREBIN (Arp et al., 2014) qui peut être utilisé directement sur le téléphone ou sur un serveur, DroidAPIMiner (Aafer et al., 2013) ou encore DroidAnalytics (Zheng et al., 2013). De plus, des techniques d'analyse dynamiques ont également été proposées, qu'elles soient basées sur des informations réseau comme Credroid (Malik and Kaushal, 2016) ou sur des informations issues du comportement de l'objet (Dini et al., 2012). De plus, comme les IDS traditionels, certains travaux utilisent des techniques de traçage pour obtenir les appels système du téléphone et ainsi être en mesure de détecter des intrusions. C'est le cas de la solution proposée par Borek, mais également de Copperdroid (Tam et al., 2015) et de Crowdroid (Burguera and Nadjm-Tehrani, 2011). Ces deux derniers travaux s'appuient respectivement sur une émulation du système Android avec Qemu et sur des évènements tracés directement sur le téléphone, mais envoyés à un serveur pour leur analyse.

2.2.1 Détection par règle

De nombreux travaux ont été proposés pour détecter des intrusions à l'aide de règles. Ainsi, des techniques utilisant des réseaux de neurones artificiels, de la reconnaissance de patron ou même des outils permettant d'identifier des intrusions dans des bases de données ont été proposés.

Canady a proposé d'utiliser des réseaux de neurones artificiels pour détecter des attaques au niveau du réseau (Cannady, 1998). Ce choix est justifié par le fait que la mise en place d'un système expert pour atteindre ce but requiert une connaissance précise des méthodes d'attaque des réseaux et qu'il peut être difficile d'identifier des intrusions dont la méthode varie un peu de celles connues. De plus, l'auteur explique que les systèmes experts ont besoin de nombreuses mise à jour pour rester efficaces contre les menaces émergentes, et qu'il est nécessaire de surveiller les transitions d'états du système plutôt que l'arrivée d'évènements précis dans le cas de détection par règle. Canady a montré que les réseaux de neurones artificiels permettent d'être plus flexible que les systèmes experts pour la détection d'intrusion réseau, puisqu'ils peuvent fonctionner même avec des données incomplètes, qu'ils sont rapides et qu'ils peuvent apprendre des attaques précédemment analysées et identifiées. Cependant, l'auteur explique que cette phase d'apprentissage peut être difficile, puisqu'elle requiert de très nombreuses données qui peuvent être difficiles à obtenir, et qu'il est compliqué de comprendre le fonctionnement exact du modèle entrainé.

Kumar s'est intéressé à la reconnaissance de patrons de certaines attaques pour reconnaitre des intrusions (Kumar and Spafford, 1994). L'article énumère différentes techniques de reconnaissance de patron qui sont applicables à la détection d'attaques, et explique que ces méthodes fonctionnent correctement pour détecter des intrusions sur les systèmes UNIX.

Cependant, il est nécessaire d'avoir identifié les attaques pour en extraire les patrons, et une variante d'une attaque pourrait conduire à des séquences d'évènements différents, rendant la détection inefficace.

De plus, des systèmes de détection d'intrusion par règles ont été proposés pour protéger les bases de données (Chung et al., 2000). Ce travail propose d'utiliser les fichiers de journalisation des bases de données surveillées pour créer un profil propre au fonctionnement de la base de données. Dès lors, si ce profil tend à changer, une potentielle intrusion peut être levée. L'article propose une architecture capable de traiter les données de journalisation de la base de données, de créer un profil associé et d'entrainer un modèle capable de détecter les déviations de ce modèle. Le profil est créé en fonction de diverses métriques, comme la fréquence des items considérés ou les relations contenues dans la base de données.

2.2.2 Détection d'anomalie

Chandola définit la détection d'anomalie comme un ensemble de techniques dont le but est de détecter des comportements inhabituels sur des systèmes (Chandola et al., 2009). Tsai explique que cela peut se faire avec des techniques d'apprentissage machine (Tsai et al., 2009). Plusieurs algorithmes peuvent être implémentés à cet effet, et plusieurs stratégies peuvent être employées, que ça soit de la classification de données, du clustering ou l'utilisation de métriques et de statistiques sur le système surveillé. Murtaza explique que la détection d'anomalie en utilisant du traçage est plus efficace avec des informations au niveau du noyau qu'avec uniquement des informations au niveau de l'espace utilisateur (Murtaza et al., 2012b). En effet, le traçage noyau permet d'obtenir beaucoup plus d'information sur l'état du système, avec des informations reliées aux processus en cours d'exécution comme les appels système et les différentes actions de l'ordonnanceur, mais également des informations au niveau des composants physiques du système comme l'utilisation du processeur ou même des informations réseau, avec la queue des paquets qui arrivent ou qui vont être émis. Dans un autre travail, le même auteur utilise trois algorithmes différents pour surveiller les anomalies d'un système : le Kernel State Model, un algorithme de reconnaissance de séquence et un algorithme de chaine de Markov caché. Les trois algorithmes reposent sur les appels système qui sont tracés sur la machine surveillée (Murtaza et al., 2014a).

Cependant, il est très important de différencier les différentes stratégies précédemment énoncées, comme le montrent les paragraphes suivants.

Statistiques

Il est possible d'utiliser des statistiques sur le système à surveiller pour détecter des anomalies lorsqu'elles sont facilement récoltables et qu'il n'est pas difficile de caractériser le comportement normal du système. En effet, si l'on connait le taux d'utilisation moyen de la mémoire vive et du processeur d'un ordinateur, ainsi que l'occupation classique de son disque dur, et que ces paramètres n'évoluent pas beaucoup au cours du temps, alors un brusque changement dans une de ces valeurs pourrait indiquer qu'une anomalie survient sur le système.

En revanche, s'il est très compliqué ou coûteux de recueillir ces informations, avec des systèmes aux ressources très limitées par exemple, ou si ces valeurs varient beaucoup sans réelle tendance, alors l'utilisation de statistiques pour surveiller l'activité du système peut être difficile.

Ainsi, Sekar utilise des machines à états sur les paquets réseau pour détecter des anomalies sur les systèmes surveillés. Il se repose sur des statistiques dont les fréquences de transition des états pour lever des alertes (Sekar et al., 2002). Dans ce contexte, l'information statistique est facilement accessible, et le travail semble fournir d'excellents résultats pour détecter des anomalies.

Groupement

Le but des techniques de groupement (*clustering* en anglais), est de trouver des relations entre les données, de sorte à créer des groupes de données similaires. Dans le cas du traçage, si l'on collecte des évènements réseau, des évènements liés aux processeurs et des appels système, les techniques de clustering pourraient permettre de placer les évènements reçus dans l'une des trois catégories précédentes. Cependant, nous pourrions imaginer créer des groupes au sein même des appels système, avec par exemple les appels d'ouverture de fichier, qui contiennent le nom du fichier ouvert en argument, et ceux ne contenant pas une telle information.

En ce qui concerne la détection d'anomalies, le but des techniques de clustering est de réussir à créer des groupes suffisamment précis pour englober l'ensemble des évènements liés à un comportement sain, mais pour ne pas y incorporer des évènements liés à une intrusion. Ainsi, des nouveaux évènements étant classifiés à l'extérieur des groupes appris seront susceptibles de lever une alerte.

Les techniques de *clustering* sont des techniques d'apprentissage non supervisé, puisqu'il n'est pas nécessaire d'associer une classe aux évènements en entrée de ces algorithmes, qui vont se charger eux-mêmes d'attribuer un groupe aux évènements qu'ils doivent traiter.

Comme l'a souligné Jain, plusieurs techniques de clustering ont été proposées afin de ca-

tégoriser au mieux les données. Ainsi, l'auteur distingue les algorithmes hiérarchiques des algorithmes partitionnaires et détaille les techniques basées sur les plus proches voisins (Jain et al., 1999). Les algorithmes hiérarchiques séparent les données sous forme de dendogramme, où les données sont regroupées suivant une hiérarchie. Par exemple, si une donnée appartient à un groupe A et à un groupe B, le groupe A étant inclus dans le groupe B, alors l'algorithme sera en mesure de retransmettre cette information. Les algorithmes partitionaires cherchent quant à eux à séparer l'espace des données en différentes partitions pour créer des catégories de données semblables. Un algorithme connu de cette catégorie est le *simple-linkage* (Gower and Ross, 1969) ou sa variante, le *complete-linkage* (Wilks, 2011). Les techniques hiérarchiques ont l'avantage de produire des sorties facilement compréhensibles (les dendogrammes peuvent être lus par des humains rapidement), mais le coût nécessaire pour produire ces diagrammes peut rapidement augmenter avec la taille des données.

Un des algorithmes partitionaires les plus étudiés est l'algorithme des K-moyens (K-means), ou le centre des groupements de données créées est recalculé à chaque fois qu'un nouvel élément est rajouté au groupe. Plusieurs variantes ont été implémentées avec divers critères d'arrêt. De même, différentes stratégies ont été élaborées pour assigner les premiers centres spatiaux des groupes ou déterminer le nombre de groupements à produire. L'appartenance d'une donnée de l'algorithme à un groupe est souvent déterminée par la distance entre cette donnée et le centre de chaque groupement.

Classification

Les algorithmes de classification sont utilisés dans le but d'associer une classe à chaque donnée en entrée de l'algorithme. Il est d'abord nécessaire que les modèles apprennent avec des données appartenant à chaque classe et étant labellisées, il s'agit donc d'apprentissage machine supervisé, bien que l'on puisse classifier des informations à l'aide de techniques de groupement. Par exemple, si l'on a créé des groupes contenant tous les évènements normaux d'un système, lorsqu'un nouvel évènement se produira et sera trop loin du centre des groupes, d'après des critères définis par l'utilisateur, alors on pourra classifier cet évènement comme étant anormal.

Dans le cadre de la détection d'anomalie, le but des techniques de classification est de déterminer si l'information analysée relève d'un comportement normal ou d'une anomalie. Cette classification peut se faire à l'échelle d'un évènement ou d'une séquence d'évènement. Chandola a identifié différentes techniques pour classifier des séquences, dont celles basées sur des noyaux, celles basées sur des fenêtres et celles basées sur des modèles de Markov.

De nombreux algorithmes de classification ont été proposés dans la littérature, nous donnons

ici une rapide revue du fonctionnement général des algorithmes que nous avons utilisés lors de nos travaux ainsi que des caractéristiques qui leur sont propres.

Les arbres de décision

Les arbres de décision sont une famille d'algorithmes qui utilisent une structure d'arbre pour classifier une entrée selon ses caractéristiques (Safavian and Landgrebe, 1991). Ainsi, des noeuds sont créés afin de maximiser l'entropie des ensembles, créée par la distinction des deux branches ou la pureté de ces ensembles. Cela permet de maximiser le gain d'information. En conséquence, les arbres de décision sont capables d'extraire des règles permettant de discriminer les données. Un exemple d'arbre de décision produit à la suite d'un apprentissage est présenté à la figure 2.2.

Ce type d'algorithme a l'avantage de pouvoir être très facilement compréhensible par les humains, puisqu'il peut produire un arbre expliquant les critères utilisés pour séparer les entrées. De plus, les arbres de décisions nécessitent très peu de préparation des données, puisqu'ils peuvent classifier des valeurs numériques ou des catégories, ne requièrent pas de normalisation et sont robustes à des données hétérogènes (qui n'ont pas les mêmes caractéristiques). Cependant, la découverte de l'arbre de décision optimal est un problème NP-complet (Laurent and Rivest, 1976) et les modèles entrainés peuvent être sujets au surapprentissage.

Les forêts d'arbres décisionnels

Les forêts d'arbres décisionnels font partie des techniques d'ensemble dites de *bagging*, ou l'on construit plusieurs classificateurs sur des sous-ensembles des données. Cette technique a été proposée par Svetnik à des fins de recherche dans le domaine du biomédical (Svetnik et al., 2003). La classification finale se fait par vote des différents classificateurs construits, qui sont ici des arbres de décision. Cet algorithme combine ainsi les avantages et les inconvénients des arbres de décision, mais permet de pallier aux difficultés rencontrées par les arbres de décision lorsqu'il est nécessaire de classifier des données avec un grand nombre de catégories, puisque les arbres ont été créés sur des sous-ensembles.

De nombreux travaux se sont intéressés aux performances de cet algorithme, et les auteurs s'accordent sur la très bonne précision de cette technique (Rodriguez-Galiano et al., 2012), (Pal, 2005), (Ham et al., 2005), (Díaz-Uriarte and De Andres, 2006), bien que des améliorations aient été proposées pour réduire ses problèmes de biais (Strobl et al., 2007).

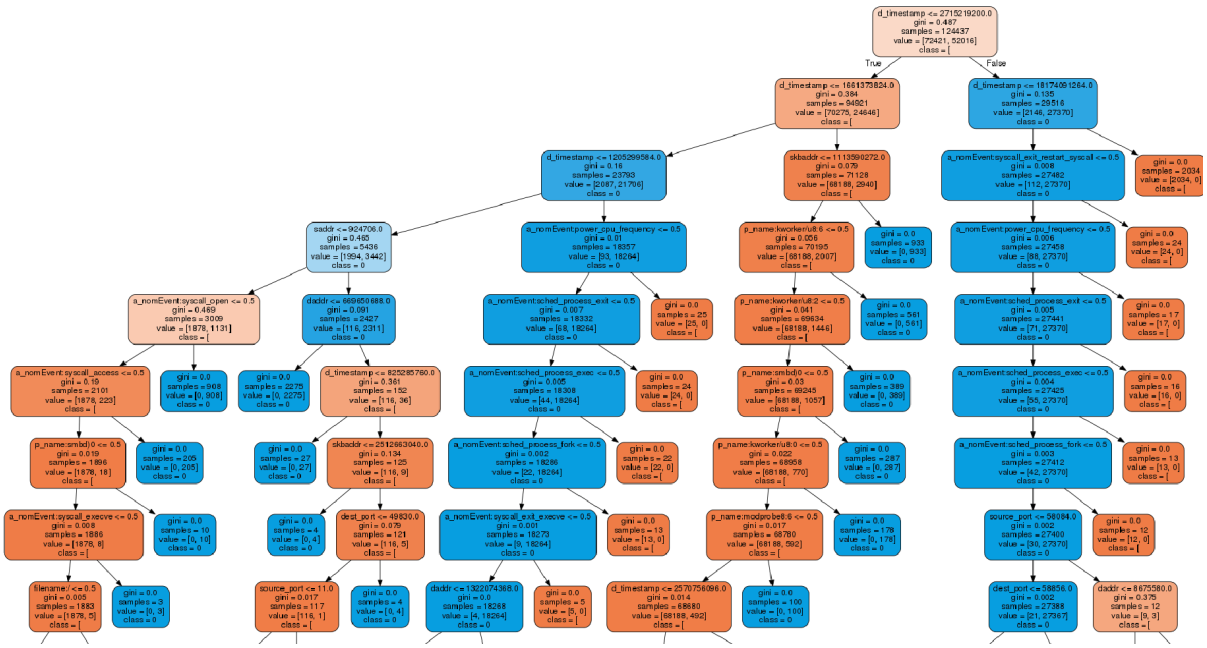

Figure 2.2 Arbre de décision produit à la suite d'un apprentissage

Les Gradient Boosted Trees

Le *Gradient Boosted Tree* fait également partie des classificateurs à ensemble, mais appartient à la famille des algorithmes de *boosting*. Il s'agit toujours de combiner l'efficacité de plusieurs classificateurs, encore une fois des arbres de décision, mais les classificateurs sont construits de manière séquentielle : à chaque itération le nouveau classificateur introduit doit réduire le biais et la variance du classificateur global (Ganjisaffar et al., 2011). Cette technique permet de corriger les erreurs de classification au fur et à mesure des itérations, et ainsi de mieux prédire les valeurs difficiles. Nous avons choisi le gradient comme critère de mesure de la performance des classificateurs ajoutés à chaque itération, mais d'autres métriques auraient pu être considérées.

Les excellentes performances de précision de cet algorithme ont été soulignées dans différents travaux comme ceux de Friedman qui détaille la théorie mathématique de cette technique (Friedman, 2002) ou ceux de Elith qui donne des instructions détaillées sur le bon usage de cette technique (Elith et al., 2008).

Les machines à support de vecteur

Les machines à support de vecteur sont un ensemble de techniques où le classificateur cherche à déterminer la frontière entre les données appartenant à une classe et les autres (**?**). Dans notre cas, il s'agirait de trouver la frontière entre les évènements légitimes du système et les évènements liés à une anomalie. Pour déterminer plus facilement cette frontière, les machines à vecteur de support vont projeter les données dans un espace de plus grande dimension où elles seront linéairement séparables, à l'aide de la fonction noyau qui a été implémentée. Cela nécessite parfois de modifier les données utilisées pour l'apprentissage et la classification, comme l'explique Hsu dans son guide pratique sur l'utilisation des SVMs (Hsu et al., 2003).

Les machines à support de vecteurs ont été utilisées dans diverses applications comme le traitement de texte (Joachims, 1998), la génétique (Brown et al., 2000) ou même dans le cas de données complexes à analyser (Vapnik and Mukherjee, 2000).

La machine à support de vecteur à une classe

L'algorithme de machine à vecteur de support à une classe fonctionne de la même manière que l'algorithme précédent, mais l'apprentissage ne se fait que sur une seule classe, ici sur les évènements correspondants à un comportement normal (Manevitz and Yousef, 2001). Ainsi, l'algorithme va apprendre à projeter les données dans un espace où il saura reconnaitre la frontière autour des évènements légitimes. Lorsqu'un évènement sera hors de cette frontière, une fois projeté dans le même espace, l'algorithme sera en mesure de détecter cette anomalie. Cette méthode permet donc de détecter des attaques qui n'ont pas été apprises lors de la phase d'entrainement, ce qui la rend particulièrement intéressante, contrairement aux autres techniques d'apprentissage énumérées.

Différents travaux ont été réalisés pour améliorer l'efficacité de cette technique comme ceux d'Erfani qui propose d'utiliser des techniques d'apprentissage machine profond pour améliorer la détection d'anomalies avec des données de haute dimensionnalité (Erfani et al., 2016), ou ceux de Li qui a détaillé quelles caractéristiques des jeux de données d'intrusions réseau utiliser pour améliorer les performances de détection des machines à support de vecteur à une classe (Li et al., 2003).

Le perceptron multicouche

Le perceptron multicouche est un réseau de neurones artificiels. Il est composé de plusieurs couches de neurones et les données sont envoyées d'un neurone d'une couche à un ou des neurones de la couche suivante (Pal and Mitra, 1992). Les neurones effectuent des opérations

sur l'entrée qu'ils reçoivent à l'aide de fonctions d'activation. Les informations sont ainsi filtrées de nombreuses fois et le résultat permet de classifier les données de l'algorithme.

Les perceptrons multicouches ont été étudiés depuis longtemps et peuvent permettre de classifier de manière efficace, comme l'a montré Atlas dans un article dans lequel il conclut que les performances d'un perceptron multicouche entrainé égalent ou dépassent celles d'arbres de décisions (Atlas et al., 1990). Cependant, Belue a montré qu'il était important de sélectionner les bonnes caractéristiques des données de l'algorithme pour obtenir des résultats cohérents (Belue and Bauer Jr, 1995).

L'algorithme LSTM

L'algorithme LSTM (pour *Long-Short Term Memory* en anglais) est un réseau de neurones artificiels récurrents, c'est-à-dire qu'il est composé de plusieurs neurones qui ont des connexions récurrentes. Il a été proposé par Hochreiter en 1997 (Hochreiter and Schmidhuber, 1997) . Ainsi, à l'aide d'une fonction d'activation, les cellules qui composent ce réseau sont capables de se rappeler d'informations déjà analysées afin de pondérer la sortie de l'information en cours d'analyse. Cela permet à l'algorithme de classifier des séquences d'évènements plutôt que les évènements en eux-mêmes. Cependant, il est nécessaire de fournir beaucoup de données à l'algorithme pour que son apprentissage soit convenable. Les ressources nécessaires aux calculs sont relativement élevées comparativement aux autres techniques décrites précédemment.

Des améliorations de l'architecture originale des réseaux LSTMs ont été proposées par Tai en utilisant des structures d'arbres avec ces réseaux (Tai et al., 2015). De même, Sak et son équipe ont réussi à rendre possible l'apprentissage de ces réseaux sur des systèmes distribués, améliorant ainsi les performances d'apprentissage des réseaux LSTMs qui sont parfois très demandants en ressources (Sak et al., 2014).

L'ensemble de ces algorithmes a besoin d'une quantité de données de qualité pour effectuer la phase d'apprentissage et générer des modèles capables de prédire la présence ou non d'une anomalie, comme le souligne TSAI en mettant l'accent sur l'importance de la phase de sélection des caractéristiques des données (Tsai et al., 2009).

2.3 Le traçage

Le but du traçage d'un système est d'obtenir des informations précises sur son état de fonctionnement. Ces informations peuvent être matérielles ou logicielles, au niveau du noyau ou du mode utilisateur, et sont généralement datées avec précision à l'aide d'estampilles de

temps. Elles sont typiquement précises à la nanoseconde et nécessitent des compteurs 64 bits pour être enregistrées. Pour obtenir ces informations, le traçage se repose sur des évènements qui vont être capturés lorsque le système atteint certains états. Leur collecte étant rapide elle ne modifie que très peu le comportement du système. Les évènements sont produits à l'aide de points de trace, qui peuvent être ajoutés statiquement à une application ou de manière dynamique lors de son exécution. Les points de trace sont généralement des petits bouts de code qui ont pour fonction de faire un appel, contenant des informations sur l'état du système, au moment où ils sont rencontrés par le processeur.

Le traceur est le logiciel chargé de collecter les évènements à mesure qu'ils surviennent et de les stocker de manière ordonnée dans ce que l'on appelle une trace. Il doit également peu perturber le système pendant son fonctionnement, afin que les informations collectées soient pertinentes.

Il est important de distinguer le traçage du débogage, de la journalisation ou du profilage, comme nous le montrons dans les paragraphes suivants.

2.3.1 Traçage, débogage, journalisation et profilage

Le but des techniques de traçage, de débogage, de logging et de profilage est similaire : il s'agit de présenter à l'utilisateur des informations sur le système surveillé afin qu'il ait une meilleure compréhension de son comportement. Cependant, ces outils diffèrent par plusieurs aspects.

La journalisation ressemble au traçage dans le sens où le but est également d'enregistrer des évènements survenant dans un système. En revanche, les données enregistrées sont de très haut niveau, contrairement aux évènements du traçage qui peuvent être de très bas niveau, avec des informations au niveau du noyau du système d'exploitation. De plus, la fréquence d'enregistrement dans la journalisation est très faible, comparativement à celle du traçage, et les informations recueillies sont typiquement les erreurs en sortie de programme. Ainsi, les outils de journalisation sont relativement peu efficaces pour enregistrer des évènements fréquents, et l'on peut considérer les traceurs comme des outils de journalisation optimisés et capables de récolter de nombreuses informations.

En ce qui concerne les outils de profilage, leur but est de fournir aux utilisateurs des informations générales sur le système, ainsi que des statistiques qui lui sont associées (Patel and Rajwat, 2013). Ainsi, le profilage ne tient pas compte des évènements qui surviennent pendant le temps où le système est surveillé, mais présente à l'utilisateur des moyennes correspondant au comportement ayant eu lieu pendant ce temps. Ainsi, le nombre de résultats présentés à la

suite du profilage d'une application n'augmente pas lorsque le temps de profilage augmente, contrairement au traçage.

Le débogage a également un but différent de celui du traçage : comprendre les raisons des erreurs survenues lors de l'exécution d'un programme (Shapiro, 1982). Ainsi, les techniques de débogage sont utilisées à des fins de développement et permettent de montrer à l'utilisateur des messages précis, utiles pour comprendre les causes d'erreur du système. De son côté, le traçage peut-être utilisé dans des cycles de production pour comprendre le comportement du système, avec des évènements ordonnés précisément et un coût d'utilisation très faible. L'utilisation d'un débogueur peut également se faire à des fins de ré-ingénieurie.

2.3.2 Analyse du mode utilisateur

Tel qu'expliqué précédemment, il est possible de tracer des systèmes dans les modes noyau ou utilisateur. Différentes techniques ont été élaborées pour y parvenir.

Valgrind

Valgrind est un outil de profilage très utilisé par les développeurs. Il s'agit d'un *framework* d'instrumentation binaire dynamique, qui permet de fournir beaucoup d'information à ses utilisateurs sur le fonctionnement d'un programme. Il est ainsi très utilisé pour vérifier les allocations dynamiques de mémoire, les lectures et écritures ou encore même les fautes de cache avec son outil Callgrind (Weidendorfer, 2008).

Ce profileur ne peut cependant pas tracer le noyau, il n'est donc pas possible de suivre les appels systèmes à l'aide de Valgrind. De plus, les informations obtenues grâce à cet outil ne sont pas estampillées, et sont souvent résumées en statistiques.

Pour fonctionner, Valgrind émule le code profilé dans une machine virtuelle avec différents outils. Bien que cette technique permette de repérer de nombreux problèmes difficiles à identifier autrement, elle ralentit considérablement les performances du programme, ce qui peut empêcher son utilisation dans des environnements de production.

GBD

GDB est un débogueur fréquemment utilisé par les développeurs. Cet outil permet également de tracer dynamiquement une application en mode utilisateur à l'aide de point d'arrêts qui vont sauver des informations dans un tampon de trace lorsqu'ils seront rencontrés au cours de l'exécution (Shebs, 2009). Cependant, cette technique de traçage n'est pas très

performante puisqu'elle provoque un changement de contexte à chaque fois qu'un point de trace est enregistré.

Une autre technique développée par cet outil pour tracer un programme utilise des points de traces rapides, où une instruction est remplacée par un saut d'exécution vers le code d'instrumentation. Cette technique permet de tracer dynamiquement le code d'un programme de manière relativement efficace, mais GDB ne possède qu'un seul tampon de trace contrôlé par un verrou, ce qui ralentit fortement les exécutions de programmes parallélisés (Shebs, 2009).

Strace

Strace est un traceur puissant, capable de présenter les appels système effectués par un programme à son utilisateur, comme l'explique Gregg (Gregg). Ce traceur est proposé nativement dans de nombreuses distributions Linux et permet également de faire du profilage de programme, puisqu'il peut fournir des statistiques sur le temps passé par un programme dans la section noyau. Le résultat du traçage du programme *ls* de Linux est présenté à la figure 2.3.

Cependant, Strace n'utilise qu'un seul fichier par processus en sortie, le rendant très peu efficace sur des programmes parallélisés, puisqu'il a recours à un verrou pour permettre l'écriture des informations récoltées. De plus, la performance de Strace est relativement mauvaise, puisqu'il repose sur *ptrace* pour tracer les appels système, ce qui introduit deux appels système supplémentaires à chaque appel système tracé. Ceci peut causer des ralentissements au programme de plus de 100 fois son temps d'exécution normal (Gregg).

Extrae

Extrae est un ensemble d'outils permettant le traçage de nombreuses applications comme MPI, OpenMP, CUDA ou encore OpenCL sur de nombreuses plateformes telles que des groupes de machine Linux, des cartes graphiques nVidia, des processeurs ARM ou des systèmes Android (Center). Plusieurs techniques ont été développées par le laboratoire de calcul de haute performance de Barcelone pour parvenir à ce résultat, comme l'utilisation de Dyninst pour insérer des points de trace de manière dynamique, ou le recours à des injections de librairies d'instrumentation avec la directive LD_Preload.

```
robin@station14:~/Software/trace-compass$ strace ls -la
execve("/bin/ls", ["ls", "-la"], [/* 54 vars */]) = 0
brk(NULL)                               = 0x5562a625c000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fef25c00000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=144924, ...}) = 0
mmap(NULL, 144924, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fef25bdc000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/x86_64-linux-gnu/libselinux.so.1", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0000k\0\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0644, st_size=155400, ...}) = 0
mmap(NULL, 2259664, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fef257b8000
mprotect(0x7fef257dd000, 2093056, PROT_NONE) = 0
mmap(0x7fef259dc000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x24000) = 0x7fef259dc000
mmap(0x7fef259de000, 6864, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fef259de000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\4\2\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=1689360, ...}) = 0
mmap(NULL, 3795296, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fef25419000
mprotect(0x7fef255ae000, 2097152, PROT_NONE) = 0
mmap(0x7fef257ae000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x195000) = 0x7fef257ae000
mmap(0x7fef257b4000, 14688, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fef257b4000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/x86_64-linux-gnu/libpcre.so.3", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\320\25\0\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0644, st_size=468920, ...}) = 0
mmap(NULL, 2564360, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fef251a6000
mprotect(0x7fef25218000, 2093056, PROT_NONE) = 0
mmap(0x7fef25417000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x71000) = 0x7fef25417000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/x86_64-linux-gnu/libdl.so.2", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\200\r\0\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0644, st_size=14640, ...}) = 0
mmap(NULL, 2109680, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fef24fa2000
mprotect(0x7fef24fa5000, 2093056, PROT_NONE) = 0
mmap(0x7fef251a4000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x7fef251a4000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/x86_64-linux-gnu/libpthread.so.0", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0Pa\0\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=135440, ...}) = 0
mmap(NULL, 2212936, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fef24d85000
mprotect(0x7fef24d9d000, 2093056, PROT_NONE) = 0
mmap(0x7fef24f9c000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x17000) = 0x7fef24f9c000
mmap(0x7fef24f9e000, 13384, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fef24f9e000
close(3)                                = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fef25bda000
arch_prctl(ARCH_SET_FS, 0x7fef25bdb3c0) = 0
```

Figure 2.3 Résultat du traçage de ls avec Strace

LTTng-UST

LTTng-UST est le traceur du mode utilisateur associé au traceur noyau LTTng, détaillé dans la suite de la revue de littérature. Il utilise la même architecture que LTTng et repose sur des points de trace insérés à des endroits arbitraires du code, qui vont envoyer au traceur des informations sur l'état du système, lorsqu'ils seront rencontrés à l'exécution du programme instrumenté. Pour ce faire, le programme est dynamiquement lié lors de son exécution à la librairie LTTng-ust.

Ce traceur est particulièrement performant, puisqu'il alloue un tampon circulaire pour chaque coeur d'exécution, permettant d'instrumenter les programmes à exécution parallélisée sans pénaliser les différents fis d'exécution (Fournier et al., 2009). De plus, les tampons sont gérés grâce à des opérations atomiques et sont partagés avec un processus d'arrière-plan responsable de transférer les données enregistrées sur le disque ou de les envoyer à travers le réseau, et de vider les tampons avant qu'ils ne soient remplis. Cependant, les tampons peuvent se remplir trop vite, si le débit du réseau ou du disque par exemple est inférieur à celui de génération des événements. Par défaut, LTTng accepte de perdre des évènements plutôt que de bloquer le processus en attente du disque ou du réseau, afin de ne pas ralentir l'exécution du programme.

Grâce à différents modules développés, LTTng-UST est capable d'instrumenter des programmes écrits en C/C++, en Java et même en Python. De plus, il est facile de combiner des traces produites dans le mode utilisateur par LTTng-UST et des traces au niveau du noyau produites par LTTng.

2.3.3 Traçage du noyau

Comme expliqué dans la section précédente, de nombreuses solutions ont été proposées pour tracer le système dans le mode utilisateur. Bien qu'utiles pour comprendre le fonctionnement de certains programmes, les outils précédents ne permettent pas de comprendre le fonctionnement global du système instrumenté, contrairement au traçage au niveau du noyau du système d'exploitation.

Ainsi, plusieurs techniques d'instrumentation ont été développées pour tracer le noyau, qu'elles soient dynamiques ou statiques, et différents outils puissants et efficaces ont été proposés.

KProbes

Les Kprobes sont des mécanismes présents dans le noyau Linux permettant d'instrumenter dynamiquement l'exécution du noyau pour obtenir des évènements lorsque les sondes sont exécutées. Elles peuvent être placées à presque toutes les adresses du noyau (Jim Keniston), mais reposent sur des interruptions pour collecter l'état du système au moment où elles sont exécutées, ce qui crée une perte de performance plus importante que des points d'instrumentation statiques.

Ftrace

Ftrace est un traceur très populaire qui est nativement inclus dans les distributions Linux. Il peut agréger l'ensemble des points de trace statiques intégrés au noyau, mais également utiliser les Kprobes pour instrumenter dynamiquement le système d'exploitation.

Cet outil peut présenter les traces récoltées sous forme d'une liste temporellement ordonnée d'évènements, mais également sous forme de graphe, de façon à permettre de mieux identifier certaines erreurs (Edge, 2009).

Comme l'explique Bird, Ftrace est également capable d'agréger des points de traces implicites, qui sont placés automatiquement dans le code par le compilateur, selon les options associées à ce dernier ou l'utilisation de macro spécifique dans le code source (Bird, 2009). Afin de collecter les évènements de manière performante, un tampon circulaire est placé sur chaque coeur du processeur, et des opérations atomiques assurent la synchronisation de l'accès à ces espaces mémoire. Ce traceur est donc relativement efficace, même avec des programmes parallélisés.

SystemTap

SystemTap est un traceur permettant d'écrire du code d'instrumentation en C, et de spécifier son emplacement dans le noyau (Eigler and Hat, 2006). C'est un outil très flexible puisqu'il permet à ses utilisateurs de choisir des points de trace statiques ou dynamiques, dans le mode utilisateur ou le mode noyau, en utilisant des filtres d'activation ou non.

Cependant, cet outil n'est pas très performant, et représente un risque de sécurité dans les systèmes qu'il instrumente. En effet, le code d'instrumentation est exécuté avec les privilèges du noyau, pouvant causer de l'exécution à distance de code malveillant (CVE, 2010). De plus, Gregg explique que l'utilisation de SystemTap peut être dangereuse dans des environnements de production, puisqu'il peut causer de graves erreurs sur les systèmes qu'il surveille (Sys,

2011).

Pour effectuer une instrumentation statique du mode utilisateur, l'outil se base sur les points de traces USDT, inspirés du travail réalisé sur DTrace (Gregg and Mauro, 2011).

eBPF

EBPF, pour *Extended Berkeley Packet Filter*, est un outil présent dans les systèmes Linux pouvant permettre de tracer ces plateformes. Il repose sur BPF (*Berkeley Packet Filter*), une machine virtuelle présente dans le noyau Linux dont le but premier est de filtrer les paquets réseau. L'extension de cette technologie, eBPF, permet de s'attacher à des points de trace statiques ou dynamiques dans différents endroits du noyau ou du mode utilisateur. À l'instar de SystemTap, l'instrumentation statique du mode utilisateur d'eBPF se fait à l'aide des points de trace USDT. Cet outil permet de convertir les traces obtenues en traces au format CTF, le format utilisé par LTTng et BareCTF, détaillé ci-après.

Cet outil semble prometteur pour le traçage, mais son utilisation et l'étendue de ses possibilités sont encore en train d'être explorées. De nombreux scripts permettent d'en faire une utilisation efficace et de récolter différentes données, mais l'utilisation d'eBPF peut être déroutante pour certains utilisateurs.

LTTng

LTTng (*Linux Trace Toolkit next generation*) est un traceur qui succède à LTT. Il a été proposé en code source libre par Desnoyer et est devenu l'un des traceurs les plus efficaces présents sur les distributions de GNU/Linux (Desnoyers and Dagenais, 2006). LTTng est capable de tracer le noyau et le mode utilisateur. L'auteur explique que la performance du traceur est l'un des principaux critères de développement, et que sa solution crée peu de surcoût à l'utilisation, avec un estampillage de temps très précis et de nombreuses informations enregistrables.

L'un des avantages de LTTng est qu'il est capable d'envoyer des traces sur le réseau ou sur la machine qui est directement tracée, et dans deux modes différents : le mode *live tracing* et le mode *snapshot*. Avec la première méthode, les évènements sont envoyés à l'hôte au fur et à mesure qu'ils sont enregistrés par le traceur, alors que le second mode permet d'envoyer l'état intégral des tampons de traçage au moment spécifié par l'utilisateur. Cela offre une grande flexibilité dans l'usage de LTTng. De plus, il est possible d'appliquer des filtres sur les points de traces activés pour les enregistrer sous certaines conditions, et des modules peuvent être développés afin d'améliorer la personnalisation dans l'utilisation du traceur.

Cet outil est capable d'utiliser les points de trace statiques présents dans le noyau ou d'insérer des points de trace dynamiques à l'aide des Kprobes détaillées ci-dessus.

Pour s'assurer de la performance du traceur, plusieurs techniques ont été mises en oeuvre. La première est le format de trace produit, le format CTF (*Commont Trace Format*), qui est un format binaire optimisé ayant pour but de devenir un standard dans le monde du traçage. Ce format permet également de diminuer le nombre de paquets réseau à envoyer par rapport aux autres formats proposés par les outils précédents. De plus, un fichier de métadonnées est envoyé avec les traces binaires, permettant d'apporter plus d'information sur le contexte des traces enregistrées, donnant ainsi des informations précises. De plus, LTTng associe plusieurs sous-tampons circulaires à chaque coeur d'exécution, permettant à l'outil de tracer différents fils d'exécution et de retransmettre précisément leur comportement, sans pénaliser l'exécution du programme parallélisé. La synchronisation de ces tampons se fait à l'aide d'opérations atomiques locales, rendant leur utilisation extrêmement efficace. Enfin, il a été décidé par les développeurs de jeter le surplus d'évènements arrivant au traceur avant qu'il n'ait le temps d'enregistrer les tampons d'évènement pleins, afin d'impacter au minimum la performance de la plateforme tracée et parce qu'il n'est simplement pas possible de bloquer le noyau du système d'exploitation sur un tampon plein sans risquer un interblocage. Ainsi, LTTng est l'un des traceurs les plus performants pour les systèmes GNU/Linux, et n'est pas dépendant d'une architecture spécifique. Il peut ainsi être déployé sur des systèmes ARM ou MIPS, ainsi que sur de nombreuses autres architectures (Desnoyers and Dagenais, 2009).

BareCTF

BareCTF est un outil en python proposé par Efficios, la même entreprise créatrice et mainteneuse de LTTng. Cet outil permet de tracer des systèmes bare-metal, c'est-à-dire des plateformes n'utilisant pas de système d'exploitation. Ce programme produit du code C capable de générer des points de traces au format CTF, le même format utilisé par LTTng, à l'aide d'un simple fichier de configuration yaml.

Bertauld a évalué la performance et l'utilité de BareCTF en traçant divers systèmes embarqués qui n'étaient pas instrumentables facilement sans cet outil (Bertauld and Dagenais, 2017).

L'auteur a par ailleurs montré comment il était possible de corréler différentes traces ainsi produites afin d'analyser plusieurs systèmes, en ayant la possibilité d'utiliser à la fois des traces générées par LTTng et par BareCTF. Cette synchronisation se fait en générant des évènements spécifiques sur un système maître utilisant LTTng et un système esclave roulant BareCTF.

Enfin, l'auteur a montré à l'ade de deux systèmes embarqués que le surcoût associé au traçage avec BareCTF était d'autant plus important que les capacités de calcul du système embarqué étaient faibles, mais que ce surcoût est acceptable pour la plupart des circuits utilisés dans l'industrie.

2.3.4 Visualisation de traces

Les traces générées au format CTF sont binaires, et ne sont donc pas lisibles directement par les humains. Ainsi, plusieurs outils de visualisation et d'analyse de trace ont été proposés, certains permettant de mettre rapidement en lumière le comportement du système surveillé, et d'autres offrant à l'utilisateur de nombreux outils de visualisation. Nous détaillerons dans la suite de cette section les outils permettant la visualisation de traces au format CTF, le format que nous avons utilisé lors de nos travaux.

Babeltrace

Le traceur LTTng a instauré un standard de stockage de trace binaire optimisé, appelé CTF, qui n'est pas lisible directement par les humains. Pour pallier à ce problème, Efficios, l'entreprise en charge du développement de LTTng, a mis en ligne le code source de Babeltrace, un outil capable d'afficher des traces CTF sous forme d'évènements textuels ordonnés. Les évènements affichés par Babeltrace sont estampillés précisément et comportent le nom de l'évènement ainsi que les arguments enregistrés au moment où le point de trace a été rencontré à l'exécution du système. Son interface textuelle est présentée dans la figure 2.4.

Babeltrace peut également convertir des traces au format CTF vers un autre format de trace et inversement, ce qui en fait un outil très utilisé dans le monde du traçage.

Cet outil fournit également une interface de programmation, accessible en C ou en Python, qui permet la lecture et l'écriture de traces au format CTF. Cela permet de filtrer les traces enregistrées, précisément selon les souhaits du programmeur, de supprimer certains évènements et même d'en fusionner.

De plus, ces interfaces de programmation permettent de convertir des données binaires en structures de données qui peuvent ensuite être modifiées pour être utilisées par d'autres algorithmes. Par exemple, avec l'interface Python, les évènements peuvent être lus de la trace et convertis en objets dictionnaire, qui après différentes étapes de modification, peuvent servir à alimenter des algorithmes d'apprentissage machine.

Cependant, la nouvelle version de Babeltrace est encore en cours de développement, et la lecture des traces au format CTF, bien qu'efficace, s'avère relativement lente lorsque le nombre

```
[17:13:30.935590402] (+0.001586505) hassbian sched_switch: { cpu_id = 0 }, { prev_comm = "swapper/0", prev_tid = 0, prev_prio = 20, prev_state = 0, next_comm = "kworker/0:2", next_tid = 116, next_prio = 20 }
[17:13:30.935597433] (+0.000007031) hassbian sched_switch: { cpu_id = 0 }, { prev_comm = "kworker/0:2", prev_tid = 116, prev_prio = 20, prev_state = 1, next_comm = "mmcqd/0", next_tid = 93, next_prio = 20 }
[17:13:30.935649829] (+0.000052396) hassbian sched_switch: { cpu_id = 0 }, { prev_comm = "mmcqd/0", prev_tid = 93, prev_prio = 20, prev_state = 1, next_comm = "kworker/0:2", next_tid = 116, next_prio = 20 }
[17:13:30.935654933] (+0.000005104) hassbian sched_switch: { cpu_id = 0 }, { prev_comm = "kworker/0:2", prev_tid = 116, prev_prio = 20, prev_state = 1, next_comm = "hass", next_tid = 744, next_prio = 20 }
[17:13:30.936051129] (+0.000396196) hassbian sched_switch: { cpu_id = 0 }, { prev_comm = "hass", prev_tid = 744, prev_prio = 20, prev_state = 130, next_comm = "swapper/0", next_tid = 0, next_prio = 20 }
[17:13:30.942053660] (+0.006002531) hassbian sched_switch: { cpu_id = 0 }, { prev_comm = "swapper/0", prev_tid = 0, prev_prio = 20, prev_state = 0, next_comm = "kworker/0:2", next_tid = 116, next_prio = 20 }
[17:13:30.942060692] (+0.000007032) hassbian sched_switch: { cpu_id = 0 }, { prev_comm = "kworker/0:2", prev_tid = 116, prev_prio = 20, prev_state = 1, next_comm = "mmcqd/0", next_tid = 93, next_prio = 20 }
[17:13:30.942088712] (+0.000028020) hassbian sched_switch: { cpu_id = 0 }, { prev_comm = "mmcqd/0", prev_tid = 93, prev_prio = 20, prev_state = 1, next_comm = "hass", next_tid = 744, next_prio = 20 }
[17:13:30.942605221] (+0.000516509) hassbian sched_switch: { cpu_id = 0 }, { prev_comm = "hass", prev_tid = 744, prev_prio = 20, prev_state = 130, next_comm = "mmcqd/0", next_tid = 93, next_prio = 20 }
[17:13:30.942660127] (+0.000063906) hassbian sched_switch: { cpu_id = 0 }, { prev_comm = "mmcqd/0", prev_tid = 93, prev_prio = 20, prev_state = 1, next_comm = "kworker/0:2", next_tid = 116, next_prio = 20 }
[17:13:30.942686210] (+0.000017083) hassbian sched_switch: { cpu_id = 0 }, { prev_comm = "kworker/0:2", prev_tid = 116, prev_prio = 20, prev_state = 1, next_comm = "swapper/0", next_tid = 0, next_prio = 20 }
[17:13:30.943996362] (+0.001310152) hassbian sched_switch: { cpu_id = 1 }, { prev_comm = "swapper/1", prev_tid = 0, prev_prio = 20, prev_state = 0, next_comm = "rcu_sched", next_tid = 7, next_prio = 20 }
[17:13:30.944009174] (+0.000012812) hassbian sched_switch: { cpu_id = 1 }, { prev_comm = "rcu_sched", prev_tid = 7, prev_prio = 20, prev_state = 1, next_comm = "swapper/1", next_tid = 0, next_prio = 20 }
[17:13:30.953982108] (+0.009972934) hassbian sched_switch: { cpu_id = 3 }, { prev_comm = "swapper/3", prev_tid = 0, prev_prio = 20, prev_state = 0, next_comm = "lttng-sessiond", next_tid = 998, next_prio = 20 }
[17:13:30.953988150] (+0.000006042) hassbian sched_switch: { cpu_id = 0 }, { prev_comm = "swapper/0", prev_tid = 0, prev_prio = 20, prev_state = 0, next_comm = "kworker/0:2", next_tid = 116, next_prio = 20 }
[17:13:30.954012420] (+0.000024270) hassbian sched_switch: { cpu_id = 0 }, { prev_comm = "kworker/0:2", prev_tid = 116, prev_prio = 20, prev_state = 1, next_comm = "swapper/0", next_tid = 0, next_prio = 20 }
[17:13:30.954041014] (+0.000028594) hassbian syscall_entry_close: { cpu_id = 3 }, { fd = 33 }
[17:13:30.954044972] (+0.000003958) hassbian sched_switch: { cpu_id = 0 }, { prev_comm = "swapper/0", prev_tid = 0, prev_prio = 20, prev_state = 0, next_comm = "lttng", next_tid = 1266, next_prio = 20 }
[17:13:30.954047629] (+0.000002657) hassbian syscall_exit_close: { cpu_id = 3 }, { ret = 0 }
[17:13:30.954118253] (+0.000070624) hassbian sched_switch: { cpu_id = 3 }, { prev_comm = "lttng-sessiond", prev_tid = 998, prev_prio = 20, prev_state = 1, next_comm = "swapper/3", next_tid = 0, next_prio = 20 }
[17:13:30.954127524] (+0.000009271) hassbian syscall_entry_shutdown: { cpu_id = 0 }, { fd = 3, how = 2 }
[17:13:30.954132264] (+0.000004740) hassbian syscall_exit_shutdown: { cpu_id = 0 }, { ret = 0 }
[17:13:30.954142264] (+0.000010000) hassbian syscall_entry_close: { cpu_id = 0 }, { fd = 3 }
[17:13:30.954144451] (+0.000002187) hassbian syscall_exit_close: { cpu_id = 0 }, { ret = 0 }
[17:13:30.954701897] (+0.000557446) hassbian syscall_entry_write: { cpu_id = 0 }, { fd = 1, buf = 1012272, count = 41 }
[17:13:30.954773095] (+0.000071198) hassbian syscall_exit_write: { cpu_id = 0 }, { ret = 41 }
[17:13:30.954779084] (+0.000005989) hassbian syscall_entry_exit_group: { cpu_id = 0 }, { error_code = 0 }
[17:13:30.955474082] (+0.000694998) hassbian sched_process_exit: { cpu_id = 0 }, { comm = "lttng", tid = 1266, prio = 20 }
[17:13:30.955563144] (+0.000089062) hassbian sched_switch: { cpu_id = 0 }, { prev_comm = "lttng", prev_tid = 1266, prev_prio = 20, prev_state = 64, next_comm = "sshd", next_tid = 844, next_prio = 20 }
[17:13:30.955563457] (+0.000000313) hassbian sched_switch: { cpu_id = 2 }, { prev_comm = "swapper/2", prev_tid = 0, prev_prio = 20, prev_state = 0, next_comm = "sudo", next_tid = 1262, next_prio = 20 }
[17:13:30.955596425] (+0.000032968) hassbian syscall_entry_write: { cpu_id = 2 }, { fd = 4, buf = 2129470659, count = 1 }
[17:13:30.955605748] (+0.000009323) hassbian syscall_exit_write: { cpu_id = 2 }, { ret = 1 }
[17:13:30.955630384] (+0.000024636) hassbian syscall_entry_read: { cpu_id = 2 }, { fd = 3, count = 1 }
[17:13:30.955633769] (+0.000003385) hassbian syscall_entry_read: { cpu_id = 0 }, { fd = 11, count = 16384 }
[17:13:30.955639498] (+0.000005729) hassbian syscall_exit_read: { cpu_id = 2 }, { ret = 1, buf = 2129471303 }
[17:13:30.955639498] (+0.000000000) hassbian syscall_exit_read: { cpu_id = 0 }, { ret = 41, buf = 2128130684 }
[17:13:30.955674081] (+0.000034583) hassbian syscall_entry_getpid: { cpu_id = 0 }, { }
[17:13:30.955676894] (+0.000002813) hassbian syscall_exit_getpid: { cpu_id = 0 }, { ret = 844 }
[17:13:30.955750279] (+0.000073385) hassbian syscall_entry_write: { cpu_id = 0 }, { fd = 3, buf = 14670184, count = 96 }
[17:13:30.955781425] (+0.000031146) hassbian syscall_entry_getuid: { cpu_id = 2 }, { }
[17:13:30.955784289] (+0.000002864) hassbian syscall_exit_getuid: { cpu_id = 2 }, { ret = 0 }
[17:13:30.955799862] (+0.000015573) hassbian net_dev_queue: { cpu_id = 0 }, { skbaddr = 0xA688DBB8, len = 162, name = "enxb827eb8ec29a", network_header_type = ( "_ipv4" : container = 1 ), network_header = { _ipv
4 = { version = 4, ihl = 5, tos = 8, tot_len = 148, id = 0x2B37, frag_off = 16384, ttl = 64, protocol = ( "_tcp" : container = 6 ), checksum = 0x7550, saddr_padding = { }, saddr = [ [0] = 132, [1] = 207, [2] = 7
2, [3] = 35 ], daddr_padding = { }, daddr = [ [0] = 132, [1] = 207, [2] = 72, [3] = 19 ], transport_header_type = ( "_tcp" : container = 1 ), transport_header = { _tcp = { source_port = 22, dest_port = 41640, se
q = 1300423992, ack_seq = 201615319, data_offset = 8, reserved = 0, flags = 0x18, window_size = 237, checksum = 0x9A5B, urg_ptr = 0 } } } }
[17:13:30.955801581] (+0.000001719) hassbian syscall_entry_open: { cpu_id = 2 }, { filename = "/etc/login.defs", flags = 131072, mode = 438 }
[17:13:30.955838820] (+0.000037239) hassbian syscall_exit_open: { cpu_id = 2 }, { ret = 6 }
[17:13:30.955849602] (+0.000010782) hassbian syscall_exit_write: { cpu_id = 0 }, { ret = 96 }
[17:13:30.955865279] (+0.000015677) hassbian syscall_entry_read: { cpu_id = 2 }, { fd = 6, count = 4096 }
[17:13:30.955879081] (+0.000013802) hassbian syscall_exit_read: { cpu_id = 2 }, { ret = 4096, buf = 19608960 }
[17:13:30.955900956] (+0.000021875) hassbian sched_switch: { cpu_id = 0 }, { prev_comm = "sshd", prev_tid = 844, prev_prio = 20, prev_state = 1, next_comm = "kworker/0:2", next_tid = 116, next_prio = 20 }
[17:13:30.955915018] (+0.000014062) hassbian sched_switch: { cpu_id = 0 }, { prev_comm = "kworker/0:2", prev_tid = 116, prev_prio = 20, prev_state = 1, next_comm = "mmcqd/0", next_tid = 93, next_prio = 20 }
[17:13:30.955944393] (+0.000029375) hassbian syscall_entry_read: { cpu_id = 2 }, { fd = 6, count = 4096 }
[17:13:30.955951320] (+0.000006927) hassbian syscall_exit_read: { cpu_id = 2 }, { ret = 4096, buf = 19608960 }
[17:13:30.955969184] (+0.000017864) hassbian sched_switch: { cpu_id = 0 }, { prev_comm = "mmcqd/0", prev_tid = 93, prev_prio = 20, prev_state = 1, next_comm = "hass", next_tid = 744, next_prio = 20 }
```

Figure 2.4 Visualisation de trace à l'aide de Babeltrace

d'évènements est très grand. De plus, la documentation de l'interface Python est encore en cours de rédaction, et aucune indication n'est pour l'instant fournie sur la manière de traiter des traces dans le mode *live tracing* de LTTng.

LTTng analyses

Le traceur LTTng offre certains outils pour analyser les traces recueillies. Ces outils s'appuient sur babeltrace pour lire les traces CTF et, bien que ce ne soient pas exactement des outils de visualisation, ils permettent d'afficher des informations utiles à l'utilisateur sur le système tracé. Ainsi, Desfossez a proposé un script permettant d'afficher les latences lors d'un transfert entrée/sortie et un autre permettant d'afficher des statistiques précises sur les appels système enregistrés par le traceur (Desfossez, 2018). Les deux résultats de ces outils sont présentés respectivement à la figure 2.5 et à la figure 2.6.

```
robin@station14:~/Bureau/TestMachineLearning/HomeAssistant/hassbian/snapshot-1-20180419-111425-44$ lttng-iolatencyfreq .
Warning: intersect mode not available - disabling
         Use babeltrace 1.4.0 or later to enable
Checking the trace for lost events...
Processing the trace: 100% [##################################################################################################################################################] Time: 0:00:02
Timerange: [2018-04-19 17:14:22.884027826, 2018-04-19 17:14:25.394401602]
I/O request latency distribution open (µs)
##########################################################################
  6.250                                                                 564
409.306                                                                   0
812.361                                                                   0
1215.417                                                                  0
1618.472                                                                  0
2021.528                                                                  0
2424.584                                                                  0
2827.639                                                                  1
3230.695                                                                  2
3633.750                                                                  3
4036.806                                                                  0
4439.862                                                                  0
4842.917                                                                  0
5245.973                                                                  1
5649.028                                                                  2
6052.084                                                                  1
6455.140                                                                  0
6858.195                                                                  0
7261.251                                                                  0
7664.306                                                                  2

I/O request latency distribution read (µs)
##########################################################################
  1.197                                                                 625
 10.080                                                                  64
 18.962                                                                  14
 27.845                                                                   5
 36.728                                                                   1
 45.611                                                                   4
 54.493                                                                   5
 63.376                                                                   1
 72.259                                                                   5
 81.142                                                                   0
 90.025                                                                   0
 98.907                                                                   0
107.790                                                                   1
116.673                                                                   2
125.555                                                                   0
134.438                                                                   0
143.321                                                                   0
152.204                                                                   0
161.087                                                                   0
169.969                                                                   1

I/O request latency distribution write (µs)
##########################################################################
 10.989                                                                   1
 16.286                                                                   1
```

Figure 2.5 Analyse LTTng sur la latence des entrées/sorties

Puisque ces codes sources sont ouverts, il est facile de développer de nouvelles analyses. Cependant, la visualisation des informations est faite dans un terminal, ce qui peut empêcher l'adoption d'un tel outil par une large communauté d'utilisateurs. De plus, même si les analyses sont puissantes, elles requièrent une bonne connaissance du traçage avec LTTng et du format CTF pour être développées.

```
robin@station14:~/Bureau/TestMachineLearning/HomeAssistant/hassbian/snapshot-1-20180419-111425-44$ lttng-syscallstats .
Warning: intersect mode not available - disabling
         Use babeltrace 1.4.0 or later to enable
Checking the trace for lost events...
Processing the trace: 100% [##################################################################################################################################################################] Time: 0:00:02
Timerange: [2018-04-19 17:14:22.884027826, 2018-04-19 17:14:25.394401602]
Per-TID syscalls statistics (usec)
startTracing.sh (2146, TID: 2146)              Count          Min      Average          Max       Stdev  Return values
 - read                                          137        2.604        7.419       61.145       7.282  {'success': 137, 'EAGAIN': 2}
 - open                                          106       10.104       18.679       75.155       9.148  {'success': 68, 'ENOENT': 39, 'ENXIO': 2}
 - close                                          77        2.136        3.132        8.229       0.899  {'success': 78}
 - access                                         21        8.698       14.419       40.572       8.387  {'success': 2, 'ENOENT': 21}
 - setresuid                                      17        4.011        6.651       13.594       3.233  {'success': 18}
 - socket                                         12       10.886     1685.355     8067.362    3054.789  {'success': 10, 'EPROTONOSUPPORT': 4}
 - setresgid                                       9        3.75         5.318        8.593       1.788  {'success': 10}
 - connect                                         8       25.573       39.264       66.145      15.419  {'success': 3, 'ENOENT': 7}
 - setitimer                                       3        7.396        9.826       11.666       2.195  {'success': 4}
 - getuid                                          3        2.656        3.003        3.281       0.318  {'success': 4}
 - getpid                                          3        3.593        4.513        5.312       0.866  {'success': 4}
 - setgroups                                       3        5.729        9.583       11.771       3.348  {'success': 4}
 - getdents64                                      2        3.542       25.338       47.135           ?  {'success': 3}
 - geteuid                                         2        1.562        2.735        3.907           ?  {'success': 3}
 - getegid                                         1        2.344        2.344        2.344           ?  {'success': 2}
 - getgid                                          1        2.448        2.448        2.448           ?  {'success': 2}
 - getcwd                                          1       10.208       10.208       10.208           ?  {'success': 2}
 - set_tid_address                                 1         5.99         5.99         5.99           ?  {'success': 2}
 - setgid                                          1        6.406        6.406        6.406           ?  {'success': 2}
 - execve                                          1      916.084      916.084      916.084           ?  {'success': 2}
 - pipe                                            1       32.292       32.292       32.292           ?  {'success': 2}
 - getppid                                         1        3.698        3.698        3.698           ?  {'success': 2}
 - setuid                                          1        9.011        9.011        9.011           ?  {'success': 2}
 - set_robust_list                                 1        3.437        3.437        3.437           ?  {'success': 2}
 - getpgid                                         1        3.958        3.958        3.958           ?  {'success': 2}
 - clone                                           1      747.075      747.075      747.075           ?  {'success': 2}
 - getpgrp                                         1        4.583        4.583        4.583           ?  {'success': 2}
Total:                                           416
-----------------------------------------------------------------------------------------------------------------------
startTracing.sh (2121, TID: 2121)              Count          Min      Average          Max       Stdev  Return values
 - read                                          137        2.448        7.717       57.968       7.46  {'success': 137, 'EAGAIN': 2}
 - open                                          106        9.844       18.694       62.344       8.774  {'success': 68, 'ENOENT': 39, 'ENXIO': 2}
 - close                                          77        2.188        3.212        8.386       1.119  {'success': 78}
 - access                                         21        8.698       14.251       48.333       9.134  {'success': 2, 'ENOENT': 21}
 - setresuid                                      17        3.906         6.36       13.802       3.184  {'success': 18}
 - socket                                         12       10.781     1642.204     7675.069    2959.244  {'success': 10, 'EPROTONOSUPPORT': 4}
 - setresgid                                       9        3.802       18.449      128.748      41.379  {'success': 10}
 - connect                                         8       25.104        41.66       62.552       13.55  {'success': 3, 'ENOENT': 7}
 - setitimer                                       3        6.354        7.916        8.854       1.362  {'success': 4}
 - getuid                                          3        1.875        3.212        5.469       1.966  {'success': 4}
 - getpid                                          3         3.23         3.82        4.427       0.599  {'success': 4}
 - setgroups                                       3        5.104       15.937       33.437      15.298  {'success': 4}
 - getdents64                                      2         4.01       25.937       47.864           ?  {'success': 3}
 - geteuid                                         2        1.562        3.541         5.52           ?  {'success': 3}
 - getegid                                         1        2.396        2.396        2.396           ?  {'success': 2}
 - getgid                                          1        3.229        3.229        3.229           ?  {'success': 2}
 - getcwd                                          1        8.907        8.907        8.907           ?  {'success': 2}
 - set_tid_address                                 1        4.531        4.531        4.531           ?  {'success': 2}
```

Figure 2.6 Analyse LTTng sur les statistiques d'utilisation des appels systèmes

Trace Compass

Trace Compass est quant à lui un puissant outil de visualisation et d'analyse de trace. Il est capable de travailler rapidement sur de grosses traces et se base sur des machines à états finis pour présenter différents résultats. Il a été développé en Java sous Eclipse, et supporte différents types de formats de trace, dont le format CTF.

L'avantage de Trace Compass est de proposer des vues claires et colorées à partir de traces enregistrées, et la conversion des traces au format CTF vers un format lisible par les humains se fait entièrement dans cet outil à l'aide de machines à états.

Les machines à états utilisés sont très efficaces pour sauvegarder des grands nombres d'évènements enregistrés (Montplaisir-Gonçalves et al., 2013), et la construction de l'arbre des états est particulièrement optimisée. Ainsi, lorsqu'une trace est chargée dans le logiciel, l'arbre d'état est calculé selon l'intervalle visible dans la vue courante, ce qui permet à cet outil d'afficher rapidement des traces, et aux utilisateurs de pouvoir explorer efficacement les différents évènements enregistrés.

En outre, diverses vues ont été développées pour analyser le système ayant été tracé, comme la *Control Flow View* qui permet de suivre les opérations effectuées par chacun des fils d'exécution des différents processus tracés, d'afficher des statistiques sur l'utilisation des

ressources du système ou encore l'affichage de statistiques relatives au système durant la période où il a été tracé. Les figures 2.7, 2.8 et 2.9 montrent les différentes vues énumérées. De plus, il est très facile de développer de nouvelles analyses et de nouvelles vues à l'aide de fichier XML, et des travaux portant sur les machines virtuelles (Nemati et al., 2017) ou des vues spécifiques à des conteneurs ont été implémentés. Pour obtenir des vues encore plus riches, il est possible d'ajouter du code Java au projet. De plus, il est possible d'attacher des scripts écrits en Python afin de traiter les traces visualisées sans avoir à modifier le code source de l'outil.

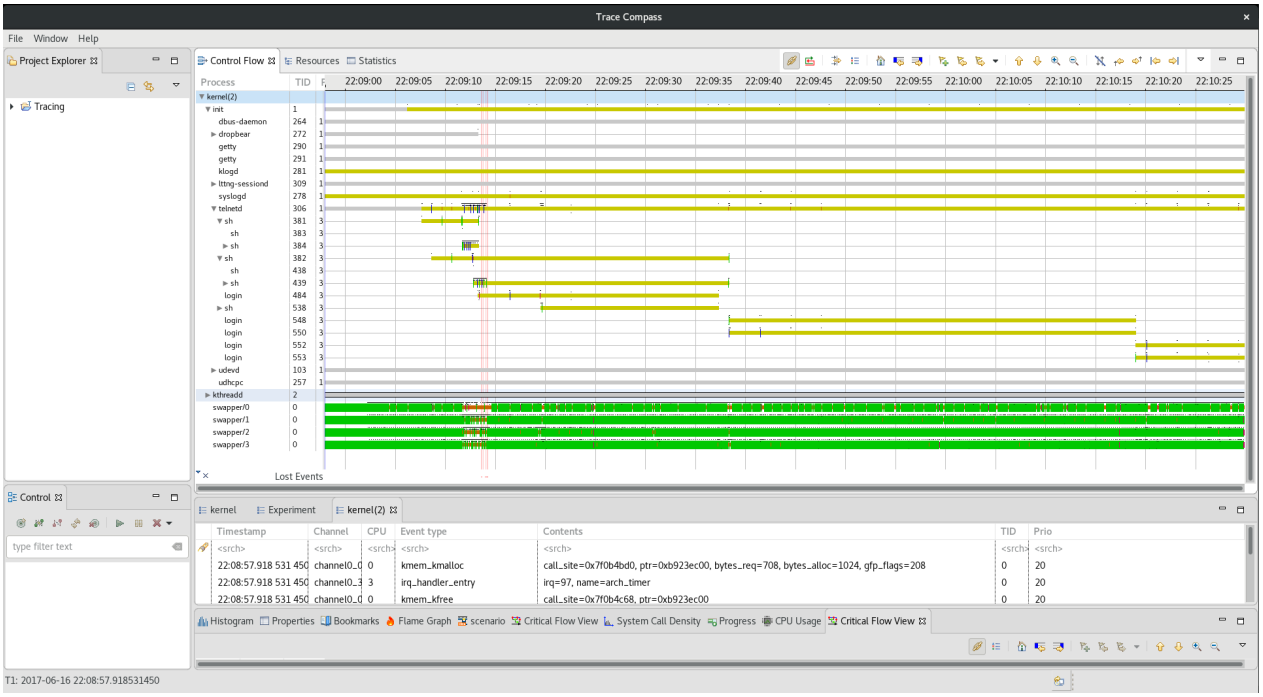

Figure 2.7 Control Flow View de Trace Compass

Trace Compass est ainsi devenu un outil incontournable pour visualiser les traces enregistrées et pour mener des analyses rapides et puissantes. Cependant, le logiciel peut être compliqué à prendre en main pour des développeurs, puisqu'il repose sur Eclipse, mais des travaux visant à détacher le projet de cette plateforme sont en cours.

2.4 Conclusion de la revue de la littérature

La rapide expansion du nombre d'objets connectés sur Internet crée de nombreux défis, dont l'un des principaux est la sécurité de ses usagers. Ainsi, de plus en plus d'attaques informatiques impliquent de tels objets, souvent très vulnérables comparés aux systèmes informa-

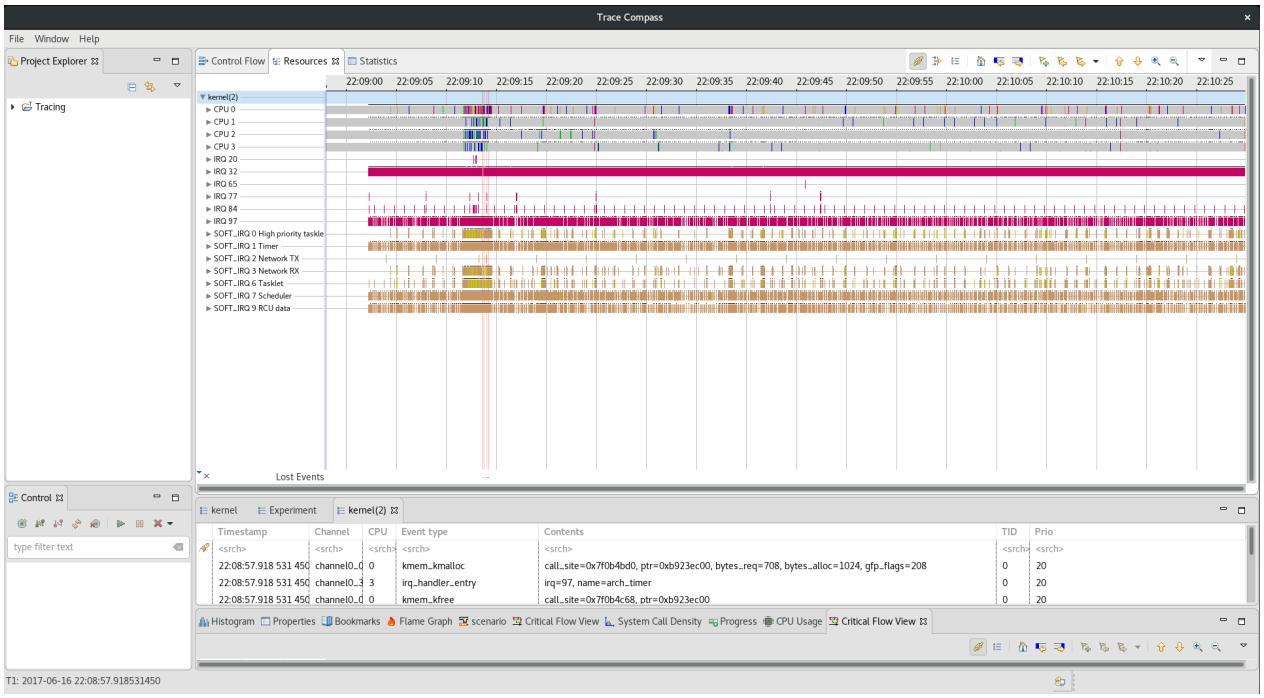

Figure 2.8 Vue des ressources du système dans Trace Compass

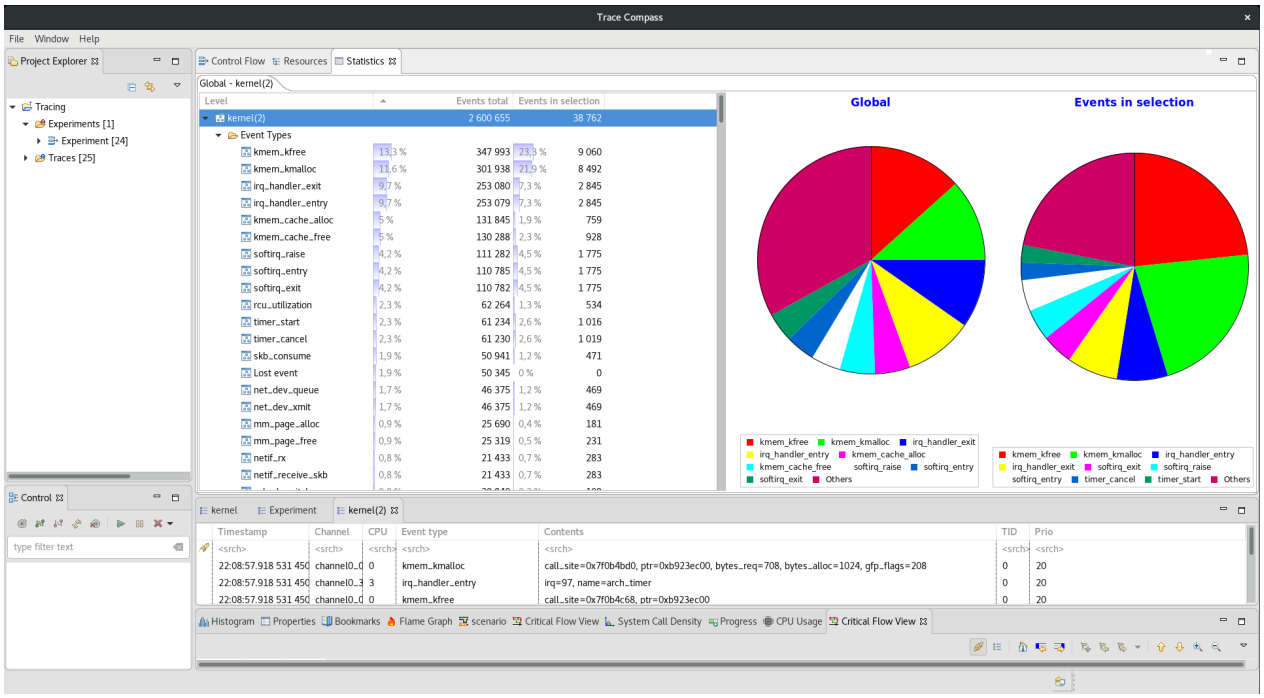

Figure 2.9 Vue de statistiques propres au système tracé dans Trace Compass

tiques traditionnels, et l'ampleur de ces attaques peut être considérable. Cette insécurité des objets connectés crée d'importants risques pour les utilisateurs et les entreprises, il devient donc urgent de proposer des solutions efficaces pour détecter des intrusions sur de tels objets.

Ce mémoire vise à détailler une solution performante pour détecter des intrusions, compatible avec la majorité des objets intelligents car peu intrusive, et utilisant des techniques d'apprentissage machine pour automatiser la détection. Les outils développés sont capables de traiter des informations à différents niveaux, qu'elles proviennent du noyau ou de l'espace utilisateur, et s'appuient sur des techniques de traçage performantes pour parvenir à ce résultat.

CHAPITRE 3 MÉTHODOLOGIE

Dans ce chapitre, nous présentons l'environnement de travail utilisé dans le cadre de nos travaux de recherche. Nous détaillons également la méthodologie employée pour valider l'efficacité de notre solution. Cette section vise à permettre de faciliter la reproduction de la solution proposée ainsi que de ses résultats. Ainsi, nous détaillons les caractéristiques matérielles des différentes plateformes utilisées ainsi que des différents projets open source nécessaires à la mise en place de notre solution.

Le fonctionnement des outils que nous proposons est détaillé dans le chapitre suivant.

3.1 Configuration matérielle

3.1.1 Objets connectés

Pour valider l'efficacité de notre solution, par rapport à sa capacité de détecter les intrusions mais également son impact sur la performance des systèmes surveillés, nous avons déployé une infrastructure classique d'objets connectés dans une maison. Elle est composée de capteurs et d'un actionneur, qui communiquent avec le protocole Z-Wave. Pour effectuer le lien entre ce protocole et l'Internet, il est nécessaire d'utiliser un autre objet, appelé contrôleur domotique ou contrôleur dans la suite de ce mémoire. Ce contrôleur devra donc disposer d'une connexion Z-Wave et d'une connexion au réseau IP classique, avec une couche logicielle capable de gérer les messages des objets connectés et de leur faire effectuer des actions.

Notre installation est capable d'allumer une prise électrique intelligente en fonction de différentes informations relevées par les capteurs présents. Elle est également en mesure d'envoyer des notifications sur des smartphones et des fureteurs Internet lorsque certaines valeurs seuils sont détectées par les capteurs.

Capteurs

Les capteurs utilisés dans nos expériences sont regroupés dans un unique boitier, le MultiSensor6 du fabricant Aeotec. Cet objet regroupe les capacités de six capteurs différents : un capteur de mouvement, un capteur de température, un capteur de lumière, un capteur d'humidité et un capteur de vibration.

Il peut être alimenté sur batterie ou via un port USB. Les caractéristiques précises des

capteurs peuvent être trouvées sur le site Internet du fabricant [1].

Actionneur

L'actionneur que nous avons sélectionné est une prise électrique connectée, capable de surveiller la consommation électrique qui passe au travers. De plus, elle peut être allumée ou éteinte avec des commandes envoyées sur le réseau Z-Wave. Il s'agit de la SmartSwitch6, du même fabricant.

La prise a également l'avantage de protéger les objets qui sont branchés dessus des courts-circuits et peut également alimenter des dispositifs USB.

Contrôleur

En guise de contrôleur, nous avons choisi un Raspberry Pi 3, un système embarqué capable d'exécuter divers systèmes d'exploitation et d'être programmé avec Python. Les caractéristiques détaillées du Raspberry Pi 3 peuvent être trouvées dans le tableau 3.1.

Tableau 3.1 Caractéristiques matérielles du RPi3

Processeur	AMRv7 BCM2837 4 coeurs, 64bits
Fréquence de l'horloge	1.2 GHz
Port Ethernet	10/100
Mémoire Vive	1GB
Wi-Fi	802.11 b/g/n (BCM43438)
Bluetooth Low Energy	BCM43438
Alimentation	2.5 A

Afin que cet objet puisse communiquer sur le réseau Z-Wave, il est nécessaire d'ajouter un dispositif USB compatible Z-Wave au Rapsberry Pi. Nous avons opté pour la Z-Stick 5 du constructeur Aeotec [2]. Ce dispositif est entièrement compatible avec le standard Z-Wave et inclus les considérations de chiffrement du protocole.

Enfin, il est nécessaire d'ajouter un logiciel capable de gérer les communications Z-Wave et les objets à contrôler. Dans cette optique, nous avons utilisé le projet à source ouverte Home Assistant, et plus particulièrement la distribution Hassbian [3]. Ce projet est devenu très populaire chez les utilisateurs de réseaux domotiques et est compatible avec de très

1. https ://aeotec.com/z-wave-sensor
2. https ://aeotec.com/z-wave-usb-stick
3. https ://www.home-assistant.io/docs/installation/hassbian/installation/

nombreux objets. La documentation est très complète et la communauté de développeurs très active.

Le contrôleur étant le seul objet connecté sur lequel nous pouvons aisément installer le traceur, qui est utilisé pour collecter les informations nécessaires à la détection d'intrusion, il sera le seul objet sur lequel nous travaillerons dans nos expérimentations. Nous ne tracerons pas les autres objets connectés pour obtenir nos résultats, mais la méthodologie de détection pourrait s'appliquer de la même manière sur ces objets en y installant le traceur.

3.1.2 Machine d'analyse

Dans le cadre de nos expériences, nous avons utilisé un ordinateur local placé physiquement proche de notre réseau domotique. Cependant, notre solution est compatible avec une analyse dans l'infonuagique ou sur un objet spécialement dédié à cette tâche. Nous détaillerons par la suite l'environnement utilisé lors de nos travaux expérimentaux.

La machine d'analyse utilisée est un ordinateur fixe comportant un processeur Intel i7-3770 cadencé à 3.40GHz avec 8 coeurs. La mémoire vive de cette machine est de 16GB, et la connexion avec le contrôleur domotique se fait par un VLAN filaire, le contrôleur Ethernet est un Intel 82579V Gigabit. La carte graphique est une AMD Readon 9 Fury. Le disque dur est un disque dur mécanique de 500 GB. La distribution utilisée sur cette machine est une Debian (stretch) 64 bits.

3.2 Environnement logiciel

Nos outils se basent totalement sur des projets à source ouverts pour fonctionner. Ces différents outils sont sujets à évoluer, nous listerons donc ici les versions des logiciels utilisés.

3.2.1 Traçage

Afin de tracer les objets connectés, nous avons eu recours à des techniques de traçage. Comme expliqué dans la revue critique de la littérature, le traceur le plus approprié pour minimiser le surcout associe au traçage sur les objets connectés est LTTng [4]. Nous avons utilisé les versions 2.9 des différentes librairies de LTTng (lttng-tools, lttng-modules et lttng-ust), aussi bien sur les objets à tracer que sur le serveur d'analyse.

La dernière version en date, LTTng 2.10, introduit des nouveautés intéressantes mais son intégration dans nos outils aurait nécessité de reprendre depuis le début les expériences alors

4. https ://lttng.org/

réalisées. Nous avons donc pris le parti de ne pas modifier la version du traceur.

De plus, le traçage nous a permis d'obtenir des informations à différents niveaux de l'objet connecté tracé, tels que les appels systèmes effectués ainsi que les arguments reçus par le noyau du système d'exploitation, des informations sur les paquets réseaux émis ou encore des informations relatives à l'ordonnanceur du processeur.

3.2.2 Apprentissage machine

En ce qui concerne l'apprentissage machine, nous avons rendu notre solution compatible avec les librairies python les plus populaires. Ainsi, nous utilisons le langage Python dans sa version 3.5.3, puisque de nombreuses librairies d'apprentissage machine ont été développées pour se langage.

Afin d'implémenter les différents algorithmes, nous avons utilisé deux librairies à source ouvertes : scikit-learn et Keras. Scikit-learn a été utilisée dans sa version 0.19.1, la version la plus stable au début du développement. Cette librairie implémente de nombreuses techniques d'apprentissage, et nous l'avons utilisé pour l'ensemble de nos algorithmes à l'exception des réseaux LTSM. Keras a été utilisée dans sa version 2.1.5, et utilise le *backend* tensorflow pour effectuer les calculs nécessaires à l'apprentissage et à la prédiction du réseau LTSM utilisé. Tensorflow a ainsi été utilisé dans sa version 1.6.0.

3.3 Validation des résultats

Afin de valider notre solution, nous avons décidé de l'implémenter dans l'environnement décrit ci-dessus. Ainsi, nous avons déployé les différents outils matériels et logiciels présentés, et avons développé les outils nécessaires au fonctionnement de notre solution.

Nous avons ensuite décidé d'implémenter diverses attaques détaillées dans la section suivante et avons créé un jeu de donnée comportant des séquences d'évènements sains et d'évènements liés à des intrusions afin d'entrainer des modèles. Ce jeu de données a été séparé en jeu d'entrainement et en jeu de test.

Une fois les modèles entrainés sur le jeu d'entrainement, nous avons vérifié leur performance sur le jeu de test en mesurant certaines métriques détaillées dans l'article suivant.

De plus, nous avons utilisé des *benchmark* en source libre pour mesurer l'impact du traçage sur la performance du système surveillé.

Enfin, nous avons utilisé un nouveau jeu de données composé d'évènements sains et d'une nouvelle attaque pour mesurer la rapidité d'analyse des modèles entrainés.

Les caractéristiques matérielles et logicielles de notre environnement de travail ont été détaillées dans cette section, ainsi que la méthodologie de validation que nous avons employée. La section suivante détaille la solution que nous proposons pour parvenir à une détection efficace d'intrusion sur les objets connectés d'une maison intelligente, et expose les différents résultats obtenus.

CHAPITRE 4 ARTICLE 1 : MULTI-LEVEL HOST-BASED INTRUSION DETECTION SYSTEM FOR INTERNET OF THINGS

Authors

Robin Gassais <robin.gassais@polymtl.ca>
José M. Fernandez <jose.fernandez@polymtl.ca>
Daniel Aloise <daniel.aloise@polymtl.ca>
Michel Dagenais <michel.dagenais@polymtl.ca>

Submitted to : Special Issue : Security and Privacy for the Internet of Things, Elsevier Journal of Computer Networks

Keywords : Host-based Intrusion Detection System, IoT, Anomaly Detection, Machine Learning, Tracing

Abstract

The Internet of things (IoT) is quickly expanding as more and more connected devices are designed, especially in homes where the appliances, the lighting system, and even the kids toys are now connected to the Internet. Vendors are competing to create and release quickly innovative connected objects, without focusing on the security issues. As a consequence, attacks involving smart devices, or targeting them, are proliferating, creating threats to users privacy and even their physical security. In addition, the heterogeneous technologies involved in IoT make the attempts to develop protection on smart devices much harder. Thus, most of the intrusion detection systems developed for those platforms are based on network activity. However, on many systems, intrusions cannot easily or reliably be detected from network traces. For instance, the network activity could be used to detect that a new software package is being downloaded on a device, but would fail to detect whether this software is harmful to the system when being executed. We propose a novel host-based automated framework for intrusion detection. Our work combines user space and kernel space information and machine learning techniques to detect various kinds of intrusions in smart devices. Our solution uses tracing techniques to automatically get devices behavior, process this data into numeric arrays to train several machine learning algorithms, and raise alerts whenever an intrusion is found. We implemented several machine learning algorithms, including deep learning ones, to get more than 98% precision in detection, while adding less than 5% CPU or memory

overhead on the device. We tested our solution on a realistic home automation system with actual threats.

4.1 INTRODUCTION

While vendors are innovating with more smart devices, and with the Internet access increasing worldwide, the Internet of Things (IoT), which is the interconnection of embedded devices with the Internet network, is quickly expanding in various areas, both within companies and in houses. Indeed, Cisco Evans (2011) predicts that there will be 50 billion smart devices connected to the Internet by 2020, or 6.58 things per inhabitant at that time. This refers to the connection of various embedded devices such as sensors, actuators, and vehicles able to interact with each other Atzori et al. (2010). While this growth induces the production of innovative objects, like connected speakers able to respond to a verbal request or order products, it creates a huge security threat for consumers and companies. Indeed, the vendors race for innovation does not leave much time to ensure the security of new smart devices. This could explain many recent cyber attacks involving connected objects, like the Mirai botnet targeting poorly secured device, using a default password, to launch one of the most powerful DDoS campaign ever seen in 2016 Kolias et al. (2017) against the Dyn DNS server. This cyber-attack succeeded in making unreachable for many hours some of the most world popular websites on the USA west coast. Most of the time, these devices have limited resources, and there is a huge heterogeneity in the connected device software, including the operating system, and the network protocols. For those reasons, the security community keeps raising alarms on the vulnerability of IoT devices, as did OWASP with its TOP 10 IoT vulnerability listOWASP.

To overcome the issues, some work has been done to detect intrusions on smart devices (Zarpelão et al., 2017). The article highlighted the need to get host-based intrusion detection systems (HIDS) since they can monitor more information about the connected device, and therefore can help to detect attacks that could not be identified with network information. Furthermore, Zarpelão and al. explain that traditional HIDS are not effective for IoT. All of the intrusion detection systems (IDS) presented in this article are network-based, while we could only find a few host-based intrusion detection systems for IoT.

Many very good HIDS have been developed for traditional systems such as OSSEC or Sagan, which provide multi-level monitoring of systems, with alerts correlation or active response. However, even if those solutions can be lightweight, for instance when using the OSSEC agent, the limited resources of smart devices cannot let those tools process the collected pieces of information directly on the device. Moreover, those tools may not be compatible

with smart objects hardware such as ARM CPUs. As a consequence, those solutions cannot detect intrusions, based on host information on smart devices. Our goals are therefore to determine if a lightweight and efficient HIDS for smart home devices can be developed and to study how machine learning algorithms could be used with tracing data to achieve good detection capabilities.

In this paper, we propose a complete framework for the dynamic automated analysis of IoT. We collect multi-level information on the monitored devices, with tracing techniques, and stream this data to an analysis engine, either located on a device inside the domotic system, on a dedicated machine inside the home, or in the cloud. The analysis engine can use several machine learning algorithms to detect anomalies on the device behavior. Then, when appropriate, alerts are raised by the analysis system with the name of the device and the kind of threat that has been detected.

We have several contributions through this article. First, we provide a complete tracing architecture where a user only has to specify what tracepoints have to be monitored and how often this information will be received. Our solution then activates tools that will send snapshots of the trace events at the user specified interval. Then, we provide an automatic tool for trace analysis, which can extract features from events and process them to become usable with open-sourced machine learning libraries. We also developed a convenient way to label each event. Finally, we optimized and compared several machine learning, and deep learning, algorithms to get the best detection capabilities.

The paper is structured as follow : Section II discusses the proposed works about intrusion detection, smart homes and tracing that have been proposed. Section III focuses on the solution proposed, with the framework architecture and a detailed review of each of its components. In section IV, we explain our experimental set-up and the proposed implementation. Section V highlights and discusses some results about the efficiency and the overhead introduced by our solution. Finally, section VII concludes on the whole framework developed.

4.2 Related Work

4.2.1 Smart Home and associated threats

We will refer to a smart home as a residence where smart devices such as sensors and actuators are connected to provide monitoring and/or new entertainment capabilities to its inhabitants. The typical architecture of such a residence is described in 4.1, where all the smart devices are part of the IoT Network, managed by a home controller that is the gateway with other Internet-based devices. This device is responsible to link the Z-Wave or the ZigBee network to

the traditional IP network. Some authors like Cheng distinguished the IoT networks according to the network protocol used by the devices (Cheng et al.) or according to the primary purpose of the device (e.g., healthcare, entertainment or surveillance). However, even in those infrastructure, the devices often rely on a home controller as a gateway to the traditional network.

Whether it is for personal comfort, with the possibility to manage the electricity consumption more efficiently, or for healthcare, Gubbi et al. (Gubbi et al., 2013) explains that homes will be the first area where connected devices will be deployed, before entering companies and more large-scale deployments. However, the article highlights that the security of such devices and networks is a crucial challenge, both at the device scale, with software and protocols, and at the cloud scale, with data protection for privacy and identity management. Hui (Hui et al., 2017) comes to the same conclusion regarding security and privacy. They are important challenges to get consumer adoption of smart homes. Moreover, the author identifies different threats, such as personal data theft like photos, videos, and documents, or the spying on personal habits that could be revealed to third parties. Moreover, smart devices are usually produced in large quantities, and vulnerabilities are often found after their release. One of the major issues is that those devices are often not updated, and if available the update process should be secured (Acosta Padilla et al., 2016b).

As smart devices are more widespread over homes, attackers are getting active in targeting such devices. Indeed, Saxena et al. (Saxena et al.) focuses on several attacks that strike smart devices, while Sikder et al. (Sikder et al., 2018) highlights the threats on sensors inside such a network. Furthermore, Mirai attack involved more than 200 000 infected devices (Antonakakis et al., 2017). While Babar et al. proposed a taxonomy for IoT attacks (Babar et al., 2011), we decided to summarize the main ideas from those article as follow :

- Data exfiltration : either eavesdropping on private life, information theft, or inference, the attacker goal is to obtain some confidential pieces of information that he could benefit from ;
- Data alteration : this may be achieved via the injection of false data or command to some device, via a replay or person-in-the-middle attack. The attacker goal here is to change the domotic system behavior to abuse it.
- Device unavailability : the hacker purpose is to make a system unreachable for the user. The target system can be a smart device that the intruder will infect, like the Brickerbot malware (Wagner et al., 2017) which bricks the device by wiping its files and corrupting its memory. However, the attacker target may be a remote system. This was the case for the infamous Mirai botnet, responsible for the largest DDoS

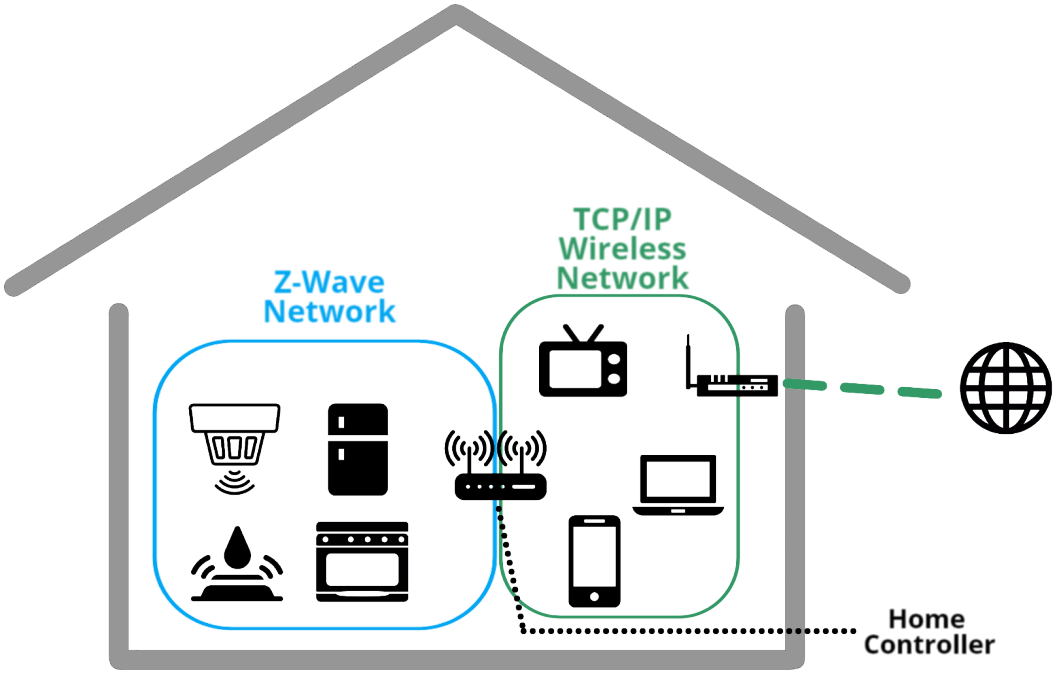

Figure 4.1 Smart home architecture

attack at the time, on remote servers such as OVH and DYN DNS (Kolias et al., 2017).

— Intrusion inside a network : connected devices, especially home controllers, may be used to enter a traditional network and then allow the attacker to bypass some security protections such as an external firewall.

— Physical security : smart devices could threaten its user physical security. For instance, if an intruder can control a connected oven and a smart gas valve, he might be able to make the oven explode and therefore hurt objects and persons around. Furthermore, with connected pacemakers having been found vulnerable, or connected cars being remotely controlled, the threats associated with cyber-physical systems are rising.

4.2.2 Intrusion detection in IoT context

Intrusion detection system

A distinction is made between network-based IDS (NIDS) and host-based IDS (HIDS), as explained by Sabahi et Movaghar (Sabahi and Movaghar, 2008), even if some work has been done using both approaches with a hybrid IDS. However, the author highlighted that,

regardless of the type of data collected, the detection can be obtained with two techniques : misuse detection or anomaly detection. The first type of detection relies on an expert system, which can be based on the known threats signature, with rules about the collected data or on state transitions in the system, while the second monitors the system behavior to detect anomalies and therefore potential intrusions. Both methods have benefits, with misuse detection being accurate and working well on known threats, according to Zarpelão et al. (Zarpelão et al., 2017), but being unable to detect new attacks, unlike anomaly detection. However, anomaly detection requires to characterize and identify the normal behavior of the system, which can be difficult to achieve.

Intrusion detection systems have been studied for a long time but, as highlighted Zarpelão et al. (Zarpelão et al., 2017), traditional IDS failed at protecting connected device for three main reasons : the devices have limited resources often insufficient to run a traditional agent, smart devices usually work under a mesh network where they can forward packets while being an endpoint, and there is a considerable heterogeneity in the technologies and network protocols used in IoT. Furthermore, little work has been done with HIDS, as Zarpelão et al. only studied NIDS. However, Nobakht et al. developed a whole framework to deploy a host-based intrusion detection system on software-defined networks (SDN)(Nobakht et al., 2016), but the researcher still relies on network information to detect intrusions. Moreover, SDNs are not yet deployed in homes, which makes this solution inconvenient for now.

While tracing has been used for a long time to detect intrusions, especially system anomalies, either in traditional or embedded systems (Burguera and Nadjm-Tehrani, 2011), Eskandari et al. (Eskandari et al., 2013) explained that intrusions can be detected via tracing from incoherent events (for instance, the execution of /bin/bash on systems that are not supposed to) or from sequences of events that are unusual.

Anomaly detection

Chandola et al. defined anomaly detection as a set of techniques that aim at finding data that does not match the expected behavior (Chandola et al., 2009). According to the author, this problem can occur in many fields, including fraud detection, insurance, fault detection and intrusion detection. Furthermore, the article highlights several techniques that have been used for years to find anomalies, including classification, clustering, and statistics.

Providing several examples of tracing to detect anomalies in software, Murtaza et al. (Murtaza et al., 2012a) highlighted the need to use kernel traces with classification techniques to improve the accuracy of anomaly detection. Other work has been proposed to combine tracing events and anomaly detection, such as did Murtaza et al. (Murtaza et al., 2014b) in a paper

where he and his coauthors compared three anomaly detection techniques that only rely on syscalls to detect intrusions. They used three different machine learning algorithms, but without studying the overhead introduced by their solution or trying more recent deep neural network techniques.

It is very important to distinguish classification, clustering, and statistics when it comes to detecting anomalies on a system.

— Classification aims at associating a class to each input element fed to the algorithm. For instance, when focusing on intrusion detection, one may want to know if the input is an intrusion or belongs to the normal system behavior. One may also want to detect if this is a network intrusion or a software intrusion. Classification can be achieved for each input event or for the sequence of input data. When focusing on each input, one might consider using machine learning algorithms, like those explained later in the article. When focusing on sequences, Chandola et al. (Chandola and Vipin, 2012) distinguished three categories of techniques : kernel-based techniques, window-based technique and Markovian techniques (including Hidden Markov Models). In the field of Machine Learning, this is an example of supervised learning techniques.

— Clustering wants to find some relation between input data to create groups of elements. For instance, while tracing a system, a user could obtain network events, syscall events, and scheduler events. It is very likely that network events look similar, quite different from syscall events. In that case, clustering could be used to automatically split the input data into three sets : network events, syscall events, and scheduler events. Several techniques have been studied, and Jain distinguished hierarchical algorithms, partitional algorithms other techniques (Jain et al., 1999). The user may not know how many different groups should be used to split the data. Those techniques belong to the second category, unsupervised Machine Learning.

— Statistics could be used to detect anomalies if the user has a good knowledge of the system behavior and can easily quantify this behavior with figures. For instance, when the goal is to know whether there is an anomaly on a machine, one can monitor some metrics such as the CPU usage, the network latency, or the memory usage. When detecting that one of those metrics is above an arbitrary threshold, one can assume there is an anomaly on the system.

4.2.3 Tracing and debugging embedded devices

Several techniques can be used to collect data about a system, such as tracing, debugging or profiling. Tracing is a set of techniques that can provide a user with detailed information on a

system, either hardware or software. Tracing is different from profiling, as detailed in (Shende, 1999). While profiling aims at getting general statistics on a system, tracing gets a precise and timestamped log of the system execution when an event occurs. Therefore, tracing can be used to investigate short duration episodes, while profiling only detects changes that affect the system for a longer period of time. Tracepoints are small pieces of code for collecting information, that are available in instrumented kernels, such as the Linux kernel, or can be added in software. When the runtime hits a tracepoint, the tracer is called, and will record an event which contains information about the state of the system at the time the tracepoint was hit.

Several tracers exist with different functionality. One of the most effective for Linux systems is LTTng (Desnoyers et al.), that can trace both user space and kernel space (Fournier et al., 2018). This tool can help users to get detailed information about a system. It is very flexible, the user can choose which tracepoints he wants to monitor and produce a list of accurate timestamped events in a binary format called CTF (Common Trace Format). In addition, Desnoyers et al. explained how easy it was to port LTTng to new embedded architectures (Desnoyers and Dagenais, 2009). The main requirement for using LTTng is to have a Linux system, which is the case for most smart devices. Nonetheless, some tracers such as Barectf can be run on bare-metal systems (Proulx). This tracer has the advantage of producing CTF traces, and Bertauld et al. explained how it is possible to correlate Barectf traces and LTTng ones (Bertauld and Dagenais, 2017). With those tools, it is possible to efficiently trace most IoT devices.

4.3 Proposed solution

4.3.1 Architecture

To achieve the best intrusion detection, according to metrics that are specified later in the article, we propose a whole infrastructure, detailed in 4.2. It is composed of several sensors and actuators and an analysis system. The smart devices are running the tracer, while the analysis device aggregates the collected traces from devices, to detect anomalies and raise an alert when an intrusion is found. In addition, the analysis engine can be used to automatically correlate raised alerts and take action to prevent the intrusion. According to the definition of Zarpelão et al. (Zarpelão et al., 2017), this host-based intrusion detection system has a hybrid placement strategy, which enables us to use powerful analysis techniques to monitor various devices, while introducing very little overhead on each device. This architecture can be quickly integrated into a home automation system, since it only requires the use of a

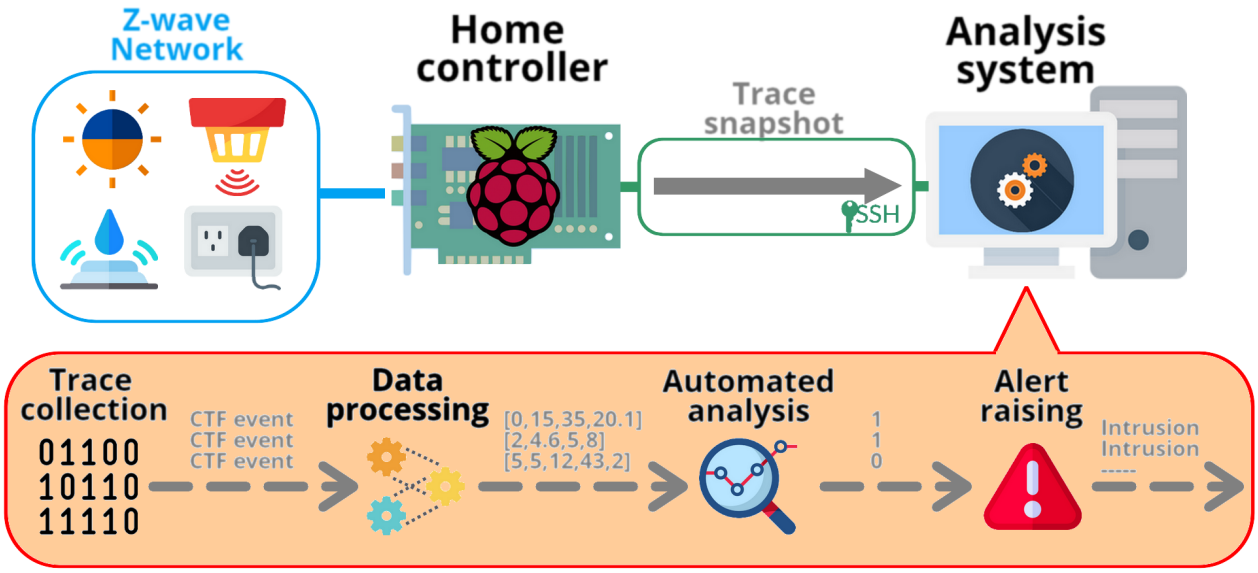

Figure 4.2 General Solution Architecture

tracer on the monitored devices, and the use of an analysis system that can run python. The analysis system can be another device or a server (inside the home or in the cloud). The heart of the solution is the analysis engine, which is composed of four main components : the trace aggregation engine, the data processing engine, the automated analysis engine, and the alert raising engine.

4.3.2 Trace collection

We rely on traces to detect intrusions. For more precision, we get multi-level information from the monitored systems, such as kernel-space and user-space data, hardware information and even network input. Furthermore, our tracer brings a minimal overhead, which is a major requirement for working in the IoT field. For a Linux-based smart device, the kernel is already instrumented (since version 2.6.32), and several powerful tracers are available. However, for devices running custom firmware, or for bare-metal systems, tracing can be done via barectf, a minimal tracer that is easy to port to such environments but that produces compatible traces in the CTF format. Consequently, since our HIDS only relies on tracing information, it could detect intrusions in most of the connected devices.

The LTTng tracer is known for its very low overhead (Murtaza et al., 2014b). Moreover, it provides detailed information on the systems, thanks to the huge number of tracepoints

available in the Linux kernel. It can quickly send trace data through the network in an optimized binary format called CTF. It is easy to enable or disable tracepoints. In recent LTTng releases, one can even apply filters to tracepoints, in order to only trace some events in specific interesting cases, optimizing the trace size. This enables the user to have a precise control on the data collected from the system. In addition, LTTng is able to get multi-level information on the system, such as syscalls, scheduler events, user-space events, network events and some hardware information, which helps to detect intrusions by monitoring the whole system.

Furthermore, LTTng can collect data through two modes : live monitoring, where the tracer sends trace data while it is being collected, or snapshot mode, where the tracer sends a snapshot of the current content of the trace buffer. Our solution can work with either of those two modes, but many devices such as sensors usually send information at regular intervals, not continuously. For instance, this can be the case with a light sensor, sending measurements of the room luminosity twice a second. This explains why we decided to use snapshot recording, since it is more suitable for sensors or actuators that only open communications at a certain frequency. Nonetheless, the proposed solution can be used with the live mode as well.

To simplify trace collection, we developed a tool able to generate an activation file from a configuration file. The activation file collects the specified tracepoints on the traced device and sends snapshots through an SSH tunnel to the analysis system, based on a user-specified frequency. This enables adapting for each device the information you want to collect, and therefore to optimize the intrusion detection in that device. The flexibility of our tool, and of the LTTng tracer, can easily accommodate every connected home device that runs a Linux-based kernel.

Our tool on the analysis system can in parallel receive snapshots from devices and perform the intrusion detection analysis, in order to detect intrusions as quickly as possible. It could also have been adapted to work with the LTTng live tracing mode, to detect intrusions in real time.

4.3.3 Data processing

The next step is to be able to feed Machine Learning algorithms, expecting vectors of metrics as input, with our collected binary CTF traces. For this purpose, we developed a complete toolchain that extracts features from CTF trace events, merge some features and create others, and finally convert string values to numeric ones. The toolchain architecture is described in 4.3.

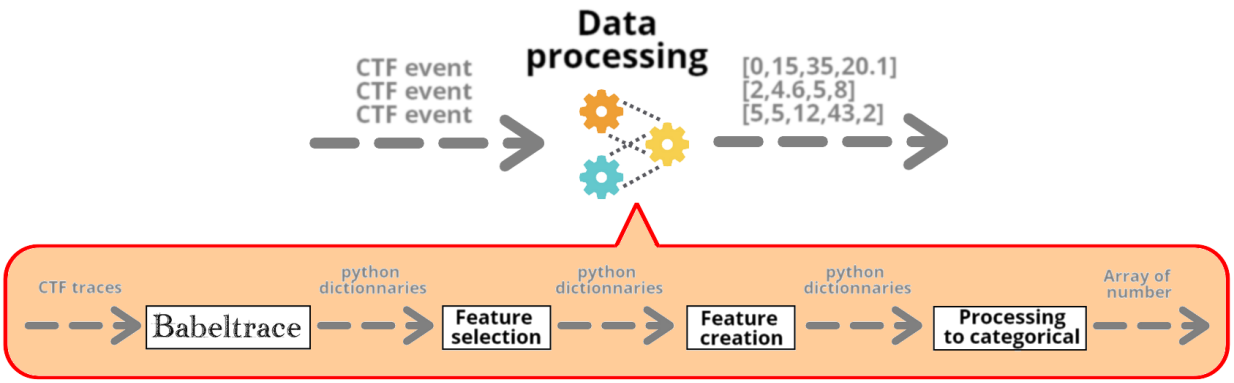

Figure 4.3 Data processing architecture

To be able to read CTF traces and extract the features, we rely on the Babeltrace API, an open-source tool developed by the same group that produced LTTng. This software is able to convert binary CTF traces to python dictionaries containing the event fields. Those will become features for the analysis algorithm, and other information such as statistics about the consumption of the device resources. Some information like the process identifiers (PID) are already numerical, others like the filename in an open syscall must be converted to a numerical identifier. We then have to select the fields that are used to detect intrusions and remove the others, to get a manageable dimensionality for our arrays. This is mandatory for most of the analysis algorithms. Indeed, the curse of dimensionality is a well-known machine learning issue that has to be taken into account when training a model.

The next step is to process extracted data from sensors, as explained by Chandola et al. (Chandola et al., 2009). To that extent, we need to improve upon our raw events, so that they contain relevant information, useful for the analysis engine. Consequently, we created a layer of abstraction by merging some events, such as entry_syscall_X and exit_syscall_X with a finite state machine (FSM), as proposed by Ezatti-Jivan et al. (Ezzati-Jivan and Dagenais, 2012), that can synthesize the information of two events into one and therefore enhance the detection. A user can quickly create new synthetic events with other FSMs, either by adding some short sections of code in the processing source code or via a configuration file. In addition, we also created new features in this step, like the duration of the event, based on the beginning and the end timestamps recorded by the tracer, or the current process name associated with the event, which can be retrieved by following the execution flow from the scheduling events. As a result, we obtain many categorical and non-categorical features that are used to detect intrusions on the device.

Thereafter, a one-hot encoder is used to convert the string values into numerical identifiers. One-hot encoding is a well-used step in data processing, as shown in Fig. 2 of Springenberg (Feurer et al., 2015). For instance, since the filename feature contained originally the filename string of the associated event, {filename : X}, we have to convert that string into a binary value that the algorithm can proceed, in this case, is_X. It will get the value 1 if the event is associated with filename X and 0 otherwise. This increases significantly the dimensionality of the array representing the collected events, but this information is crucial for the intrusion detection part. At the end of this step, we have an array which only contains categorical values, which can then feed the machine learning algorithm.

The resulting features are detailed in table 4.1

4.3.4 Automated analysis

The final step is to use the collected trace data to automatically detect intrusions on the device. For this, one must know the normal behavior of the connected device to be able to analyze a different behavior. While it might be easy to know the standard behavior of a device, either from the vendor or by manually checking the device behavior, it can require a lot of time and a deep knowledge about the device operation. Therefore, we propose an automated solution that may be efficient with a wide range of devices. This is possible with the use of machine learning techniques, learning the difference between a benign behavior and an Intrusion. Furthermore, it can be very complicated to manually define the rules characterizing a standard behavior, while machine learning will learn by itself the boundary between good behavior and intrusions, even in some sophisticated attacks where the attacker can mimic the standard system behavior (Wagner and Soto, 2002).

However, machine learning can require more work than specifying some intrusion detection rules. Indeed, any machine learning algorithm has to be fed with enough high-quality data to ensure accurate detection. The supervised learning algorithms also need the output classification of the training data to be able to learn what behavior is good or not. Consequently, we developed a tool able to create a dataset and automatically label it. This is done by tracing the device during two phases : first tracing the device in normal activity, then tracing the device under attack. The collected traces of the first step are then processed as described in the last paragraph, and the information about the events are stored in a database. This includes the network addresses and the ports used, the name of the process that generated the event, the name of the filename associated with an event, and more. The information stored in the database are precise enough to perform the detection at the event scale. Indeed, if the Home Assistant software has to access file /etc/passwd as part of the controller normal

Table 4.1 Post processing feature list

Feature	Type	Description
filename	string	Filename manipulated by the syscall
source_port	int	Port of the source network packet
dest_port	int	Port of the destination network packet
p_name	string	Name of the process that created the event
protocol	int	Value representing the used network protocol
parent_comm	string	Parent's process name
child_comm	string	Children's process name
pathname	string	Pathname manipulated by the syscall
ret	int	Value of return of the syscall
saddr	string	Address of the source network packet
daddr	string	Address of the destination network packet
d_timestamp	int	Event duration
a_nomEvent	string	Event name

behavior, then the information will indicate that the Home Assistant process has accessed this file, with the return value of this operation and its duration.

In the second phase, we collected traces of the device while subjected to attacks. To label the generated traces, we process the recorded events and then check for each event if its information is contained in the database of normal behavior. If so, we label the event as part of the good behavior. If not, we label it as an intrusion. We then store the processed event and the label in a CSV file, that will be the input for our algorithm. Thus, if an attack during this phase reads the /etc/passwd file from a bash script, this will not be found in the database of good behavior, since this file is not accessed in this way during normal operation. As a conclusion, the collected information is sufficiently large to distinguish between similar events in the normal and the intrusion case.

Once the dataset is generated, we can use different machine learning algorithms to find the boundary between the normal behavior and an intrusion. With this preprocessing, the events are suitable for feeding to many machine learning algorithms, including those of the popular open-source libraries Scikit-learn and Keras. Once our models have been trained with the dataset, we can trace the device again and give the collected processed events to the models. They can then detect attempted intrusions on the device.

4.3.5 Alert raising

The final stage of the analysis engine is to raise an alert whenever an anomaly is found, with information about the suspicious event. This is achieved with the display of an alert message

on the analysis system. In operation, such an alert would normally be fed to a monitoring system.

Although it has not yet been implemented, the alert engine could correlate the events associated with an intrusion to take actions to stop the intrusion. In the Mirai botnet case, once the alert has been raised, the analysis system could shut down the infected device and restart it, removing the injected piece of malware. It could also recognize the Mirai infection pattern and alert the user to change the devices passwords to prevent another infection.

4.4 Use Case

4.4.1 Attacks

To train our model, we decided to use real threats that IoT face today on our home assistant controller. We thus implemented a Mirai like infection, a scan of the IoT network, a hail mary attempt with Metasploit, a directory listing and a vulnerability scanning on the controller web interface, a basic ransomware, and a spying tool. Those attacks are described in the next paragraphs.

Our solution can work on every connected device that can run a CTF tracer, but we only studied it on the home controller since it was the only open-sourced system we had in our setup. To implement our solution, we installed the LTTng tracer on the home controller, and didn't modify the device in another way since the Linux kernel was yet instrumented with a lot of tracepoints.

First, we set up the Mirai botnet using its source code on GitHub, and we studied the traces that have been generated by this infection. Infected Mirai devices try to brute force other devices passwords with a list of default factory passwords. If it succeeds, it then downloads via the *busybox wget* command a malware from the C&C server, changes its name on the device, does a chmod and then runs it. For this reason, because we only wanted to trace the infection part, to be detected by our IDS, we implemented this attack with a modified malware that would just echo "Infected" on the device, once the malware process is created. We used a local workstation as the C&C server and compiled the Mirai source code to deploy it on this server. The workstation is a local PC running Debian 8 and being composed of an Intel i7-3770 CPU and a 16 GB RAM. It was connected to the targetted controller with a router and an Ethernet link. Then we launched the attack from the server that would bruteforce our home controller and install a malware on it. We only modified the C&C Mirai source code so that it would specifically attack our controller's IP address. Since a telnet server was enabled by default on the controller and the credentials (root, root) were working,

we didn't have to make another modification of the source code of the malware. As a result, the Mirai study is similar the real Mirai infection, since the C&C server was set up the same way the Mirai author has set up his, except the infected device will not download the real malware but our program. However, once infected by our program, and not the real Mirai malware targetting device, our controller was not able to launch a request against a targetted server on the command of the C&C server, as it would be possible in an infection with the real Mirai malware.

The second intrusion was a nmap scan to discover other devices on the traditional TCP/IP network. Indeed, whenever an attacker wants to exploit a network, he first has to survey the network to find potential targets. This is usually done with a network scan via the nmap tool. To be efficient against hacking attempts, our solution has to be able to detect such network activity when it comes from a monitored device.

Then, we used Metasploit to launch several exploits on our target. Since we patched the system with the latest security fixes and used up-to-date software, the Hail Mary process did not manage to exploit our controller. Nonetheless, the traces left by this attempt can then be used with our models, which will then be able to detect such attempts.

Then, we used two attacks against the web interface of the device. As OWASP highlighted in its Top 10 IoT Vulnerabilities (OWASP), it was the major attack vector : a directory listing done with the Dirbuster tool with its standard wordlist, and a vulnerability scanning with Nikto that did not find any vulnerability.

The next attack is a ransomware that encrypts all the files in a specified directory with AES 256 ECB mode. We only traced the exploitation phase, i.e. the encryption phase. This mimics the behavior of other ransomware such as Brickerbot (Wagner et al., 2017) that heavily targeted smart devices last year.

Finally, we developed a spying tool that records all the information sent to the home assistant controller via its API and resends that information to another server through the Internet. It is a specific malware that targets this device, and it does not change much the device behavior, unlike to the other intrusions.

4.4.2 Resulting dataset

As described earlier, we first traced the system while only issuing standard actions. We switched on a light via a Z-Wave command, powered on a smart plug with the controller web panel or activated a motion sensor so that a notification was sent to a remote smartphone and laptop. Our home automation system is only composed of affordable devices that are

described in the next section.

We traced the controller for one hour with a snapshot sent every 30 seconds, except the specific test where we took a snapshot of the controller every second. We then processed the recorded snapshot as described earlier, and created our events database. To reduce the risk of getting an imbalanced dataset, we removed several instances of events that were too frequent in our snapshots. Undersampling the dataset is a common practice in machine learning when dealing with an imbalanced dataset, as explained in Kotsiantis (Kotsiantis et al., 2006), but several other methods exist, as highlighted in the article.

The second tracing phase was with the device under attack. We took a snapshot of the trace buffers at every second to record every event associated with each attack. To label those events, we compared each with our database. Whenever the information of the event is already in the database, we remove the event. The events that do not match the database are labeled as intrusions.

Table 4.2 Enabled tracepoints list

Syscall	Network	Scheduler	Kernel module	CPU
access,chdir,chmod ,chown ,chroot,clone,close,connect, copy_file_range,creat,delete_module ,execve,execveat,exit,exit_group ,faccessat,fallocate,fchdir,fchmod ,fchmodat,fchown,fchownat,finit_module ,fork,getcpu,getcwd ,getdents,getdents64 ,getegid,geteuid,getgid,getpgid,getpgrp ,getpid,getppid,gettid,getuid,kexec_file_load ,kexec_load,kill,lchown,link,linkat,listen ,migrate_pages,mkdir,mkdirat,mount ,mq_unlink,open,openat,open_by_handle_at ,newstat,pipe,pipe2,pivot_root,read ,readahead,readlink,readlinkat,readv ,reboot,remap_file_pages,rename ,renameat,renameat2,restart_syscall ,rmdir,sched_getparam,setdomainname ,setfsgid,setfsuid,setgid,setgroups ,sethostname,setitimer,set_mempolicy ,setns,setpgid,setpriority,setregid ,setresgid,setresuid,setreuid,setrlimit ,set_robust_list,setsid,setsockopt ,set_tid_address,settimeofday,setuid ,setxattr,shutdown,socket,symlink ,symlinkat,sysctl,sysfs,sysinfo,syslog ,tgkill,tkill,umount,unlink,unlinkat,unshare ,vfork,write ,writev	net_dev_queue	sched_process_exit sched_switch, sched_process_fork sched_process_exec	module_load	power_cpu_frequency

For both tracing parts, we used several tracepoints, including syscall, network queue event, scheduler event, CPU events and kernel module events. An exhaustive list of the tracepoints

used is presented in table 4.2. We launched a new tracing session for each attack, ensuring that there was no event from a previous attack in a recorded snapshot. As a result, we have a dataset composed of 58% of benign events and 42% of intrusion-related events. The repartition of the various events is highlighted in Figure 4.4.

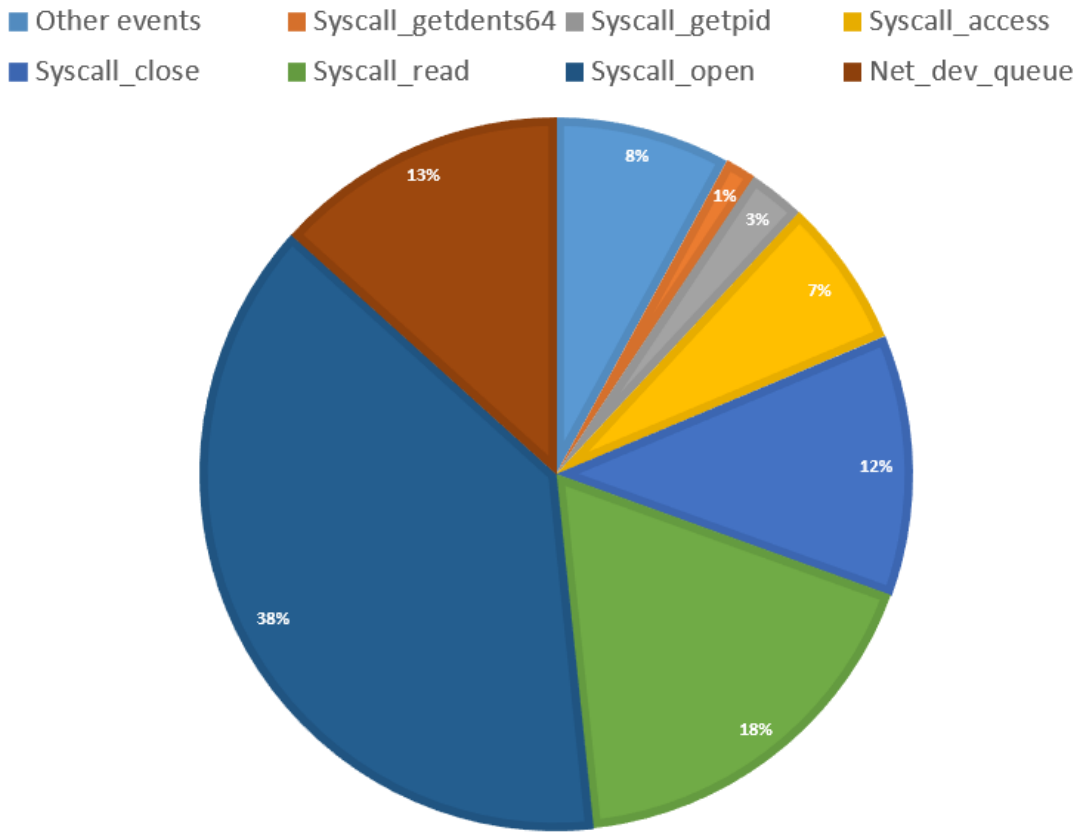

Figure 4.4 Dataset event repartition

4.4.3 Learning methodology

To train our models while avoiding over-fitting, we split our dataset between a training and a testing set, with 66% of the data used in the training set and 34% in the testing set. Splitting the dataset into training and testing sets is a common practice in machine learning, as explained by HSU (Hsu et al., 2003). There is, however, no consensus in the field about the optimal repartition. We used a k-fold cross-validation, which consists in splitting the dataset into k sets, and then compute the model learning on k-1 sets and test on the last set. The performance is measured by the average score of some metrics on the computed loop. To optimize the hyper-parameters of our various algorithms, we used a grid search,

which consists in testing all the combinations of various hyper-parameters and keeping those yielding the best score for their model. The metrics used to measure the efficiency of our algorithms are described later in this paper.

4.4.4 Classification algorithm

To enhance our detection performance, we implemented several algorithms that can be divided into different categories : supervised learning vs unsupervised learning or event-based detection vs sequence of events-based detection. Table 4.3 provides a quick overview of those algorithms. We decided to use both very recent deep neural network techniques and more traditional machine learning algorithms to measure the benefits of using newer methods, that are often more resource hungry than older ones. The algorithms we implemented use various methods to detect anomalies :

Table 4.3 Machine learning algorithm used

	Event classification	Sequence classification
Supervised learning	Decision Tree Random Forest GBT SVM MLP	
Unsupervised learning	OneClass SVM	LSTM

— Decision Trees (DT) use a graph like structure to classify the dataset by creating nodes from previous nodes that maximize the purity of each node. We use Gini Impurity as a criterion to split the data in each node and maximize the information gain.
— Random forest (RF) is a bagging ensemble method where several decision trees are independently created during training. Each tree is created in a random sub-ensemble of the dataset with a random number of features. Each tree will then vote for a class, and the class with the maximum number of voters will be associated with the input.
— Gradient Boosted Trees (GBT) is a boosting ensemble method where classifiers are built sequentially to reduce the bias and the variance of the global classifier. We use decision trees as the basics estimators.
— Support Vector Machines (SVM) is a set of techniques that aim at finding the boundary between the different classes by mapping the data into another space. To predict new data, the same mapping is used.
— MultiLayer Perceptron (MLP) is a feed-forward artificial neural network where several layers of neurons are connected that can separate even not linearly separable data.

— One Class SVM is a particular use-case of SVM techniques where the classifier only learns the boundary of one class, here the standard behavior of the system. Then, whenever an event is outside of the determined boundary, it is classified as an anomaly. This technique is able to detect new intrusions since we do not need to train this algorithm with attacks samples.

— Long Short-Term Memory (LSTM) is a recurrent neural network, meaning that there are some recurrent connections in the network. Thus, recurrent networks have two sources of input : present data and data that has already been through the network. Therefore, the prediction of an entry event will affect the prediction of the later. Each unit of an LSTM network is composed of an LSTM cell, which has the ability to store, write or read a piece of information, according to the input it receives. Thus, LSTM networks are able to classify a sequence of data.

4.5 Evaluation

4.5.1 Set up

To implement our solution, we used several devices that would typically be used in a home automation system. Our home automation controller, the device responsible for the action of actuators according to data received from the sensors, is a Raspberry Pi 3, an ARM based embedded system running Linux. This device has the exact same hardware characteristics than the HomeSeer HomeTroller Zee S2, another home automation controller. Our Raspberry Pi 3 runs Home Assistant, a modified Rasberry Pi Debian distribution with one of the most popular open-source home automation platforms. Home Assistant is compatible with many smart devices and IoT protocols. To get a representative home automation network, we plugged a Z-Wave USB stick, the Z-Stick Gen5 from AEOTEC. Our controller can then manage z-wave sensors and actuators.

We used several sensors to be able to get data from the environment, with the Multisensor 6 from AEOTEC, a z-wave device that can monitor motion, temperature, light, humidity, and vibration. As an actuator for our home automation system, we used the smart switch 6 from Aeotec. It can be turned on or off via a z-wave command and can monitor the energy consumption of its electrical outlet.

In order to get a realistic domotic system, we set up some rules on the home assistant controller. For instance, whenever the light is above 166 Lumix, i.e. the light is switched on, the controller will send a notification to a remote desktop and a smartphone through the Pushbullet API, a service aiming at sending a message to multiple platforms. Another

example is a rule that switched on the smart plug when the motion sensor detects activity.

The analysis machine is a Debian based computer with an Intel i7-3770 CPU clocked at 3.40GHz with 8 cores. It has 16 GB RAM and is connected to the same VLAN as the home controller.

4.5.2 Metrics

To evaluate the efficiency of our detection engine, we used several metrics commonly employed in model evaluation. Those metrics are defined in the next paragraphs in the context of binary classification, since we want to predict good or abnormal behavior.

In our binary classification problem, True Positive (TP) is the number of correctly classified events as positive, True Negative (TN) is the number of correctly classified events as negative, False Positive (FP) is the number of falsely classified events as positive and False Negative (FN) is the number of falsely classified events as negative.

The common information to get from a model is its accuracy, which is the ratio between the number of good prediction and the total number of predictions. It can be defined by equation 4.1.

$$Accuracy = \frac{TP + TN}{N_{events}} \tag{4.1}$$

However, accuracy is not enough to determine a model efficiency. Indeed, a high accuracy cannot assure that the model will have a great detection power, especially if the dataset is imbalanced. Even if our dataset is balanced, it can still be useful to consider the Precision and Recall metrics, which can help to select among the different algorithms.

When it comes to classifying an input, the precision tells how many of the predicted events of a class were correctly predicted. This is the ratio between good predictions and the total number of events from a class. Recall reports how many of the classified events that should have been selected were actually selected. Since both of those metrics are relevant regarding our detection problem, we also introduced the F1-score, which is based on the precision and recall of the system. Those metrics can be obtained from equation 4.2

$$
\begin{aligned}
Recall &= \frac{TP}{TP + FN} \\
Precision &= \frac{TP}{TP + FP} \\
F1 &= 2 * \frac{Precision * Recall}{Precision + Recall}
\end{aligned}
\tag{4.2}
$$

We want to maximize those metrics which reflect the detection performance of our system. As described earlier, we computed our metrics 5 times with 5-fold validation, with a training dataset and a testing dataset.

4.5.3 Tracing overhead

To measure the tracing overhead with LTTng, we used the sysbench benchmark first while tracing was enabled, and second while it was not. We used the integrated CPU benchmark with the following arguments. In this computation, the system verifies prime numbers by dividing the number with all values between 2 and the square root of the number. The argument –cpu-max-prime sets the highest number to verify during the benchmark.

```
$ sysbench --test=cpu --cpu-max-prime=20000 run
```

We traced our device with the tracepoints described in table 4.2. This contains 108 syscalls (including open, close, access, write, fork, chmod, chown, mkdir), network queue with net dev queue kernel event, scheduler events (switch, process fork, process exit), kernel module (load and free) and some CPU events (frequency). We analyzed the system overhead while tracing and sending a snapshot through the network by varying the duration between two snapshots. The benchmark was repeated ten times, the mean results are presented in table 4.4.

Moreover, we used the sysbench memory benchmark, which allocates a 1 kB buffer that will be read or written, sequentially or randomly, until the memory-total-size argument is reached. As before, we ran our benchmark ten times and present in table 4.5 the mean results. We used this specific command :

```
$ sysbench --test=memory --memory-total-size=20G run
```

Table 4.4 CPU overhead according to snapshot frequency

Snapshot frequency	Total time	Average time (per request)	95 percentile time (per request)
No tracing	371.51 s	37.15 ms	37.23 ms
0.5 s	+ 1.52 %	+ 1.53 %	+ 7.49 %
1 s	+ 1.15 %	+ 1.15 %	+ 6.28 %
2 s	+ 0.80 %	+ 0.81 %	+ 4.78%
3 s	+ 0.50 %	+ 0.51 %	+ 3.82 %
4 s	+ 0.44 %	+ 0.45 %	+ 2.94%
5 s	+ 0.39 %	+ 0.39 %	+ 2.21 %
10 s	+ 0.25 %	+ 0.25%	+ 1.24 %

Table 4.5 Memory overhead according to snapshot requency

Snapshot frequency	Memory overhead
No tracing	3.75 s
0.5 s	+ 2.73 %
1 s	+ 1.82 %
2 s	+ 2.25 %
3 s	+ 2.52 %
4 s	+ 2.31 %
5 s	+ 1.99 %
10 s	+ 1.87 %

These results highlight the low overhead of the LTTng tracer. Indeed, we traced a powerful device connected to an electrical outlet. Moreover, because the overhead introduced by tracing is low, our solution can be used even on limited resources devices. The CPU overhead can be reduced by decreasing the snapshot frequency. This will not, however, reduce the memory overhead, which is about 2.21 % for this application.

4.5.4 Training efficiency

While we focus on the previously defined metrics to measure the efficiency of our models, we also noticed that some of our models needed significantly more computing time than others during the training phase. Indeed, the MLP took an 8 times longer training phase, while GBT needed more than 6 000 times the duration. The testing phase results are presented in the next table.

While those results are very good, we have to insure that our models did not overfit our training set and can predict well future input events.

4.5.5 Analysis efficiency

Precision, recall, and F1 score

As explained earlier, we want to maximize the precision, recall and F1 score of our detection engine. The computation of those efficiency metrics has been done with the scikit-learn built-in function *score*. The test set results are the average of the five cross-validations and are presented in Table 4.6.

Table 4.6 Anomaly detection efficiency of various algorithm

	Accuracy	F1	Precision	Recall
DT	99.97%	99.96%	99.98%	99.93%
RF	99.99%	99.99%	99.99%	99.99%
GBT	100%	99.99%	99.99%	99.99%
MLP	57.85%	23.07%	46.78%	27.25%
SVM	53.32%	99.99%	47.33%	99.99%

Those results highlight two major points. First, the choice of the algorithm has a great impact on the detection accuracy. While our results emphasize that some algorithms could provide a top detection quality, some others fail at detecting intrusions on our dataset. This can be explained by the heterogeneity of the events that make our dataset. Indeed, as explained earlier, each field of a processed event is a feature for the machine learning algorithms. However, events have very different fields, as explained in figure. Some algorithms such as Decision Trees (DT), Random Forest (FT) or Gradient Boosted Trees (GBT) are very robust to that kind of data issues, while others like MultiLayer Perceptron (MLP) and Support Vector Machines (SVM) are not. We could apply some data transformations, like removing some fields or putting some default values, even normalizing some fields like the source IP address, but then we would have lost some relevant knowledge from the event. Indeed, it is essential to know the origin of the intrusion to have an adapted response to the security event.

Secondly, we can achieve very high detection rates with our solution. Indeed, because a smart device has been designed to have a specific behavior, it is easier than for traditional computers to learn this behavior, thus to detect any anomaly. Furthermore, it prevents an intruder from breaking our system using mimicry attacks that have been described by Sabahi et al. (Sabahi and Movaghar, 2008). Our framework can precisely detect various kinds of intrusions while avoiding to raise to many false alarms or failing to detect intrusions.

Syscall_open : {filename : "sudoTracing.sh", d_timestamp :1780004334 , pathname : "/etc/login.defs", p_name : "sudoTracing.sh"}

Syscall_read : {filename : "sudoTracing.sh", d_timestamp :1780004334, p_name : "sudoTracing.sh"}

...

Syscall_readlinkat : {pathname : "/proc/sudoTracing.sh/exe", d_timestamp : 1780004334, p_name : "system-journal"}

...

Net_dev_queue : {p_name : "sshd", d_timestamp : 890024167, source_port : 22, dest_port : 41640, saddr : 1322077235, daddr : 1322077219 }

Figure 4.5 Relevant fields of processed events

Analysis latency

To measure the analysis latency, we generated new snapshots containing a basic attack that every classifier studied was able to detect. This attack only run a freshly downloaded bash script which echoes "Infected" every 0.5 seconds. The created events have not been used in the training dataset or in the testing dataset ; this is novel data for the classifiers. We defined the analysis latency as the mean time required for each classifier to predict the class of one event, making our measure independent of the network latency or the snapshot frequency. However, the more often snapshots are recorded, the quicker the intrusion alert will be raised, but the snapshot frequency should not be higher than the time needed to classify each event of the previously received snapshot. In most cases, we want to reduce the analysis latency to detect a security event as soon as possible. However, some users may want to run many analysis engines at the same time to do alert correlation when the priority is to reduce the number of false alarms.

The total detection latency can be defined as follows, if the analysis engine has been previously loaded, where the network latency and the intrusion event raised (and its time) can vary :

$$\begin{aligned} Latency_{detection} = Latency_{network} + \\ Time_{intrusion_event} - Time_{first_event} \end{aligned} \tag{4.3}$$

The results are exposed in Table 4.7.

These results highlight the need to carefully select the anomaly detection model. Indeed, GBT provides better results than the DT algorithm, but the time needed to predict the class of an event is nearly 7.5 times more than the former algorithm.

Table 4.7 Analysis latency

	Avg. time per event
DT	1.23 ms
RF	1.68 ms
GBT	9.28 ms
OneClass SVM	6.79 ms
MLP	2.16 ms

Discussion

Our results showed that some algorithms can provide better detection performance than others, but may require more time for learning or for classification.

Indeed, Decision Trees, Random Forest and Gradient Boosted Trees can provide really great intrusion detection capabilities, based on tracing events, but those algorithms cannot detect intrusions that have not been previously learned. This is unlike OneClass SVM, which is much slower than other algorithms to predict the class of the incoming events.

Furthermore, only LSTM focuses on sequences of events, which can help detect some complicated intrusions. For instance, the attacker may attempt a mimicry attack, even if this is more complicated on smart devices than on traditional systems.

Moreover, our results, with our enabled set of tracepoints, show that DT based algorithms (including GBT of RF) provide very efficient intrusion detection capabilities, because they are very robust with input events that have different features. However, if the monitored events have nearly the same structure, for instance when only syscall events are being monitored, some algorithms may provide better results, especially deep-neural network based algorithms.

Finally, tracing the home controller is very useful to detect attacks on the domotic system. For instance, it can detect attempts to eavesdrop information with a malware similar to the one tested. This attack can be very difficult to detect when only relying on the network information, if the covert channel is efficient enough. However, tracing the other devices could really improve the performance of our solution since it could prevent more threats, such as ransomware on a smart TV or a Brickerbot attack on an alarm system.

4.5.6 Limitation

Our solution can be quickly integrated into home automation systems since it only requires having a tracer sending traces in CTF format, such as LTTng or BareCTF. However, those tools rely on the TPC/IP stack, which is a limitation for some device. Indeed, some connected

devices lack such connections and instead rely on Z-Wave, Zigbee or Insteon to send data in smart homes. Those protocols work in mesh networks where each device can send and forward information, and where the network topology can vary with time. This is very different from traditional TCP/IP networks.

Most of the algorithm that worked well with our experiments are supervised learning techniques, implying that they can only detect attacks similar to what they learned. As a consequence, another limitation of our work is the number of attacks that have been learned, and the lack of real malware being used. Indeed, since most IoT threats are not open-source, we could only create attacks that mimic the behavior of real-world threats, like we did with our ransomware and spying tools. Collecting malware samples targeting our monitored device could have been achieved with a honeypot emulating ARM hardware and the same OS, for instance by running Home Assistant on a QEMU virtual host.

In addition, to enhance our detection capabilities, we could have tried to develop a misuse detection engine to combine with our anomaly detection engine. This would require more powerful hardware for the analysis system, and could threaten the detection speed.

4.6 Conclusion and Future Work

In this paper, we presented a complete novel and flexible framework to detect intrusions on smart devices. It combines several machine learning algorithms and tracing techniques on the monitored devices. Our solution has shown very accurate intrusion detection results. We explained how it could be tuned to adapt to different devices, and explained why this solution performs well, benefiting from its host-based approach.

Although we achieved great intrusion detection precision with our solution, some work can be done to improve our system usability. One future work would be to adapt the open-source code of LTTng to support IoT protocols such as Z-Wave, ZigBee, and Insteon, and to study the traffic created with this tool in such a mesh network.

It would be interesting to study the scalability of the analysis engine. Indeed, since the recorded snapshots can be analyzed independently from each other, our solution can be parallelized and scale to numerous devices, in small or bigger networks. However, the resources necessary to scale the monitored networks should be studied. The solution presented obtained excellent results and was optimized for quick detection. Further study could focus on the learning step processing time, since the learning phase has to be updated frequently to account for always evolving new threats. Another interesting area for further study is to select different models according to the targeted device, and even to combine more than one

algorithm to improve further the detection accuracy.

4.7 ACKNOWLEDGMENT

We would like to gratefully acknowledge the Natural Sciences and Engineering Research Council of Canada (NSERC), Prompt, Ericsson, Ciena, and EfficiOS for funding this project.

CHAPITRE 5 DISCUSSION GÉNÉRALE

Les résultats présentés dans l'article précédent montrent que notre solution est capable de détecter de manière très efficace des intrusions sur des objets connectés. Nous avons eu recours à différentes techniques d'apprentissage machine, et certaines se révèlent excellentes pour atteindre notre objectif. De plus, la solution a un impact très limité sur la performance des objets surveillés.

Les paragraphes suivants précisent certaines informations liées aux résultats obtenus et détaillent les limitations soulevées dans l'article.

5.1 Retour sur les résultats

5.1.1 Jeux de données générés

Le jeu de donnée principal utilisé pour l'apprentissage a été stocké au format CSV et est disponible en source libre sur Github [1], tout comme l'ensemble des outils développés dans le cadre des travaux de recherche. Il a été séparé en jeux d'entraînement et de test composés respectivement de 66% et 34% du jeu de donnée total. Bien qu'il n'y ait pas de consensus dans la communauté sur la répartition des données entre les jeux d'entraînement et de test, il semble que cette répartition soit l'une des plus courantes.

Nous avons également utilisé un autre jeu de données pour déterminer la latence de détection de chacun des algorithmes. Le fait de choisir des données inédites pour cette mesure permet d'éviter toute optimisation en mémoire de l'exécution des algorithmes. Cela permet également de se rapprocher d'un cadre d'utilisation réel du système de détection d'intrusion, où les tentatives d'intrusions ne seront pas exactement semblables à celles contenues dans le jeu de donnée.

La sélection des évènements utilisés pour détecter les intrusions ainsi que de leurs *métriques* a été effectuée de manière empirique en analysant le comportement du contrôleur domotique et des points de traces disponibles. Ainsi, plusieurs champs d'évènements ne sont pas remplis dans beaucoup d'autres évènements, comme les adresses sources et destinations qui ne sont présentes que dans les évènements réseau. Ces informations étant pertinentes pour détecter des comportements malveillants, nous avons décidé de les conserver, bien que certains algorithmes que nous avons utilisés soient très peu robustes à ce type de données hétérogènes.

1. https ://github.com/Blouglou2/TracingIDS

5.1.2 Apprentissage machine

Nous avons montré que les algorithmes basés sur des arbres de décision offrent les meilleures performances de détection avec les données que nous avons récoltées et la façon dont nous les avons transformées. Cependant, les autres algorithmes utilisés dans nos travaux pourraient obtenir de meilleures performances avec un traitement différent des évènements. Notre parti-pris était de modifier les évènements au minimum pour ne pas augmenter les délais de détection, mais nous avons également conservé certains champs d'évènements peu fréquemment remplis. Ainsi, plusieurs techniques comme la suppression de ces champs ou la normalisation des valeurs des différents champs des évènements pourraient être envisagées pour améliorer la qualité de la détection des autres algorithmes, mais elles requièrent une plus grande puissance de calcul et un travail d'optimisation important.

5.1.3 Efficacité du traçage

Nous avons montré que notre architecture de traçage était très performante et introduisait une faible baisse des performances du système. Nos mesures ont été menées avec un outil à source libre et conduites onze fois pour chaque cas envisagé. Cependant, nos travaux ont été menés sur un Raspberry Pi 3 et, bien que cette plateforme ait été utilisée dans de nombreux projets d'objets connectés, la plupart des capteurs et des actionneurs actuellement présents sur le marché disposent de caractéristiques matérielles moins performantes. Ainsi, la baisse de performance introduite par notre solution pourrait s'avérer plus importante sur certains systèmes, même si le relativement faible impact de notre solution sur le contrôleur laisse présager de la bonne efficacité de notre solution sur les autres plateformes connectées.

5.2 Scénarios d'attaque

Notre solution s'est révélée efficace pour détecter les attaques que nous avons implémentées. Ces dernières ont été sélectionnées selon plusieurs scénarios :

— Un attaquant profite de la faible sécurisation des objets. Cela correspond aux attaques par force brute sur les mots de passe comme celle mise en oeuvre par Mirai, aux tentatives d'exploitations automatiques à l'aide de l'outil metasploit ou les attaques ciblant l'interface web du contrôleur. L'attaquant passe ici à l'action, car il a une grande opportunité pour réussir ses tentatives d'intrusion.

— Le scénario suivant met en scène un attaquant qui se sert des objets connectés comme d'un point de pivotage dans le réseau de sa cible. Cela a été illustré par l'implémentation d'un scan réseau partir d'un objet infecté par l'attaquant. Ici également,

l'opportunité de l'attaquant grandit avec l'introduction d'objets connectés non sécurisés.

— Le cas d'attaque suivant repose sur un attaquant qui a envie de nuire de manière économique à ses victimes. Ce cas a été implémenté à l'aide du rançon-logiciel que nous avons développé.
— Le dernier scénario met en scène un attaquant qui cible spécifiquement une victime. Dans cette optique, nous avons développé un outil malveillant ciblant spécifiquement notre contrôleur et ses interfaces de programmation, avec notre logiciel espion.

Il est important de noter que l'implémentation de l'attaque relative au botnet Mirai a été permise car son code source a été publié librement sur Internet. Nous nous sommes intéressé à la phase d'infection de cette célèbre attaque sur les objets connectés et avons seulement modifié son code source pour qu'il attaque spécifiquement notre contrôleur domotique. Nous avons mis en place un serveur de commande et de contrôle, et implémenté l'attaque qui a trouvé le mot de passe de notre cible par force brute et obtenu un accès à cette dernière via telnet. Ensuite, un virus a été téléchargé depuis le serveur de commande et de contrôle et a été executé avec les plus hauts privilèges. Puisque nous n'étudiions que la phase d'infection de l'attaque Mirai, nous avons remplacé le virus téléchargé par un simple programme affichant "Infecté" sur un terminal. La phase d'infection implémentée est donc totalement identique à l'infection dans le monde réel de Mirai, et nous n'avons pas étudié les attaques lancées par un objet infecté du type *TCP flood* et autre.

Nos travaux ont démontré l'efficacité de notre solution et de son implémentation sur l'ensemble de ces scénarios d'attaque.

CHAPITRE 6 CONCLUSION ET RECOMMENDATIONS

Ce chapitre final revient sur les objectifs initiaux du travail, expose ses limitations et conclut sur la qualité de la solution que nous avons proposée. Nous détaillons également quelques travaux futurs à mener pour augmenter un peu plus l'intérêt du projet.

6.1 Synthèse des travaux

Notre objectif principal de recherche était d'étudier la faisabilité de la mise en oeuvre d'un système de détection d'intrusion basé sur le comportement des objets connectés. De nombreuses contraintes liées à l'écosystème des objets connectés ont alors été soulevées. Nous désirions de plus explorer diverses techniques d'apprentissage machine pour parvenir à améliorer la sécurité des objets connectés, sans avoir à étudier le comportement de chacun des objets disponibles sur le marché. La solution que nous avons proposée remplit parfaitement ces critères avec des outils pouvant être déployés facilement sur les objets connectés, et une architecture permettant de simplement spécifier les points de trace à surveiller et l'algorithme d'apprentissage à utiliser pour être fonctionnelle. Nous avons montré qu'elle offre de très bonnes performances de détection avec une validation expérimentale.

La revue de la littérature a souligné les différents enjeux liés à la sécurité des objets connectés. Nous avons ainsi montré les difficultés d'implémenter des solutions de sécurité efficaces et générales pour ces systèmes, particulièrement en utilisant des informations liées au comportement de l'objet lui-même. Nous avons également détaillé les différents travaux réalisés à l'aide d'apprentissage machine pour détecter des intrusions, et mis en lumière la performance des techniques de traçage pour récolter des informations sur un système. Cela nous a conduit à recourir au traceur LTTng qui offrait les meilleures performances pour notre projet, et nous avons pu développer une solution tirant parti des fonctionnalités avancées du traceur, tout en poursuivant les efforts menés dans le cadre des travaux sur la détection d'intrusion.

Nous avons ensuite détaillé l'environnement utilisé dans le cadre de nos travaux. Nous avons pris le parti de disposer d'un enviromment matériel et logiciel similaire à celui qui est présent dans la plupart des maisons connectées afin de pouvoir évaluer la faisabilité du déploiement de notre solution ainsi que de vérifier ses performances. Les critères de performance ont été sélectionnés pour évaluer l'impact sur le bon fonctionnement du système surveillé et la qualité de la détection. Cela a également pour but de faciliter la reproductibilité de nos travaux dans le cadre de futurs projets.

Ensuite, nous avons détaillé notre solution et les outils que nous avons proposés. Tous ces outils ont été proposés en source libre [1]. Nous avons développé une architecture logicielle capable de tracer un objet connecté, d'envoyer les informations enregistrées sur une machine d'analyse et de débuter cette analyse pendant qu'elle reçoit d'autres traces. De plus, nous avons fourni des outils permettant de générer automatiquement des jeux de données et de les labéliser dans le but d'utiliser des techniques d'apprentissage machine. Cette labélisation ne nécessite que de tracer le système dans son comportement sain et de réitérer ce traçage dans des conditions d'attaque. Enfin, nous avons proposé des outils permettant d'utiliser des traces pour nourrir des algorithmes d'apprentissage machine. Des *métriques* sont ainsi extraites des traces recueillies, certaines sont créées et d'autres modifiées, dans le but de parvenir à des meilleures capacités de détection. Les données sont ensuite converties en vecteurs de nombres pouvant être utilisé avec les librairies d'apprentissage machine les plus populaires du langage Python, comme Scikit-learn ou Keras. Nous avons également mesuré la performance de notre solution et des divers algorithmes implémentés.

Nous avons enfin apporté plus d'informations sur les résultats obtenus et avons détaillé les principales causes de différence de performance des modèles que nous avons entraînés. Nous avons fourni quelques recommandations d'implémentation pour améliorer les performances de certains modèles.

6.2 Limitations de la solution proposée

Outre les limitations détaillées dans l'article, le travail d'optimisation de nos outils déjà réalisé pourrait être poussé plus loin. En effet, nous avons parallélisé certaines tâches gourmandes de traitement des évènements reçus sur le serveur d'analyse, ainsi que la classification de ces évènements par les algorithmes, mais nous avons eu recours aux fils d'exécution du langage Python. D'après diverses études, la gestion de ces fils n'est pas optimale puisqu'elle repose sur un autre fil qui utilise de nombreux verrous. Une solution à ce problème serait d'implémenter la gestion des fils d'exécution au niveau du système d'exploitation ou avec des librairies plus performantes, par exemple en langage C++.

De plus, pour optimiser nos outils, il serait intéressant de déléguer certains calculs à l'interpréteur, notamment avec un recours plus important à la fonction """map""" du langage Python, ou d'utiliser des implémentations de python plus performantes, comme PyPy qui implémente la compilation JIT.

1. https ://github.com/Blouglou2/TracingIDS

6.3 Améliorations futures

Nous avons déjà souligné l'intérêt d'implémenter l'envoi des traces sur les protocoles de communication spécifiques à l'Internet des objets, comme les protocoles Z-Wave ou Zigbee dans la section 4 de ce mémoire.

De plus, il serait intéressant d'étudier la fréquence des *snapshot* idéale à envoyer à la machine d'analyse en fonction du type d'objet à surveiller. De même, une étude du nombre d'objets à tracer en fonction de la taille du réseau d'objets connectés ou du type d'attaque que l'on souhaite pouvoir détecter contribuerait à l'intérêt de la solution proposée.

Enfin, une piste d'amélioration de la solution proposée serait de combiner nos outils de détection d'anomalies avec des outils de détection d'intrusion par règles, qui pourraient accélérer la détection dans des cas triviaux. Ces outils pourraient également s'appuyer sur des statistiques telles que la fréquence des appels système pour lever rapidement une alerte en cas de détection d'intrusion. De plus, il serait judicieux de continuer à développer notre solution pour qu'elle puisse prendre des actions en fonction des intrusions détectées. Ainsi, ce nouvel outil pourrait décider de redémarrer l'objet qui a été identifié comme étant infecté par le *botnet* Mirai, ce qui aurait pour effet de supprimer le logiciel malveillant. De plus, le système pourrait prévenir l'utilisateur de la menace et lui recommander de changer le mot de passe par défaut. La solution résultante ne serait ainsi plus un système de détection d'intrusion, mais un système de prévention d'intrusion.

RÉFÉRENCES

"CVE-2009-4273", https://www.cvedetails.com/cve/CVE-2009-4273/, 2010.

"SigFox basics", http://www.rfwireless-world.com/Terminology/SIGFOX-technology-asics.html.

"Using SystemTap", http://dtrace.org/blogs/brendan/2011/10/15/using-systemtap/, 2011.

"A Brief Technology Overview of the Lighting Control Marketplace X-10, Insteon, Z-Wave, ZigBee, RadioRA2, Universal Powerline Bus (UPB)", http://www.simply-automated.com/documents/white_papers/SAIWhitePaperTechnologyhomeautomationNov212014.pdf.

"What is LoRaWAN", https://www.link-labs.com/blog/what-is-lorawan.

"Cybersecurity Considerations For Connected Smart Home Systems And Devices", https://library.ul.com/wp-content/uploads/sites/40/2017/03/CS10027_White_Paper-Web_02.pdf.

Y. Aafer, W. Du, et H. Yin, "Droidapiminer : Mining api-level features for robust malware detection in android", dans *International conference on security and privacy in communication systems*. Springer, 2013, pp. 86–103.

F. J. Acosta Padilla, E. Baccelli, T. Eichinger, et S. K, "The Future of IoT Software Must be Updated", 06 2016.

F. J. Acosta Padilla, E. Baccelli, T. Eichinger, et K. Schleiser, "The Future of IoT Software Must be Updated", dans *IAB Workshop on Internet of Things Software Update (IoTSU)*. Dublin, Ireland : Internet Architecture Board (IAB), Jun 2016. En ligne : https://hal.inria.fr/hal-01369681

F. A. Alaba, M. Othman, I. A. T. Hashem, et F. Alotaibi, "Internet of Things security : A survey", vol. 88, no. Supplement C, 2017, pp. 10 – 28. DOI : https://doi.org/10.1016/j.jnca.2017.04.002. En ligne : http://www.sciencedirect.com/science/article/pii/S1084804517301455

T. Anantvalee et J. Wu, *A Survey on Intrusion Detection in Mobile Ad Hoc Networks*.

Boston, MA : Springer US, 2007, pp. 159–180. DOI : 10.1007/978-0-387-33112-6_7. En ligne : https://doi.org/10.1007/978-0-387-33112-6_7

M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis *et al.*, "Understanding the mirai botnet", dans *USENIX Security Symposium*, 2017, pp. 1092–1110.

D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, et C. Siemens, "DREBIN : Effective and Explainable Detection of Android Malware in Your Pocket." dans *Ndss*, vol. 14, 2014, pp. 23–26.

L. Atlas, R. Cole, Y. Muthusamy, A. Lippman, J. Connor, D. Park, M. El-Sharkawai, et R. Marks, "A performance comparison of trained multilayer perceptrons and trained classification trees", vol. 78, no. 10. IEEE, 1990, pp. 1614–1619.

L. Atzori, A. Iera, et G. Morabito, "The Internet of Things : A survey", vol. 54, no. 15, 2010, pp. 2787 – 2805. DOI : https://doi.org/10.1016/j.comnet.2010.05.010. En ligne : http://www.sciencedirect.com/science/article/pii/S1389128610001568

S. Babar, P. Mahalle, A. Stango, N. Prasad, et R. Prasad, "Proposed security model and threat taxonomy for the Internet of Things (IoT)", dans *International Conference on Network Security and Applications*. Springer, 2010, pp. 420–429.

S. Babar, A. Stango, N. Prasad, J. Sen, et R. Prasad, "Proposed embedded security framework for internet of things (IoT)", dans *Wireless Communication, Vehicular Technology, Information Theory and Aerospace & Electronic Systems Technology (Wireless VITAE), 2011 2nd International Conference on*. IEEE, 2011, pp. 1–5.

L. M. Belue et K. W. Bauer Jr, "Determining input features for multilayer perceptrons", vol. 7, no. 2. Elsevier, 1995, pp. 111–121.

T. Bertauld et M. R. Dagenais, "Low-level trace correlation on heterogeneous embedded systems", vol. 2017, no. 1, Jan 2017, p. 18. DOI : 10.1186/s13639-016-0067-1. En ligne : https://doi.org/10.1186/s13639-016-0067-1

S. Bharadwaja, W. Sun, M. Niamat, et F. Shen, "Collabra : a xen hypervisor based collaborative intrusion detection system", dans *Information technology : New generations (ITNG), 2011 eighth international conference on*. IEEE, 2011, pp. 695–700.

T. Bird, "Measuring function duration with ftrace", dans *Proceedings of the Linux Symposium*. Citeseer, 2009, pp. 47–54.

M. P. Brown, W. N. Grundy, D. Lin, N. Cristianini, C. W. Sugnet, T. S. Furey, M. Ares, et D. Haussler, “Knowledge-based analysis of microarray gene expression data by using support vector machines”, vol. 97, no. 1. National Acad Sciences, 2000, pp. 262–267.

Z. U. Burguera, Iker et S. Nadjm-Tehrani, “Crowdroid : Behavior-based Malware Detection System for Android”, dans *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, série SPSM ’11. New York, NY, USA : ACM, 2011, pp. 15–26. DOI : 10.1145/2046614.2046619. En ligne : http://doi.acm.org/10.1145/2046614.2046619

J. Cannady, “Artificial neural networks for misuse detection”, dans *National information systems security conference*, vol. 26. Baltimore, 1998.

B. S. Center, “Extrae”, https://tools.bsc.es/extrae.

B. A. K. Chandola, Varun et Vipin, “Anomaly detection for discrete sequences : A survey”, vol. 24, no. 5. IEEE, 2012, pp. 823–839.

V. Chandola, A. Banerjee, et V. Kumar, “Anomaly Detection : A Survey”, vol. 41, no. 3. New York, NY, USA : ACM, Jul 2009, pp. 15 :1–15 :58. DOI : 10.1145/1541880.1541882. En ligne : http://doi.acm.org/10.1145/1541880.1541882

S.-T. Cheng, C.-H. Wang, et y. p. Gwo-Jiun Horng, journal=2008 3rd International Conference on Sensing Technology, “OSGi-Based Smart Home Architecture for heterogeneous network”.

C. Y. Chung, M. Gertz, et K. Levitt, *DEMIDS : A Misuse Detection System for Database Systems*. Boston, MA : Springer US, 2000, pp. 159–178. DOI : 10.1007/978-0-387-35501-6_12. En ligne : https://doi.org/10.1007/978-0-387-35501-6_12

M. Conti, A. Dehghantanha, K. Franke, et S. Watson, “Internet of Things security and forensics : Challenges and opportunities”, vol. 78, no. Part 2, 2018, pp. 544 – 546. DOI : https://doi.org/10.1016/j.future.2017.07.060. En ligne : http://www.sciencedirect.com/science/article/pii/S0167739X17316667

H. Debar, M. Dacier, et A. Wespi, “Towards a taxonomy of intrusion-detection systems”, vol. 31, no. 8, 1999, pp. 805 – 822. DOI : https://doi.org/10.1016/S1389-1286(98)00017-6. En ligne : http://www.sciencedirect.com/science/article/pii/S1389128698000176

J. Desfossez, “Analyses scripts for LTTng kernel and user-space traces”, https://github.com/lttng/lttng-analyses, 2018.

M. Desnoyers et M. Dagenais, “Deploying LTTng on exotic embedded architectures”, dans *Embedded Linux Conference*, vol. 2009, 2009.

M. Desnoyers et M. R. Dagenais, “The lttng tracer : A low impact performance and behavior monitor for gnu/linux”, dans *OLS (Ottawa Linux Symposium)*, vol. 2006. Citeseer, 2006, pp. 209–224.

M. Desnoyers, M. R. Dagenais, et École Polytechnique De Montréal, “The LTTng tracer : A low impact performance and behavior monitor for GNU/Linux Mathieu Desnoyers”.

R. Díaz-Uriarte et S. A. De Andres, “Gene selection and classification of microarray data using random forest”, vol. 7, no. 1. BioMed Central, 2006, p. 3.

G. Dini, F. Martinelli, A. Saracino, et D. Sgandurra, “MADAM : a multi-level anomaly detector for android malware”, dans *International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security*. Springer, 2012, pp. 240–253.

J. Edge, “A look at ftrace”, https://lwn.net/Articles/322666/, 2009.

F. C. Eigler et R. Hat, “Problem solving with systemtap”, dans *Proc. of the Ottawa Linux Symposium*, 2006, pp. 261–268.

J. Elith, J. R. Leathwick, et T. Hastie, “A working guide to boosted regression trees”, vol. 77, no. 4. Wiley Online Library, 2008, pp. 802–813.

S. M. Erfani, S. Rajasegarar, S. Karunasekera, et C. Leckie, “High-dimensional and large-scale anomaly detection using a linear one-class SVM with deep learning”, vol. 58. Elsevier, 2016, pp. 121–134.

S. Eskandari, W. Khreich, S. S. Murtaza, A. Hamou-Lhadj, et M. Couture, “Monitoring system calls for anomaly detection in modern operating systems”, dans *2013 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, Nov 2013, pp. 19–20. DOI : 10.1109/ISSREW.2013.6688856

D. Evans, “The Internet of Things : How the Next Evolution of the Internet Is Changing Everything”, 2011. En ligne : https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf

N. Ezzati-Jivan et M. R. Dagenais, “A Stateful Approach to Generate Synthetic Events from Kernel Traces”, vol. 2012. New York, NY, United States : Hindawi Publishing Corp., Jan 2012, pp. 6 :6–6 :6. DOI : 10.1155/2012/140368. En ligne : http://dx.doi.org/10.1155/2012/140368

A. Fendall, T. D. Hunt, D. Rajendran, et M. Nikora, “Assisted Living : Domestic Power Monitoring utilising Home Automation Products and Cloud Storage”. CITRENZ, 2015.

M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, et F. Hutter, “Efficient and Robust Automated Machine Learning”, dans *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, et R. Garnett, éds. Curran Associates, Inc., 2015, pp. 2962–2970. En ligne : http://papers.nips.cc/paper/5872-efficient-and-robust-automated-machine-learning.pdf

B. Fouladi et S. Ghanoun, “Security evaluation of the Z-Wave wireless protocol”, vol. 24, 2013, pp. 1–2.

T. R. P. Foundation, “Raspberry Pi 3 model B”, vol. 6, 3, p. 2018. En ligne : https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

P.-M. Fournier, M. Desnoyers, et M. R. Dagenais, “Combined tracing of the kernel and applications with LTTng”, dans *Proceedings of the 2009 linux symposium*. Citeseer, 2009, pp. 87–93.

P.-M. Fournier, M. Desnoyers, et M. R Dagenais, “Combined tracing of the kernel and applications with LTTng”, 06 2018.

J. H. Friedman, “Stochastic gradient boosting”, vol. 38, no. 4. Elsevier, 2002, pp. 367–378.

Y. Ganjisaffar, R. Caruana, et C. V. Lopes, “Bagging gradient-boosted trees for high precision, low variance ranking models”, dans *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*. ACM, 2011, pp. 85–94.

J. C. Gower et G. J. Ross, “Minimum spanning trees and single linkage cluster analysis”. JSTOR, 1969, pp. 54–64.

B. Gregg, “Strace Wow Much Syscall”, http://www.brendangregg.com/blog/2014-05-11/strace-wow-much-syscall.html.

B. Gregg et J. Mauro, *DTrace : dynamic tracing in oracle Solaris, Mac OS X and freeBSD*. Prentice Hall Professional, 2011.

J. Gubbi, R. Buyya, S. Marusic, et M. Palaniswami, “Internet of Things (IoT) : A vision, architectural elements, and future directions”, vol. 29, no. 7, 2013, pp. 1645 – 1660, including Special sections : Cyber-enabled Distributed Computing for Ubiquitous Cloud and Network Services & Cloud Computing and Scientific Applications — Big Data, Scalable Analytics, and Beyond. DOI : https://doi.org/10.1016/j.future.2013.01.010. En ligne : http://www.sciencedirect.com/science/article/pii/S0167739X13000241

J. Ham, Y. Chen, M. M. Crawford, et J. Ghosh, “Investigation of the random forest framework for classification of hyperspectral data”, vol. 43, no. 3. IEEE, 2005, pp. 492–501.

S. Hochreiter et J. Schmidhuber, “Long Short-Term Memory”, vol. 9, no. 8. Cambridge, MA, USA : MIT Press, Nov. 1997, pp. 1735–1780. DOI : 10.1162/neco.1997.9.8.1735. En ligne : http://dx.doi.org/10.1162/neco.1997.9.8.1735

S. A. Hofmeyr, S. Forrest, et A. Somayaji, “Intrusion detection using sequences of system calls”, vol. 6, no. 3. IOS Press, 1998, pp. 151–180.

R. G. Hollands, “Will the real smart city please stand up ? Intelligent, progressive or entrepreneurial ?” vol. 12, no. 3. Taylor & Francis, 2008, pp. 303–320.

C.-W. Hsu, C.-C. Chang, C.-J. Lin *et al.*, “A practical guide to support vector classification”. Taipei, 2003.

T. K. Hui, R. S. Sherratt, et D. D. Sánchez, “Major requirements for building Smart Homes in Smart Cities based on Internet of Things technologies”, vol. 76, no. Supplement C, 2017, pp. 358 – 369. DOI : https://doi.org/10.1016/j.future.2016.10.026. En ligne : http://www.sciencedirect.com/science/article/pii/S0167739X16304721

A. K. Jain, M. N. Murty, et P. J. Flynn, “Data Clustering : A Review”, vol. 31, no. 3. New York, NY, USA : ACM, Sep 1999, pp. 264–323. DOI : 10.1145/331499.331504. En ligne : http://doi.acm.org/10.1145/331499.331504

M. H. Jim Keniston, Prasanna S Panchamukhi, “Kernel Probes (Kprobes)”, https://www.kernel.org/doc/Documentation/kprobes.txt.

T. Joachims, “Text categorization with support vector machines : Learning with many relevant features”, dans *European conference on machine learning*. Springer, 1998, pp. 137–142.

P. Kannadiga et M. Zulkernine, “DIDMA : A distributed intrusion detection system using mobile agents”. IEEE, 2005, pp. 238–245.

C. Kolias, G. Kambourakis, A. Stavrou, et J. Voas, “DDoS in the IoT : Mirai and Other Botnets”, *Computer*, vol. 50, no. 7, pp. 80–84, 2017. DOI : 10.1109/MC.2017.201

A. P. Kosoresow et S. A. Hofmeyer, “Intrusion detection via system call traces”, vol. 14, no. 5, Sep 1997, pp. 35–42. DOI : 10.1109/52.605929

S. Kotsiantis, D. Kanellopoulos, P. Pintelas *et al.*, “Handling imbalanced datasets : A review”, vol. 30, no. 1, 2006, pp. 25–36.

D. Kozlov, J. Veijalainen, et Y. Ali, “Security and Privacy Threats in IoT Architectures”, dans *Proceedings of the 7th International Conference on Body Area Networks*, série BodyNets ’12. ICST, Brussels, Belgium, Belgium : ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2012, pp. 256–262. En ligne : http://dl.acm.org/citation.cfm?id=2442691.2442750

S. Kumar et E. H. Spafford, “A pattern matching model for misuse intrusion detection”, 1994.

H. Laurent et R. L. Rivest, “Constructing optimal binary decision trees is NP-complete”, vol. 5, no. 1, 1976, pp. 15–17.

I. Lee et K. Lee, “The Internet of Things (IoT) : Applications, investments, and challenges for enterprises”, vol. 58, no. 4. Elsevier, 2015, pp. 431–440.

K.-L. Li, H.-K. Huang, S.-F. Tian, et W. Xu, “Improving one-class SVM for anomaly detection”, dans *Machine Learning and Cybernetics, 2003 International Conference on*, vol. 5. IEEE, 2003, pp. 3077–3081.

W. Ma, P. Duan, S. Liu, G. Gu, et J.-C. Liu, “Shadow attacks : automatically evading system-call-behavior based malware detection”, vol. 8, no. 1, May 2012, pp. 1–13. DOI : 10.1007/s11416-011-0157-5. En ligne : https://doi.org/10.1007/s11416-011-0157-5

J. Malik et R. Kaushal, “CREDROID : Android malware detection by network traffic analysis”, dans *Proceedings of the 1st ACM Workshop on Privacy-Aware Mobile Computing*. ACM, 2016, pp. 28–36.

L. M. Manevitz et M. Yousef, “One-class SVMs for document classification”, vol. 2, no. Dec, 2001, pp. 139–154.

C. Modi, D. Patel, B. Borisaniya, H. Patel, A. Patel, et M. Rajarajan, “A survey of intrusion detection techniques in Cloud”, vol. 36, no. 1, 2013, pp.

42 – 57. DOI : https://doi.org/10.1016/j.jnca.2012.05.003. En ligne : http://www.sciencedirect.com/science/article/pii/S1084804512001178

A. Montplaisir-Gonçalves, N. Ezzati-Jivan, F. Wininger, et M. R. Dagenais, "State history tree : an incremental disk-based data structure for very large interval data", dans *Social Computing (SocialCom), 2013 International Conference on*. IEEE, 2013, pp. 716–724.

G. Mulligan, "The 6LoWPAN architecture", dans *Proceedings of the 4th workshop on Embedded networked sensors*. AC, 2007, pp. 78–82.

S. S. Murtaza, A. Sultana, A. Hamou-Lhadj, et M. Couture, "On the Comparison of User Space and Kernel Space Traces in Identification of Software Anomalies", dans *2012 16th European Conference on Software Maintenance and Reengineering*, March 2012, pp. 127–136. DOI : 10.1109/CSMR.2012.23

S. S. Murtaza, A. Hamou-Lhadj, W. Khreich, et M. Couture, "Total ADS : Automated Software Anomaly Detection System", dans *2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation*, Sept 2014, pp. 83–88. DOI : 10.1109/SCAM.2014.37

S. S. Murtaza, A. Sultana, A. Hamou-Lhadj, et M. Couture, "On the comparison of user space and kernel space traces in identification of software anomalies", dans *Software Maintenance and Reengineering (CSMR), 2012 16th European Conference on*. IEEE, 2012, pp. 127–136.

S. S. Murtaza, A. Hamou-Lhadj, W. Khreich, et M. Couture, "Total ADS : Automated software anomaly detection system", dans *Source Code Analysis and Manipulation (SCAM), 2014 IEEE 14th International Working Conference on*. IEEE, 2014, pp. 83–88.

T. Nam et T. A. Pardo, "Conceptualizing smart city with dimensions of technology, people, and institutions", dans *Proceedings of the 12th annual international digital government research conference : digital government innovation in challenging times*. ACM, 2011, pp. 282–291.

H. Nemati, S. D. Sharma, et M. R. Dagenais, "Fine-grained Nested Virtual Machine Performance Analysis Through First Level Hypervisor Tracing", dans *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, série CCGrid '17. Piscataway, NJ, USA : IEEE Press, 2017, pp. 84–89. DOI : 10.1109/CCGRID.2017.20. En ligne : https://doi.org/10.1109/CCGRID.2017.20

M. Nobakht, V. Sivaraman, et R. Boreli, “A Host-Based Intrusion Detection and Mitigation Framework for Smart Home IoT Using OpenFlow”, dans *2016 11th International Conference on Availability, Reliability and Security (ARES)*, Aug 2016, pp. 147–156. DOI : 10.1109/ARES.2016.64

OWASP, “OWASP Top 10 IoT Vulnerabilities”. En ligne : https://www.owasp.org/index.php/Top_IoT_Vulnerabilities

M. Pal, “Random forest classifier for remote sensing classification”, vol. 26, no. 1. Taylor & Francis, 2005, pp. 217–222.

S. K. Pal et S. Mitra, “Multilayer Perceptron, Fuzzy Sets, Classifiaction”, 1992.

R. Patel et A. Rajwat, “A survey of embedded software profiling methodologies”, 2013.

P. Proulx, “Tracing bare-metal systems : a multi-core story - LTTng”. En ligne : https://lttng.org/blog/2014/11/25/tracing-bare-metal-systems/

V. F. Rodriguez-Galiano, B. Ghimire, J. Rogan, M. Chica-Olmo, et J. P. Rigol-Sanchez, “An assessment of the effectiveness of a random forest classifier for land-cover classification”, vol. 67. Elsevier, 2012, pp. 93–104.

M. Roesch *et al.*, “Snort : Lightweight intrusion detection for networks.” dans *Lisa*, vol. 99, no. 1, 1999, pp. 229–238.

R. Romera, “Home automation and Cybercrime”, http://apac.trendmicro.com/cloud-content/apac/pdfs/security-intelligence/white-papers/wp-home-automation-and-cybercrime.pdf, 2017.

F. Sabahi et A. Movaghar, “Intrusion Detection : A Survey”, dans *2008 Third International Conference on Systems and Networks Communications*, Oct 2008, pp. 23–26. DOI : 10.1109/ICSNC.2008.44

S. R. Safavian et D. Landgrebe, “A survey of decision tree classifier methodology”, vol. 21, no. 3, 1991, pp. 660–674.

H. Sak, A. Senior, et F. Beaufays, “Long short-term memory recurrent neural network architectures for large scale acoustic modeling”, dans *Fifteenth annual conference of the international speech communication association*, 2014.

U. Saxena, J. S. Sodhi, et t. y. v. n. p. k. d. I. m. Y. Singh, booktitle=2017 7th International Conference on Cloud Computing, Data Science Engineering - Confluence.

R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang, et S. Zhou, "Specification-based Anomaly Detection : A New Approach for Detecting Network Intrusions", dans *Proceedings of the 9th ACM Conference on Computer and Communications Security*, série CCS '02. New York, NY, USA : ACM, 2002, pp. 265–274. DOI : 10.1145/586110.586146. En ligne : http://doi.acm.org/10.1145/586110.586146

E. Y. Shapiro, *Algorithmic Program Debugging. ACM Distinguished Dissertation.* MIT press, 1982.

S. Shebs, "GDB tracepoints, redux". Citeseer, 2009, pp. 105–112.

S. Shende, "Profiling and Tracing in Linux", 1999.

A. K. Sikder, G. Petracca, H. Aksu, T. Jaeger, et A. S. Uluagac, "A Survey on Sensor-based Threats to Internet-of-Things (IoT) Devices and Applications", vol. abs/1802.02041, 2018.

S. Skorobogatov, "How Microprobing Can Attack Encrypted Memory", dans *2017 Euromicro Conference on Digital System Design (DSD)*, Aug 2017, pp. 244–251. DOI : 10.1109/DSD.2017.69

C. Strobl, A.-L. Boulesteix, A. Zeileis, et T. Hothorn, "Bias in random forest variable importance measures : Illustrations, sources and a solution", vol. 8, no. 1. BioMed Central, 2007, p. 25.

V. Svetnik, A. Liaw, C. Tong, J. C. Culberson, R. P. Sheridan, et B. P. Feuston, "Random forest : a classification and regression tool for compound classification and QSAR modeling", vol. 43, no. 6. ACS Publications, 2003, pp. 1947–1958.

K. S. Tai, R. Socher, et C. D. Manning, "Improved semantic representations from tree-structured long short-term memory networks", 2015.

K. Tam, S. J. Khan, A. Fattori, et L. Cavallaro, "CopperDroid : Automatic Reconstruction of Android Malware Behaviors." dans *NDSS*, 2015.

C.-F. Tsai, Y.-F. Hsu, C.-Y. Lin, et W.-Y. Lin, "Intrusion detection by machine learning : A review", vol. 36, no. 10, 2009, pp. 11 994 – 12 000. DOI : https://doi.org/10.1016/j.eswa.2009.05.029. En ligne : http://www.sciencedirect.com/science/article/pii/S0957417409004801

V. Vapnik et S. Mukherjee, “Support vector method for multivariate density estimation”, dans *Advances in neural information processing systems*, 2000, pp. 659–665.

C. Wagner, A. Dulaunoy, G. Wagener, et S. Mokkadem, “An extended analysis of an IoT malware from a blackhole network”, 06 2017.

D. Wagner et P. Soto, “Mimicry Attacks on Host-based Intrusion Detection Systems”, dans *Proceedings of the 9th ACM Conference on Computer and Communications Security*, série CCS ’02. New York, NY, USA : ACM, 2002, pp. 255–264. DOI : `10.1145/586110.586145`. En ligne : `http://doi.acm.org/10.1145/586110.586145`

J. Weidendorfer, “Sequential Performance Analysis with Callgrind and KCachegrind”, dans *Tools for High Performance Computing*, M. Resch, R. Keller, V. Himmler, B. Krammer, et A. Schulz, éds. Berlin, Heidelberg : Springer Berlin Heidelberg, 2008, pp. 93–113.

D. S. Wilks, “Cluster analysis”, dans *International geophysics*. Elsevier, 2011, vol. 100, pp. 603–616.

F. Wortmann et K. Flüchter, “Internet of things”, vol. 57, no. 3. Springer, 2015, pp. 221–224.

B. B. Zarpelão, R. S. Miani, C. T. Kawakani, et S. C. de Alvarenga, “A Survey of Intrusion Detection in Internet of Things”. Elsevier, 2017.

B. B. Zarpelão, R. S. Miani, C. T. Kawakani, et S. C. de Alvarenga, “A survey of intrusion detection in Internet of Things”, vol. 84, 2017, pp. 25 – 37. DOI : `https://doi.org/10.1016/j.jnca.2017.02.009`. En ligne : `http://www.sciencedirect.com/science/article/pii/S1084804517300802`

M. Zheng, M. Sun, et J. Lui, “Droidanalytics : a signature based analytic system to collect, extract, analyze and associate android malware”, 2013.