Évaluation à l'échelle de l'Internet du niveau d'exposition des objets connectés face aux risques de sécurité

Pierre-Marie Junges

UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Évaluation à l'échelle de l'Internet du niveau d'exposition des objets connectés face aux risques de sécurité

THÈSE

présentée et soutenue publiquement le 7 juillet 2022

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Pierre-Marie Junges

Composition du jury

Rapporteurs :	Vincent Nicomette	Professeur, INSA Toulouse
	Valérie Viet Triem Tong	Professeure, CentraleSupélec Rennes
Examinateurs :	Noura Limam	Chargée de recherche, Université de Waterloo
	Michaël Rusinowitch	Directeur de recherche, Inria Nancy - Grand Est (Président du jury)
Encadrants :	Olivier Festor	Professeur, Université de Lorraine (Loria) (Directeur de thèse)
	Jérôme François	Chargé de recherche, Inria Nancy - Grand Est (Co-directeur de thèse)

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thesul.

Remerciements

Quelle aventure !

Telle une série Netflix, mes quatre années de thèse auront été pleines de rebondissements, entrecoupées par une pandémie à l'échelle mondiale. Mais quelle expérience de vie !

Tout cela n'aurait pas été possible sans Jérôme François et Olivier Festor, mes deux directeurs de thèse. Je vous remercie très sincèrement de m'avoir accueilli dans l'équipe RESIST, et de m'avoir accompagné au cours de ma thèse. Sans vos conseils et votre soutien, la plupart de mes travaux n'auraient pas vu le jour.

Je souhaite également remercier les rapporteurs et examinateurs de cette thèse d'avoir accordé de leur temps à mes travaux de recherche et pour avoir accepté de faire partie de mon jury de thèse.

Je tiens également à remercier les membres de l'équipe RESIST que j'ai pu côtoyer, et avec lesquels j'ai toujours pu échanger dans une ambiance conviviale. Je remercie particulièrement Isabelle Chrisment pour ses discussions bienveillantes qui m'ont permis de voir mes problèmes sous un autre angle.

Merci à tous les collègues et amis que j'ai eu la chance d'avoir au cours de ma vie de thésard. En particulier Thomas Lacour, Adrien Hemmer, Frédéric Beck, Loïc Rouch, Mingxiao Ma, Abir Laraba.

Enfin, je remercie ma famille de m'avoir accompagné et soutenu durant mes années de thèse. Mes remerciements les plus sincères à mes parents pour leur soutien quotidien et de m'avoir procuré un environnement propice pour mes études.

À toutes et à tous, merci.

À mes parents,
mes soeurs.

Sommaire

Introduction générale

Contexte

De nos jours, les objets de l'Internet des Objets (IdO) sont massivement déployés dans la domotique ou l'industrie 4.0. Depuis 2015, leur nombre est en augmentation, estimé à plus de 35 milliards en 2021 [11]. D'ici 2025, chaque personne disposera, en moyenne, de 9 objets IdO [139]. En parallèle de cette croissance fulgurante, les objets IdO sont devenus des cibles privilégiées des attaquants. D'après un rapport de NETSCOUT [1] publié en 2018, un objet IdO reçoit une attaque généraliste dans les 5 premières minutes, puis une attaque spécifique à l'objet dans les 24 heures qui suivent sa connexion à l'Internet.

Une fois compromis, ces objets IdO sont utilisés pour propager des attaques, comme renseigné dans un rapport de SYMANTEC [2] en 2019, où il est indiqué que plus de 90% des cyberattaques ciblant les objets IdO sont effectuées par d'autres objets IdO, comme des routeurs ou des caméras connectées.

Les objets IdO peuvent également être utilisés par des attaquants, ou botnets, pour effectuer des attaques à large échelle. Parmi ces botnets [91, 13, 107], nous pouvons citer Bashlite ou Mirai qui ont compromis, chacun, plusieurs centaines de milliers d'objets IdO en effectuant une attaque par *brute-force* à l'aide d'un dictionnaire d'identifiants et de mots de passe par défaut utilisés dans les objets IdO. Le second botnet, Mirai, a notamment défrayé la chronique en 2016 en effectuant une attaque par déni de service générant plus de 600 Go par seconde [98]. En plus des attaques par *brute-force*, les objets IdO souffrent naturellement de vulnérabilités associées aux binaires présents dans ces derniers. Pour détecter en amont des déploiements ces vulnérabilités, des analyses statiques ou dynamiques appliquées aux binaires ont été proposées. Selon un rapport d'Hewlett-Packard (HP) [3] datant de 2014, les 10 objets IdO les plus vendus souffrent, en moyenne, de 25 vulnérabilités chacun.

La sécurité d'objets IdO doit être étudiée selon plusieurs axes. Il est possible d'effectuer des tests d'intrusions sur un objet IdO physique mais, compte tenu du large nombre d'objets IdO disponibles, cette approche pose des problèmes de passage à l'échelle. Analyser les firmwares d'objets IdO permet de résoudre ce problème, car ces derniers sont disponibles en ligne depuis le site de leur fabricant. À partir d'un firmware, il est ensuite possible de connaître l'intégralité des binaires et des fichiers de configuration utilisés par un objet IdO. Quand cela est possible, une analyse de firmwares permet d'obtenir une vue détaillée de la sécurité d'un objet IdO. Elle permet d'évaluer la sécurité de n'importe quel binaire présent dans le firmware et donc, de détecter de nouvelles vulnérabilités. Cependant l'analyse de firmwares doit faire face à plusieurs chal-

1. Disponible sur `https://www.netscout.com/sites/default/files/2019-02/SECR_001_EN-1901%20-%20NETSCOUT%20Threat%20Intelligence%20Report%202H%202018.pdf`, dernier accès le 09/05/2022
2. Disponible sur `https://docs.broadcom.com/doc/istr-24-2019-en`, dernier accès le 09/05/2022
3. Disponible sur `https://www.hp.com/us-en/hp-news/press-release.html?id=1744676`, dernier accès le 09/05/2022

lenges, comme l'absence de composants matériels ou l'hétérogénéité des architectures processeur utilisées, ce qui rend difficile l'exécution des binaires présents dans le firmware.

Nous évaluons la sécurité des firmwares d'objets IdO, en inspectant les vulnérabilités connues associées aux binaires présents dans les firmwares d'objets IdO au moment de leur publication. De plus, nous mesurons également si les binaires présents sont mis à jour ou non par les fabricants. Notre approche est innovante par rapport à l'état de l'art, car elle permet d'avoir une vue globale des binaires déployés dans l'IdO, et notamment d'observer les objets IdO déployés avec des vulnérabilités connues ou les vagues de correctifs appliqués aux firmwares.

Préalablement à une cyberattaque, la phase de reconnaissance permet à un attaquant d'inspecter les machines connectées à un réseau local, ou à l'Internet, afin de détecter des machines potentiellement exploitables. Cette phase est généralement effectuée par des méthodes d'identification actives (ou *active fingerprinting* en anglais) pour détecter des propriétés d'une machine connectée, comme ses services ou son système d'exploitation. Cependant, ces méthodes nécessitent de nombreuses interactions avec la cible ce qui réduit la furtivité de la phase de reconnaissance. De plus, les propriétés inférées sont souvent imprécises ou incomplètes.

Nous proposons une méthode d'identification active élaborée à partir des données extraites de firmwares d'objets IdO. Cette approche est originale, mais théorique, car elle ne nécessite pas d'étudier le trafic réseau d'objets IdO physiques et permet d'inférer des propriétés précises, comme la version des binaires ou les mots de passe présents dans l'objet.

À la suite de cette phase de reconnaissance, les objets potentiellement exploitables sont attaqués. Pour étudier les cyberattaques, et ainsi appréhender le comportement des attaquants, les pots de miel peuvent être employés. Ces derniers sont des systèmes informatiques conçus spécifiquement pour recevoir, puis étudier, des cyberattaques. Dans notre cas d'étude qu'est l'IdO, nous avons défini un pot de miel à forte interaction simulant le fonctionnement d'un objet IdO à partir de son firmware. Or, ces objets déploient des services différents et utilisent également des composants matériels hétérogènes, ce qui rend leur simulation difficile mais possible via l'émulation. Les méthodes d'émulation d'objets IdO existantes modifient le firmware de l'objet à émuler et ces modifications peuvent être détectées par un attaquant averti. C'est pourquoi notre solution intègre des fonctionnalités permettant de cacher ces modifications aux attaquants, et ainsi intercepter les cyberattaques le plus furtivement possible.

Dans le cas où un objet IdO n'a pas son firmware de disponible en ligne, il est toujours possible d'analyser son trafic réseau afin de mettre en évidence un autre problème de sécurité : le risque de fuite de données utilisateur. En effet, les objets IdO peuvent également souffrir de ce type d'attaques [52, 41, 15, 8] qui permet à un attaquant de déduire les activités effectuées par un utilisateur sur ses objets IdO, comme la fermeture des volets roulants connectés ou l'ouverture d'une serrure connectée. Seule l'analyse du trafic réseau permet de détecter ces attaques. À la différence des travaux existants où il est possible d'observer le trafic réseau d'un objet IdO, nous nous sommes intéressés au cas 1) d'un attaquant localisé dans l'Internet, et 2) d'une box domotique connectée à plusieurs objets. Cette dernière sert de relai à un utilisateur pour interagir avec ses objets IdO, et son trafic réseau traverse l'Internet et peut donc être intercepté par un attaquant. Nous proposons une méthode qui, à partir du trafic réseau chiffré d'une box domotique, infère les actions utilisateur effectuées vers un ou plusieurs objets IdO associés à la box. Notre méthode est dite passive, car elle consiste uniquement à observer le trafic réseau entre la box domotique et l'Internet.

Problématiques

Compte tenu du grand nombre d'objets IdO mis sur le marché, et de leur hétérogénéité dans leur fonctionnalité, leur marque ou leur protocole de communication, les contributions présentées dans ce manuscrit doivent répondre à un certain nombre de contraintes.

Dans le cas de notre méthode d'identification passive, notre positionnement dans l'Internet fait qu'il est impossible d'intercepter les communications sans-fil des objets IdO connectés à une box domotique. D'ailleurs même en étant à proximité des objets IdO, il est possible que ces derniers utilisent un protocole de communication sans-fil propriétaire ne permettant pas d'analyser directement leur trafic réseau. Notre solution doit ainsi répondre aux problèmes suivants :

— le chiffrement des données qui empêche d'analyser le contenu des données échangées entre une box domotique et l'Internet,

— l'abstraction induite par la box domotique. Le trafic réseau de la box domotique nous empêche de déterminer le ou les objets destinataires des paquets reçus par la box. Par exemple, si un paquet vers une box domotique est destiné à une caméra connectée ou à une prise connectée.

Notre analyse de firmwares doit être flexible, car les binaires présents dans les firmwares d'objets IdO peuvent être compilés pour des architectures processeurs diverses (comme MIPS ou ARM), et le contenu des binaires peut différer suivant les firmwares étudiés. De plus, compte tenu du grand nombre de firmwares à analyser, notre approche doit être capable de s'adapter à chaque firmware sans action spécifique.

Définir une méthode d'identification active à partir des données extraites de firmwares d'objets IdO est difficile et implique les contraintes suivantes :

— Furtivité : les interactions supposées avec l'objet ciblé doivent être limitées afin que notre méthode soit utilisée lors de la phase de reconnaissance d'une attaque sans être détectée.

— Précision des propriétés à identifier : Plus les informations prédites par notre méthode sont précises (par exemple le nom et la version d'un binaire), plus les attaques générées peuvent être spécifiques et efficaces.

— Évolutivité : les objets IdO étant de plus en plus nombreux, notre méthode doit être capable d'identifier les propriétés associées à ces nouveaux objets rapidement.

Enfin, l'émulation d'objets IdO et leur utilisation en tant que pots de miel doivent également répondre aux problèmes suivants :

— Furtivité : le pot de miel ne doit pas être détecté par des attaquants ainsi, les fonctionnalités implémentées dans ce dernier ne doivent pas révéler l'existence du pot de miel aux attaquants.

— Évolutivité : l'émulation d'objets IdO requiert de nombreux scripts pour simuler son fonctionnement à partir de son firmware. Ainsi, l'infrastructure logicielle (ou *framework*) proposée se doit d'être facilement maintenable, ou modifiable selon les objectifs d'émulation de l'utilisateur.

— Performances d'émulation : la furtivité peut limiter les performances d'émulation. Si tel est le cas, ces limitations doivent être aussi faibles que possibles.

Contributions

Quatre contributions sont présentées dans ce manuscrit. Elles évaluent, à différents niveaux, la sécurité d'un objet IdO. Notre première contribution porte sur les risques de fuite de données

utilisateurs lorsqu'une box domotique contrôle un ou plusieurs objets IdO. Malgré que la box domotique recoive l'intégralité du trafic réseau destiné aux objets IdO, nous proposons une méthode capable de décomposer la taille des données chiffrées d'une requête reçue par la box domotique en une liste d'actions utilisateur destinées aux objets IdO. Notre méthode permet à un attaquant localisé dans l'Internet et capable d'intercepter les communications vers la box domotique, d'inférer les actions utilisateur malgré la présence de la box domotique. Nous avons appliqué notre approche sur une box domotique connectée à 16 objets IdO, et réussi à inférer les actions utilisateur dans plus de 91.2% des cas.

Dans notre seconde contribution, nous évaluons la sécurité d'un objet IdO par rapport au contenu de son firmware, c'est-à-dire la composition des binaires présents dans ce dernier. Pour ce faire, nous proposons une analyse hybride de firmwares capable de combiner une analyse statique et dynamique. À l'aide de cette dernière, nous extrayons le nom et la version des binaires présents, ainsi que le nombre de binaires capables de déployer un service pouvant être exploité, comme TELNET, HTTP ou SSH. Ces informations sont ensuite utilisées pour détecter 1) le nombre de services qu'un objet peut déployer, 2) la présence de vulnérabilités connues, ou 3) l'utilisation de binaires obsolètes. Les résultats de notre analyse de 4 730 firmwares d'objets IdO ont permis de mettre en lumière certaines pratiques de développement, comme la surreprésentation d'un même binaire (et version) dans les objets d'une même marque (par exemple `lighttpd` et `dropbear`), ou la présence de vulnérabilités connues dans les binaires utilisés pour déployer les serveurs HTTP et SSH dans les firmwares publiés avant 2016. D'ailleurs, nous avons également remarqué que, dès 2017, des mesures semblent avoir été prises pour corriger les vulnérabilités et mettre à jour les versions des binaires.

À partir des données de firmwares extraites, nous élaborons dans une troisième contribution, une méthode d'identification active de propriétés (comme la version d'un binaire ou les mots de passe) d'un objet IdO à l'aide de plusieurs algorithmes d'apprentissage automatique, c'est-à-dire des classificateurs *Random Forest*. Ces derniers sont entrainés à l'aide 1) l'intégralité des données de firmwares, ou 2) avec les données spécifiques à une marque. Notre méthode repose sur l'hypothèse qu'il est possible de supposer les services déployés par un objet à partir des binaires présents dans son firmware. Par conséquent, nous pouvons présumer les résultats d'un scan de ports TCP/UDP. Notre solution a comme objectif d'être utilisée peu après la phase de reconnaisance où, à partir des services déployés par un objet, un attaquant souhaite identifier d'autres propriétés sur ledit objet à l'aide des classificateurs *Random Forest* entrainés à ces tâches. L'évaluation de notre méthode montre qu'il est possible pour un attaquant de prédire la marque d'un objet à partir de ses services dans 76.85% des cas. Des propriétés plus détaillées peuvent également être prédites comme le nom d'un utilisateur ou un mot de passe présents dans l'objet dans plus de 97.05% des cas, et en moyenne moins de deux tentatives.

Enfin, dans notre quatrième contribution, nous présentons notre infrastructure logicielle d'émulation d'objets IdO, et de déploiement en pot de miel. L'infrastructure logicielle introduite permet une correction itérative des erreurs d'émulation tout en considérant les contraintes de furtivité inhérentes à un pot de miel. Chaque type d'erreur d'émulation est corrigé par un composant dédié, et appelé *Enhancer*. Par exemple, les fichiers spéciaux (`/dev` et `/proc`) manquants sont créés dynamiquement, les interfaces réseau de l'objet émulé sont configurées, notre librairie partagée est compilée avec les fonctionnalités compatibles avec l'objet émulé comme le déchiffrement des paquets HTTPS ou la restriction de l'accès au serveur HTTP(S). Notre infrastructure logicielle a été implémentée afin de pouvoir intégrer facilement de nouveaux *Enhancers*.

Nous avons évalué les performances d'émulation de notre approche, et montré que cette dernière était capable d'émuler jusqu'à 825 (82.5%) objets IdO contre 454 (45.4%) pour la méthode de l'état de l'art. Cependant, seuls 443 (53.69%) objets émulés peuvent être déployés en pot de

miel avec une furtivité maximale. Nous montrons ensuite que nos pots de miel ont été capables d'exfiltrer plus de 900 virus informatiques (35 distincts) dont 26 (74.28%) étaient inconnus ou publiés récemment dans la base de données de virus informatiques VirusTotal [4].

Organisation

Ce manuscrit est découpé en quatre parties.

Partie 1 : État de l'art

Dans cette première partie, nous détaillons dans le chapitre 1 les solutions existantes relatives à l'analyse du trafic réseau d'un objet connecté. Dans un premier temps, nous présentons les méthodes passives, puis actives, permettant d'identifier le type des objets IdO connectés à un réseau. Dans un second temps, nous détaillons les solutions capables d'identifier les actions utilisateur.

Le chapitre 2 présente les solutions d'analyse statique et dynamique de firmwares d'objets IdO existantes dans la littérature. Enfin, dans le chapitre 3, nous détaillons les différents pots de miel proposés afin d'étudier les cyberattaques ciblant 1) les objets IdO, ou 2) les objets de l'Internet industriel (IIdO), selon le niveau d'interaction utilisé.

Partie 2 : Analyse du trafic réseau chiffré d'objets IdO

La deuxième partie présente notre méthode d'identification passive des actions utilisateur. Plus particulièrement, nous détaillons dans le chapitre 4, notre approche permettant d'inférer les actions utilisateur du trafic réseau chiffré d'une box domotique. Nous discutons premièrement de l'environnement domotique étudié, puis dans un second temps, nous détaillons les hypothèses ainsi que la méthode utilisée afin 1) d'analyser le trafic réseau de la box, et 2) d'extraire de ce dernier, les actions. Enfin, nous présentons les performances de notre méthode pour inférer les actions utilisateur à destination d'un à seize objets IdO associés à une box domotique.

Partie 3 : Analyse de firmwares d'objets IdO

La troisième partie regroupe nos deux contributions utilisant les données de firmwares d'objets IdO.

La première contribution est présentée dans le chapitre 5 et introduit notre analyse de firmwares d'objets IdO. L'objectif de ce chapitre est d'évaluer la sécurité des firmwares d'objets IdO selon trois axes 1) son niveau d'exposition, et 2) le niveau d'obsolescence et 3) le délai post vulnérabilité des binaires trouvés dans le firmware. Le premier axe inspecte le nombre de services qu'un firmware est capable de déployer. Ainsi, plus ce nombre est grand, plus l'objet déploie de services, et a potentiellement plus de chances d'être attaqué. Le second axe vérifie si les binaires présents dans un firmware sont récents par rapport à la dernière version des binaires disponibles au moment de la publication du firmware. Enfin, le troisième axe vérifie si les binaires présents souffrent ou non de vulnérabilités connues au moment de la publication du firmware.

Afin d'inspecter un firmware selon ces trois axes, notre analyse hybride de firmwares combine une analyse dynamique et une analyse statique. L'intérêt de l'analyse hybride est de limiter les erreurs lors de l'extraction d'information des données d'un firmware. Notre approche détecte les erreurs d'exécution lors d'une analyse dynamique et peut basculer automatiquement vers une

4. Disponible sur `https://www.virustotal.com/`, dernier accès le 12/05/2022

analyse statique. Nous avons ensuite appliqué notre analyse hybride sur des milliers de firmwares d'objets IdO publiés entre 2009 et 2019, et évalué la sécurité de ces derniers, en nous focalisant notamment sur les binaires utilisés pour déployer les services HTTP et SSH.

La contribution présentée dans le chapitre 6 propose une méthode d'identification active des propriétés d'un objet IdO, comme le nom ou la version des binaires utilisés pour déployer des services comme HTTP ou SSH. Notre méthode repose sur l'hypothèse que les services déployés par un objet IdO peuvent être dérivés du contenu de son firmware. Par conséquent, il est possible de supposer les résultats d'un scan de ports TCP/UDP. À partir de cette information initiale, un premier algorithme d'apprentissage automatique est entrainé de façon à prédire d'autres propriétés, comme la marque de l'objet IdO. Au fur et à mesure des prédictions, les nouvelles connaissances inférées servent à entraîner de nouveaux classificateurs afin de prédire des propriétés de plus en plus fines. D'ailleurs, nous évaluons justement les performances de nos classificateurs à prédire la marque d'un objet IdO à partir des résultats d'un scan de ports, puis à effectuer des prédictions plus complexes, comme les noms d'utilisateurs ou la version du binaire déployant HTTP.

Partie 4 : Étude des cyberattaques ciblant les objets IdO

La quatrième contribution est présentée dans le chapitre 7. Cette dernière introduit une infrastructure logicielle permettant d'émuler un objet IdO à partir de son firmware, puis de déployer cet objet émulé en tant que pot de miel à forte interaction. Notre contribution étend la méthode d'émulation proposée par FIRMADYNE [36], en intégrant des fonctionnalités permettant d'améliorer les performances globales d'émulation tout en considérant la furtivité du pot de miel. L'objectif étant de déployer rapidement un pot de miel à partir du firmware d'un objet IdO. Pour ce faire, l'infrastructure logicielle proposée permet une correction itérative des erreurs d'émulation, et est conçue de manière à intégrer facilement de nouvelles fonctionnalités. En complément, nous avons également modifié un noyau et un module Linux. Dans ce dernier, des fonctionnalités inhérentes à un pot de miel ont été définies et implémentées, comme l'interception et l'exfiltration des activités effectuées par les attaquants depuis le pot de miel. Les performances de notre approche sont ensuite évaluées selon 1) sa capacité à émuler des objets IdO accessibles depuis le réseau, et 2) aux attaques reçues lors de son déploiement en pot de miel.

Première partie

État de l'Art

Chapitre 1

Analyse du trafic réseau d'un objet connecté

Sommaire

1.1 Introduction

L'avènement de la domotique ou *maison intelligente* ont bouleversé notre quotidien. La possibilité de contrôler à distance des objets connectés a sensiblement amélioré notre confort de vie en laissant, par exemple, des capteurs gérer de manière automatique d'autres objets telle que l'ouverture ou la fermeture de volets selon l'exposition lumineuse. Selon un rapport de l'agence de la transition écologique (ADEME) datant de 2019 [5], les (multi)prises intelligentes qui, selon les demandes de l'utilisateur, autorisent ou non l'alimentation électrique des objets branchés à ces dernières, peuvent servir à mieux contrôler notre consommation électrique [28, 64], ainsi l'Internet des Objets (IdO) et ses objets associés (objets IdO) s'inscrivent parfaitement dans les démarches écologiques actuelles.

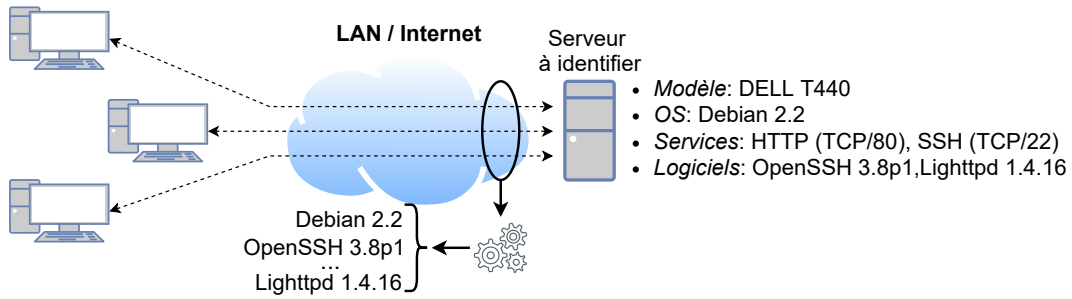
Laisser des objets connectés contrôler notre environnement personnel ou professionnel n'est pas sans risque. De part leur connectivité à un réseau informatique, ces objets peuvent être exposés sur l'Internet et par conséquent être compromis par un attaquant extérieur entraînant un risque de fuite de données (par exemple, voler du flux vidéo potentiellement compromettant [27, 163]) ou bien une utilisation malveillante de ces objets dans le cadre d'un botnet [13, 26, 91]. Cependant, pour mener à bien ce genre d'attaques sur un objet connecté, un attaquant doit d'abord identifier une ou plusieurs caractéristiques associées à l'objet comme sa marque, son modèle ou bien ses services déployés.

5. Disponible sur `https://ademe.fr/sites/default/files/assets/documents/infographie-economiser-eau-energie-2019.pdf`, dernier accès le 07/01/2022

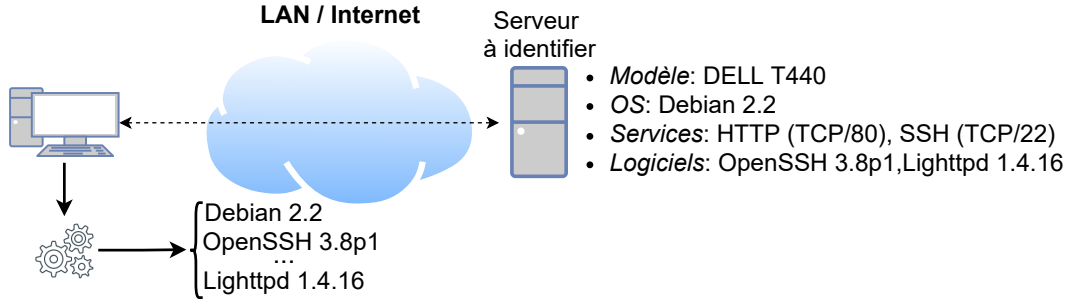
Dans ce chapitre, nous présentons les différentes techniques d'identification ou «fingerprint» d'un objet dans un réseau informatique. Nous discuterons dans un premier temps des techniques d'**identification passives** où aucune interaction avec l'objet étudié n'est autorisée puis nous présenterons les **identifications actives** qui, elles, supposent d'interagir avec lesdits objets. En complément, nous détaillerons comment les techniques d'identification passives peuvent être adaptées pour 1) inférer les actions effectuées par un objet IdO, ou 2) détecter les objets compromis dans un réseau informatique.

1.2 Identification d'un objet connecté

Une fois connecté à un réseau informatique, un objet IdO peut communiquer avec les autres objets présents localement sur le même réseau informatique ou bien avec un serveur connecté au Cloud et par extension, l'Internet. L'identification d'un objet connecté ou *fingerprinting* en anglais, repose sur l'utilisation d'une empreinte ou signature construite à partir de données calculées ou extraites à partir des paquets réseau transmis ou reçus par l'objet étudié. Cette pratique introduite vers les années 2000 servait initialement à identifier de manière fiable les utilisateurs de sites web [115, 61, 94] et n'est donc pas spécifiquement liée au monde de l'Internet des Objets. Néanmoins, compte tenu des risques de sécurité et des récentes attaques ciblant spécifiquement l'IdO [49, 75, 7, 82, 26, 91], l'usage du *fingerprinting* fut adapté à ce nouveau contexte qui, contrairement à l'Internet *traditionnel*, consiste en une interconnexion d'objets hétérogènes en termes de technologie de communication, de puissance de calcul ou bien de fonctionnalités [135, 26, 10].



(a) Méthode passive



(b) Méthode active

FIGURE 1.1 – Identification ou *fingerprinting* d'un serveur

Dans cette section, nous détaillons les travaux permettant d'identifier avec différents niveaux de précision, les propriétés d'un objet qui, en se basant sur l'exemple de la figure 1.1, sont :

— son type (serveur),

— son modèle (DELL T440),

— ses protocoles supportés (HTTP sur le port 80 et SSH sur le port 22),

— la version des logiciels utilisée (SSH : OpenSSH_3.8p1 et HTTP : Lighttpd 1.4.16),

— son fonctionnement normal, par exemple, seuls deux services sont continuellement actifs sur le serveur et ce dernier interagit uniquement avec 3 autres objets.

L'identification de ces propriétés est dépendante du point d'observation et de la technologie de communication utilisée par l'objet IdO. En effet, comme illustré dans les figures 1.1(a) et 1.1(b), deux méthodes sont principalement utilisées puis détaillées dans les prochaines sections.

1.2.1 Méthodes passives

Cette première méthode, illustrée par la figure 1.1(a), n'implique pas d'interaction avec l'objet à identifier. Elle est totalement furtive mais nécessite un accès ou une visibilité du trafic réseau dudit objet.

1.2.1.1 Détection d'activités suspectes ou malveillantes

Les attaques informatiques augmentant d'année en année [83, 123], notamment lors de grands événements comme la crise COVID selon un rapport de l'agence européenne chargée de la sécurité des réseaux et de l'information (ENISA) [6], ces dernières génèrent de nombreuses communications perceptibles sur les réseaux informatiques. L'observation passive des paquets échangés au sein d'un réseau informatique permet à des outils comme les systèmes de détection et prévention d'intrusions [108, 17, 142] (respectivement IDPS en anglais) de détecter les activités anormales ou suspectes ciblant un ou des objets connectés présents dans les réseaux informatiques surveillés.

Dans le cadre de la surveillance d'un objet connecté, la distinction de ses activités normales et malveillantes peut être effectuée selon [142] via :

— une comparaison entre la signature de son trafic réseau et des signatures d'attaques connues. Cette méthode requiert une mise à jour régulière des signatures d'attaques connues et, naturellement, ne peut pas détecter les nouvelles attaques ou *0-day* en anglais.

— Ainsi, pour pallier ce manque d'évolutivité, une signature peut être construite à partir du trafic réseau de l'objet lorsque ce dernier effectue des actions dites normales. Ensuite, cette dernière sera comparée aux signatures générées par le nouveau trafic réseau de l'objet et si elles diffèrent de manière trop importante alors on considérera qu'une attaque ou une compromission de l'objet aura été observée.

Compte tenu de l'hétérogénéité des fonctionnalités des objets IdO et du nombre croissant d'attaques les ciblant, construire des signatures liées à des attaques ou virus informatique est souvent long et difficile. C'est pourquoi, suivant les caractéristiques de l'objet IdO à surveiller [95], la détection d'anomalies peut être effectuée via :

— une méthode supervisée [122, 112] qui utilise des signatures d'attaques connues,

— une méthode non-supervisée [106, 153] qui ne connaît pas à l'avance les signatures de vulnérabilités mais va définir le comportement normal d'un objet à partir de son trafic réseau.

6. Disponible sur `https://www.enisa.europa.eu/news/enisa-news/enisa-threat-landscape-2020`, dernier accès le 07/01/2022

Enfin, dès qu'une activité suspecte est détectée, une alerte est généralement envoyée à l'administrateur et l'objet concerné voit son trafic réseau entrant ou sortant restreint par des règles de pare-feu ou *firewall* en anglais.

Dans les deux prochaines sections, nous détaillerons respectivement les systèmes de détection et prévention d'intrusions adaptés à l'Internet des Objets dans leurs approches supervisées puis non-supervisées.

1.2.1.2 Détection supervisée

Les objets IdO souffrent de nombreuses vulnérabilités [36, 90] ainsi, identifier ces derniers dans un réseau informatique pourrait être utilisé par des attaquants pour cibler spécifiquement ces objets. Par exemple, il est possible de distinguer des objets IdO des autres objets connectés à un réseau informatique universitaire, notamment en construisant des signatures à partir du temps des connexions, les tailles moyennes des paquets ou bien le nombre de requêtes DNS et NTP effectuées pendant une fenêtre de temps donnée [155, 154]. Parmi les 20 objets IdO connectés au réseau informatique universitaire étudié, les chercheurs identifient le modèle de chacun d'eux avec une précision supérieure à 95% à l'aide d'un algorithme de classification *Random Forest* entraîné à l'aide des signatures précédemment détaillées. Le type et le modèle de 10 objets IdO peuvent également être identifiés en analysant les interactions réseau effectuées par ces objets, et notamment leurs requêtes DNS [68]. L'identification, avec une précision moyenne de 93%, du modèle d'un objet IdO utilisant le protocole sans-fil ZigBee est possible à l'aide de signatures composées de la moyenne des tailles de données, le temps moyen inter-paquet ou bien l'écart-type des tailles de données dérivés à partir du flux de paquets de ces objets [8].

[122] s'intéresse au système d'éclairage intelligent Philips Hue et montre, qu'entraîner l'algorithme *Support Vector Machine* [30] (SVM), avec des signatures construites à partir de la taille des données dans les paquets et du temps inter-paquets, permet de détecter avec une précision supérieure à 94% des attaques de type violation d'accès non-autorisé comme celles présentées dans [54]. De la même manière, [112, 110, 111, 9, 124, 154] sont capables d'identifier les modèles des objets IdO connectés à un réseau informatique en utilisant une classification par approche 1) un-contre-tous [112, 110] ou 2) multi-classes [111, 9, 124, 154], entraînée à partir de signatures composées de valeurs calculées à partir du flux de paquets ou d'attributs présents dans les entêtes des couches réseau, transport ou application. Parmi ces attributs il y a la taille des données de la couche application, le numéro de port UDP/TCP ou bien la valeur de l'attribut Time-To-Live (TTL) de l'entête IP [111] ainsi, par exemple, 23 attributs sont utilisés dans [112] et [9]. D'un autre côté les signatures composées de valeurs calculées à partir du flux de paquets peuvent contenir la taille moyenne des paquets échangées, l'écart type des tailles des paquets [124], ou bien la taille des paquets transmis via un protocole comme SSDP ou NTP[154].

Les méthodes présentées précédemment étudient généralement un nombre faible d'objets IdO, de l'ordre de la trentaine, ce qui, compte tenu du grand nombre d'objets IdO proposés au public, n'est pas représentatif. Les auteurs d'IoT Inspector [76] ont regroupé et labellisé le trafic réseau de 54 094 objets IdO collecté auprès 5 404 utilisateurs. Ce travail a nécessité d'associer manuellement un trafic réseau à un modèle d'objet IdO, ce qui pose un problème d'évolutivité car chaque nouvel objet IdO proposé sur le marché devra à son tour être associé à son trafic réseau, avant de pouvoir être intégré dans IoT Inspector.

1.2.1.3 Détection non supervisée

Les auteurs de [119] détectent avec une précision de 95.60% les comportements anormaux d'objets IdO via une méthode d'apprentissage fédérée où chacun des 33 objets IdO étudiés voit son flux de paquets automatiquement analysé par un réseau de neurones récurrents à portes (*Gated Reccurent Unit (GRU)* en anglais).

[106] identifie le type de 33 objets IdO avec une précision supérieure à 98% grâce à une utilisation automatique de l'algorithme de clustering k-Nearest Neighbors (kNN) entraîné sur des signatures construites à partir de 33 attributs extraits ou calculés à partir d'une transformation de Fourier discrète (TFD) suivie d'une autocorrélation discrète effectuée sur le flux de chaque protocole réseau utilisé par un objet IdO. Parmi ces attributs, on pourra citer le nombre de flux, la moyenne des périodes ou bien le nombre de flux ayant une période comprise dans un intervalle donné. Les auteurs de [152, 153] entraînent de manière non-supervisée un algorithme de clustering K-means par objet IdO sur des signatures construites à partir, entre autres, de la taille moyenne des paquets, le nombre d'octets par seconde pour les protocoles DNS, NTP ou SSDP sur plusieurs fenêtres de temps données pour identifier le type d'objet IdO [152] ou des anomalies dans leurs communications [153].

[135, 31] présentent un IDS pour objets IdO utilisant 6LoWPAN afin de les protéger des attaques *sinkhole* consistant à rediriger le trafic du réseau vers un nœud appartenant à l'attaquant. En effet, le protocole de routage *Routing Protocol for Low power and Lossy networks* (RPL) s'effectuant nœud par nœud jusqu'à atteindre le destinataire, un nœud compromis peut forcer les autres nœuds à l'utiliser pour faire transiter les paquets échangés.

L'utilisation d'algorithmes de Deep Learning pour détecter des attaques sur des objets IdO a été étudiée par [164, 183, 109, 101] avec notamment des réseaux de neurones de type auto-encodeur. De plus, [109] montre qu'un auto-encodeur est plus performant que des algorithmes comme Isolation Forest, SVM ou que Local Outlier Factor (LOF) pour détecter des attaques de botnets [91, 13, 26] comme Mirai [13].

1.2.2 Méthodes actives

Illustré par la figure 1.1(b), cette approche étudie les réponses d'un objet connecté à des requêtes prédéfinies ce qui utilise la bande passante du réseau et donc la rend moins furtive que l'approche passive. Néanmoins, les objets IdO ne communiquant pas nécessairement de manière continue pendant un long laps de temps, il peut en effet être utile de forcer les communications avec ces objets via une méthode d'identification active.

[134] identifie avec une précision supérieure à 94% le modèle des objets connectés dont des objets IdO à l'aide d'un réseau de neurones artificiels entraîné à partir de signatures composées des temps séparant l'envoie et la réponse d'une requête ping. Les auteurs de [33] proposent également une méthode d'identification du modèle des objets sans-fil Wi-Fi (802.11) via une analyse des réponses à des requêtes de type *Probe Request* ou *Authentication Request* dans lesquelles certains attributs ont des valeurs non standard.

L'un des avantages de l'approche active est qu'il est possible d'identifier les services déployés, y compris ceux ne générant pas de trafic réseau, ou bien le système d'exploitation d'objets connectés à un réseau local ou à l'Internet. Parmi ces approches, nous pouvons citer Nmap [121] qui permet d'identifier pour toute machine connectée, ses services actifs ainsi que le nom et la version des logiciels utilisés à l'aide de requêtes UDP ou TCP, en analysant les bannières de certains services (en anglais *banner grabbing*). Cette méthode est également utilisée pour identifier son système d'exploitation via l'envoi de paquets TCP et ICMP puis une analyse

des attributs présents dans leurs réponses. D'autres outils d'identification active [16, 145, 144] utilisent uniquement ICMP [16] ou une seule requête TCP *SYN* suivie d'une analyse des temps de réponses des paquets TCP *SYN-ACK* envoyés par la cible [145, 144] pour caractériser le système d'exploitation.

Des analyses à l'échelle de l'Internet dans [63, 182] montrent qu'il est possible d'identifier la marque ou le modèle d'objets IdO connectés à l'Internet en utilisant du traitement automatique des langues [63] ou un réseau de neurones [182] faisant correspondre des mots clés présents dans la couche application [63, 182] et certains attributs [182] des entêtes réseau et transport, comme le Time-To-Live ou le numéro de port que l'on aura extrait préalablement des paquets réponses de ces objets connectés, à des modèles d'objets IdO existants.

En complément, des moteurs de recherche comme Shodan [7] ou Censys [59] scannent l'Internet de manière active et continue afin de cartographier les objets avec les services déployés qui y sont connectés. Dans ces deux approches, les informations des services sont inférées via la technique du *banner grabbing* qui consiste à extraire des mots clés prédéfinis présents dans les entêtes de protocoles comme HTTP ou SSH pour déduire le logiciel et/ou sa version utilisée.

Dans [88], l'algorithme ZMap [60] initialement utilisé par Censys [59] est amélioré de 129% permettant ainsi de scanner plus rapidement les objets de l'Internet et ce, de manière plus précise vis-à-vis des informations sur les services déployés. Leur approche détecte mieux certains services comme SSH, Modbus ou Siemens S7. Les auteurs de [88] présentent également une méthode d'identification du système d'exploitation (OS) basée sur le classificateur k*[12] (*KStar* en anglais) entraîné à partir de signatures composées : d'attributs des entêtes réseau et transport comme le TTL, la taille maximale de segment (MSS) ou la taille de fenêtre TCP extraits durant leur analyse. Leur méthode présente des performances moyennes (51% de précision), cependant certains OS comme FreeBSD ou Gentoo sont identifiés avec un rappel supérieur à 65%.

1.3 Identification d'actions utilisateur

En complément des risques de sécurité [91, 13, 26], les objets IdO souffrent également de problèmes de fuite de données utilisateur, inférées à partir du trafic réseau généré par ces objets IdO.

Comme présenté dans la figure 1.2, un attaquant peut effectuer une écoute passive des communications d'un réseau informatique de deux manières. La première (1) suppose que l'attaquant se trouve à proximité ou est déjà connecté au réseau informatique ciblé ainsi, ce dernier pourra intercepter le trafic réseau sans-fil mais également les paquets réseau de chacun des objets connectés. Le second point d'observation (2) permet à un attaquant localisé dans l'Internet d'intercepter uniquement le trafic réseau entrant et sortant du réseau informatique privé. Ce point d'observation ne permet pas d'obtenir les communications sans-fil des objets présents dans le réseau ni de connaître précisément quel objet envoie ou reçoit des paquets, car le routeur connectant le réseau privé à l'Internet va masquer cette information.

Dans [52] les auteurs montrent qu'un attaquant capable d'intercepter les communications d'appareils de fitness sans-fil utilisant Bluetooth Low Energy (BLE) peut identifier les activités effectuées ainsi que l'identité des utilisateurs à l'aide d'un arbre de décision entraîné par des signatures dérivées de propriétés liées aux flux de paquets BLE comme le débit des données, le nombre de paquets sans données ou initiant une connexion BLE. De même, en utilisant les données d'accéléromètres et gyroscopes obtenables par des montres connectées, [180] déduit 37 mouvements de doigts, mains et bras des utilisateurs avec une précision de 98%, alors que [171]

7. Disponible sur `https://www.shodan.io/`, dernier accès le 07/01/2022

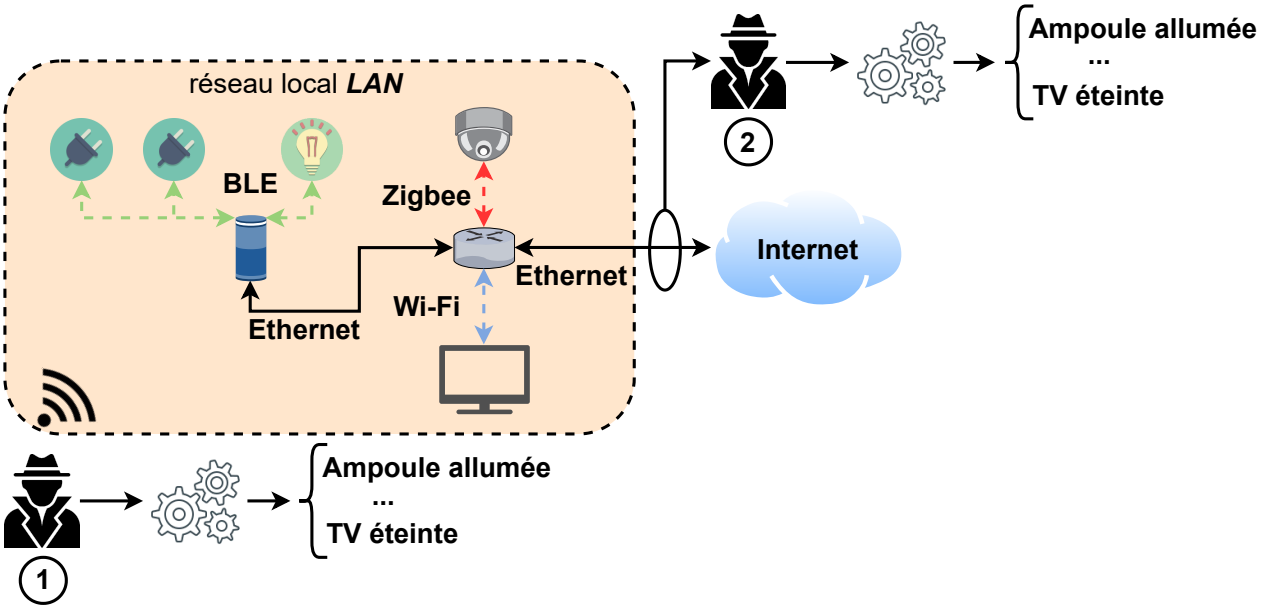


FIGURE 1.2 – Exemple d'écoute passive des communications sans-fil d'un réseau informatique effectuée par un attaquant localisé à (1) proximité ou (2) en dehors du réseau informatique ciblé

présente MoLe, une méthode inférant les mots tapés au clavier afin de réduire l'espace de recherche d'une future attaque par force brute (*brute-force* en anglais). Leur approche retourne notamment dans 50% des cas 24 mots possibles, et dans 30% des cas seulement 5.

Les auteurs de [41] étudient un réseau domestique comportant un thermostat ainsi qu'un détecteur de fumée connectés en Wi-Fi et montre que les actions de ces objets comme la détection de fumée ou un ajustement du thermostat sont identifiables dans le trafic réseau par la fréquence des paquets NTP ou les tailles de donnée constantes des paquets SSL. Parmi les actions évaluées, les cinq activations du détecteur de fumée sont détectées à 100% alors que l'identification des deux réglages possibles du thermostat sont identifiés avec une précision de 67% et 88% suivant le réglage demandé.

Une démonstration de faisabilité est présentée dans [14] où quatre objets IdO, dont 1 caméra et 1 prise connecté, sont identifiés à l'aide de leurs requêtes DNS respectives puis, les actions utilisateur effectuées sur chacun d'eux sont identifiées à partir du débit de paquets échangés entre l'objet IdO et le serveur de son revendeur localisé dans le cloud. Par exemple, la caméra transmet plus de données lorsqu'une personne se situe à proximité, et l'activation ou désactivation de la prise connectée génère des pics de débit à environ 70 000 octets/seconde, soit 28 fois supérieur au débit lorsqu'aucune action n'est effectuée. Cependant les auteurs ne mesurent pas les performances de leurs observations.

L'utilisation d'un VPN permet de masquer le trafic réseau individuel de chacun des objets IdO présents dans un réseau informatique, et ainsi éviter que des attaquants identifient le modèle des objets IdO ou les actions utilisateur effectuées sur ces derniers [15].

Similairement, [165] étudie un environnement de type *maison intelligente* avec 19 objets IdO, soit environ trois fois plus d'objets IdO que [14, 15]. Ces travaux montrent également que les actions utilisateur peuvent être inférées en analysant les flux de paquets échangés entre chaque objet IdO et le serveur de son revendeur, par exemple, les tailles des données échangées ainsi que l'ordre dans lequel elles sont transmises, ou bien la durée moyenne d'une session TCP.

Leur approche identifie chaque action utilisateur avec un rappel moyen supérieur à 97% qui, dans leur expérimentation, correspond à identifier correctement 97 actions sur les 100 actions réellement effectuées. De plus, pour les ampoules connectées de Tp-Link ou une serrure connectée de Kwikset, leur méthode détecte à 100% l'action utilisateur effectuée. Cependant, leur approche requiert d'exécuter entre 50 et 100 fois chaque action disponible sur un objet IdO pour générer sa signature correspondante, ce qui peut poser des problèmes de mise à l'échelle.

Les actions utilisateur effectuées sur 22 objets IdO sont identifiées à partir de signatures composées de 197 attributs dérivés des données temporelles du flux de paquets associé aux actions, comme la durée moyenne et médiane d'un flux ou son entropie. Enfin, un algorithme de classification *Random Forest* est entraîné sur ces dernières et permet d'identifier avec une précision de 96% les actions effectuées [8]. Par exemple, les actions *ouvrir* et *fermer* d'une serrure connectée de marque August Home sont correctement identifiées avec une précision respective de 67% et 100% ; les déclenchements des capteurs d'ouverture de portes ou de mouvements de marque DLink sont parfaitement détectés. Ainsi, en utilisant les données temporelles, leur approche s'adapte aux objets IdO qui chiffrent leur trafic réseau.

1.4 Synthèse

Depuis 2008, de nombreuses solutions ont été proposées pour identifier les propriétés listées dans le tableau 1.1 en analysant le trafic réseau d'un objet connecté de manière passive ou active comme illustré dans la figure 1.1. Les approches présentées se focalisent sur les objets de l'Internet des Objets. Néanmoins, certaines d'entre elles, notamment les approches actives [121, 59], fonctionnent également pour tout objet connecté suivant le modèle TCP/IP.

Cible de l'identification	**Approche**		
	Passive		***Active***
	Supervisé	*Non-supervisé*	
Type	[155, 110, 68, 154, 8]	∅	[63, 182]
Modèle	[155, 110, 111, 112, 14, 68, 9, 154]	∅	[121, 33, 134, 63, 182]
Service	∅	∅	[121, 59] et Shodan [7]
Logiciel	∅	∅	[121, 16, 145, 59, 88, 144] et Shodan [7]
Attaque	[122, 112, 111, 124]	[135, 164, 183, 31, 109, 101, 119, 106, 152, 153]	∅
Action utilisateur	[52, 180, 171, 41, 14, 165, 8]	∅	∅

Tableau 1.1 – Références triées selon l'information à identifier et le type d'approche

Initialement, ces études utilisaient de simples statistiques sur les tailles ou le nombre de paquets puis, se sont orientées vers une analyse des flux de paquets via notamment l'utilisation

de méthodes d'apprentissage automatique (ou *machine learning* en anglais). Cette évolution dans les attributs analysés peut s'expliquer par plusieurs facteurs notamment le développement d'algorithmes d'intelligence artificielle plus performants (par exemple, les réseaux de neurones récurrents) ou l'évolution des objets IdO qui proposent plus de fonctionnalités et ainsi génèrent un trafic réseau plus dense et moins facilement interprétable notamment par l'utilisation du chiffrement.

Les études portant sur la détection d'actions utilisateur montrent que l'usage du chiffrement ne protège pas nécessairement la vie privée des utilisateurs. En effet, les objets IdO étant limités en puissance de calcul et autonomie, ils communiquent principalement lorsqu'une tâche leur est donnée et ainsi, génèrent un flux de paquets souvent spécifique à cette dernière. Ainsi, des statistiques sur le nombre de paquets ou d'octets de données échangés entre un objet IdO et son serveur, localisé dans le cloud, sont souvent utilisées pour construire les signatures liées à chaque tâche. L'avantage de ces analyses basées sur les flux de paquets est qu'elles fonctionnent aussi bien en se plaçant la sonde dans l'Internet, ou dans le réseau local où se trouvent les objets IdO.

Ces études se sont intéressées principalement aux objets IdO seuls c'est-à-dire des objets IdO avec lesquels un utilisateur peut directement interagir or, il existe des box domotiques (*IoT gateway* ou *IoT hub* en anglais) permettant de gérer plusieurs objets IdO directement. Pour ce faire, ces dernières réceptionnent les actions demandées par un utilisateur depuis un ou plusieurs serveurs localisés dans le cloud puis transmettent les actions aux bons objets IdO. Par conséquent, un attaquant localisé dans le même réseau informatique que les objets IdO ou dans l'Internet ne sera plus capable de déduire précisément les actions utilisateur via les méthodes présentées dans ce chapitre.

Ainsi, notre première contribution [191] s'intéresse aux risques de fuite de données utilisateur qu'un attaquant situé dans l'Internet pourrait inférer à partir du trafic réseau entrant et sortant d'une box domotique gérant plusieurs objets IdO tels que des ampoules ou des prises connectées. Les méthodes passives présentées dans ce chapitre centrées sur des interactions simples et unitaires ne sont pas utilisables, car le trafic réseau de la box domotique peut être chiffré et donc ne permet pas de déterminer le ou les objets IdO destinataires des actions utilisateur. De ce fait, notre approche intercepte et décompose la taille des requêtes reçues par la box domotique en une liste de combinaisons d'actions utilisateur possibles qui peuvent, ensuite, être utilisées pour déterminer le modèle des objets IdO.

Dans une seconde contribution [188], nous présentons une approche active d'identification des logiciels utilisés pour déployer les services d'un objet IdO connecté à un réseau informatique LAN ou à l'Internet. Cette contribution se différencie des approches actives existantes par sa capacité à déduire les logiciels et leurs versions uniquement à partir de la liste des services déployés par un objet IdO. Autrement dit, notre approche nécessite uniquement l'information qu'un service existe sur un numéro de port TCP/UDP donné au lieu, par exemple, d'analyser le contenu des bannières retournées par les services déployés. Notre contribution est originale car elle est dérivée d'une analyse à grande échelle de firmwares d'objets IdO au lieu d'analyser directement des objets IdO physiques. En effet, à partir de firmwares d'objets IdO, il est possible de savoir quels sont les services déployés par un objet ainsi que le nom et la version des logiciels utilisés. Ainsi, nous utilisons les données extraites de l'analyse de firmwares pour déduire, à l'aide d'un algorithme d'apprentissage automatique, les noms et versions des logiciels à partir de la liste des services actifs.

Chapitre 2

Analyse de logiciel interne

Sommaire

2.1 Introduction

Dans le précédent chapitre, nous avons vu qu'il était souvent possible de déterminer, à partir du trafic réseau, les objets de l'Internet des Objets (IdO) connectés à un réseau informatique, ou bien si ces derniers avaient été compromis par une attaque informatique. Néanmoins, l'analyse du trafic réseau ne permet pas de déterminer si un objet IdO est vulnérable à une ou plusieurs attaques connues ou inconnues (*0-day* en anglais). Avoir accès au logiciel interne ou *firmware* en anglais d'un objet IdO permet de vérifier de manière plus précise son fonctionnement et ainsi ses potentielles vulnérabilités sachant que, d'après un rapport d'Hewlett-Packard (HP) [8], un objet IdO souffre en moyenne de 25 vulnérabilités.

Selon la définition donnée dans un rapport de l'ANSSI[9], un firmware est un ensemble de logiciels présents dans un objet connecté en «sortie d'usine». Ces logiciels dépendent des fonctionnalités proposées par l'objet mais également du système d'exploitation dans lequel ils sont exécutés et qui, dans le cas de l'Internet des Objets, est principalement Linux [36, 173]. Comme illustré par la figure 2.1, un firmware d'un objet IdO se trouve généralement sous la forme d'une structure hiérarchique de dossiers ce qui permet de mesurer la sécurité de ce dernier en analysant :

1. le contenu des fichiers de configuration, comme par exemple `/etc/shadow` stockant les identifiants et mots de passe ou alors `/etc/ssh/sshd_config` servant à configurer le serveur SSH, et
2. le fonctionnement des exécutables (ou binaires) comme `lighttpd` ou `dropbear` permettant de déployer respectivement une interface web et un serveur SSH.

Vérifier qu'un exécutable ne souffre pas de vulnérabilités peut être évalué :

8. Disponible sur `https://www.hp.com/us-en/hp-news/press-release.html?id=1744676`, dernier accès le 07/01/2022
9. Disponible sur `https://www.ssi.gouv.fr/uploads/2019/11/anssi-guide-hardware_security_requirements.pdf`, dernier accès le 07/01/2022

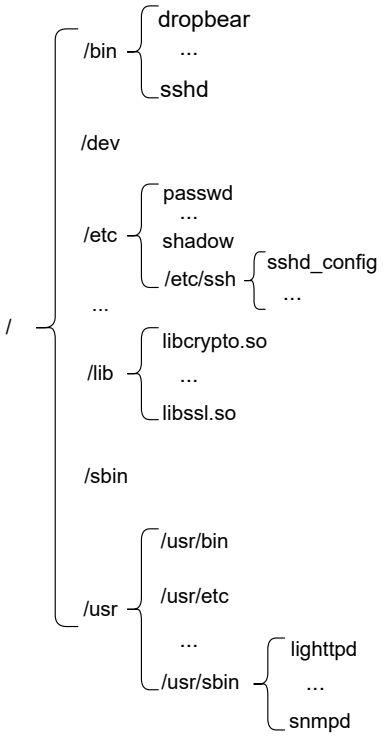


FIGURE 2.1 – Arborescence de dossiers sur Linux

— sans l'exécuter via une analyse des instructions incluses dans son code machine, ou si possible, son code source.

— en exécutant le binaire, son fonctionnement en situation «pseudo réelle» peut être analysé. Des informations hardware comme le contenu des registres CPU, l'état de la pile ou de la mémoire peuvent également être obtenues.

Deux principales approches peuvent être utilisées pour analyser un firmware, l'approche statique où les binaires ne sont pas exécutés et l'approche dynamique qui, elle, va analyser les binaires en les exécutant. Nous détaillerons ces techniques dans les deux prochaines sections.

2.2 Analyse statique

Le déploiement des services ou le système d'exploitation peuvent être paramétrés à l'aide de fichiers de configuration. Par exemple, les fichiers `/etc/passwd` et `/etc/shadow` sont utilisés pour stocker les identifiants et mots de passe des utilisateurs du système, et le fichier `/etc/ssh/sshd_config` sert à configurer le serveur SSH.

Les auteurs de [48] ont analysé 373 firmwares et remarqué que les librairies `zlib` et `openSSL`, utilisées respectivement pour la compression et le chiffrement de données souffraient de vulnérabilités connues comme celles décrites dans les CVE-{2006-4339, 2009-3245}.

Dans [44], les auteurs effectuent une analyse de 32 356 firmwares téléchargés à partir des sites internet de fabricants d'objets connectés comme Belkin, D-Link, Samsung etc. Parmi les mauvaises pratiques de développement impliquant des risques de sécurité, les auteurs ont noté :

— la présence de certificats SSL auto-signés ainsi que des clés de chiffrement privées. Des études antérieures ont également noté la présence de ces deux éléments [80, 81, 70]. De

plus, les auteurs confirment que ces certificats étaient utilisés par environ 35 000 objets IdO connectés à l'Internet.

— l'utilisation de nombreux mots de passe par défaut identiques dans les firmwares. Ainsi, les auteurs ont listé les mots de passe les plus utilisés comme `<empty>`, `pass`, `logout` et `helpme`. Pour établir cette liste, les mots de passe stockés dans les fichiers `/etc/passwd` et `/etc/shadow` présents dans les firmwares ont été répertoriés puis une tentative de déchiffrement de la valeur chiffrée de ces derniers a été effectuée à l'aide du dictionnaire de mots de passe John The Ripper [126].

— des fichiers de configuration associés aux serveurs web, comme `lighttpd.conf` avec `lighttpd`[10] ou `boa.conf` avec `boa`[11], contenant des options vulnérables. Les auteurs ont remarqué que 81% de ces serveurs web étaient démarrés avec le privilège de type `root`, ce qui peut compromettre l'intégralité de l'objet connecté en cas d'attaque.

[141] s'est intéressé aux risques de sécurité dans l'IdO industriel et notamment les compteurs connectés de marque Schneider. Ce dernier a montré qu'en analysant manuellement le code du binaire responsable de l'accès à distance, il est possible de trouver l'identifiant et le mot de passe d'un utilisateur non répertorié. Or, définir un utilisateur non répertorié s'apparente à un mécanisme de type porte dérobée (ou *backdoor* en anglais) où il est possible pour une tierce personne ayant connaissance de cette information d'accéder à des ressources ou fonctionnalités à l'insu de l'utilisateur principal. Par exemple, la CVE-2018-6213 présente une vulnérabilité dans un routeur de marque D-Link où un utilisateur non répertorié ayant les droits d'administrateur est accessible depuis le serveur web. De manière générale, une *backdoor* peut être utilisée par un attaquant pour accéder à un objet IdO puis compromettre ce dernier ou l'utiliser comme relai pour attaquer une autre machine connectée au même réseau informatique.

Compte tenu du grand nombre de firmwares disponibles en téléchargement [44], il est nécessaire d'automatiser la détection de *backdoors* ou de failles de sécurité dans les binaires inclus dans les firmwares.

Ainsi, les approches [53, 148] basées sur l'exécution symbolique [32, 20] consistent à explorer tous les chemins d'exécution d'un programme en associant aux variables rencontrées des valeurs *symboliques* (formules logiques) qui seront ensuite interprétées afin de déterminer les valeurs concrètes satisfaisant ces conditions. À partir du code source d'un binaire et de l'outil d'exécution symbolique KLEE [35], les auteurs de [53] ont détecté plusieurs dizaines d'erreurs de type violation d'accès mémoire dans des firmwares. Cependant, le code source des binaires n'est pas nécessairement disponible. Ainsi, [148] proposent de désassembler le code des binaires présents dans un firmware d'objet IdO puis d'isoler les portions d'instructions «critiques», comme les comparaisons de chaînes de caractères, pour ensuite y appliquer une exécution symbolique. Leur approche est ensuite évaluée sur trois firmwares contenant des *backdoors* connues, et détecte effectivement chacune d'entre elles. Un exemple d'utilisation de comparaisons de chaînes de caractères pour détecter des utilisateurs *backdoor* est présentée dans [71], où les deux comparaisons de chaînes de caractères `strcmp(R0, "3sadmin")` et `strcmp(R0, "27988303")` présentes dans le binaire d'une caméra connectée permettent d'identifier un utilisateur non répertorié ayant comme identifiant `3sadmin` et mot de passe `27988303`.

Des analyses de teinte [46, 84, 50, 181] (ou *taint analysis* en anglais) peuvent également être utilisées pour étudier la propagation de données au cours de l'exécution d'un programme et ainsi détecter des vulnérabilités liées aux entrées utilisateur non vérifiées[12] comme les attaques :

10. Disponible sur `https://www.lighttpd.net/`, dernier accès le 07/03/2022
11. Disponible sur `http://www.boa.org/`, dernier accès le 07/03/2022
12. Disponible sur `https://cwe.mitre.org/data/definitions/20.html`, dernier accès 01/12/2021

— *Shellshock*[13] affectant le binaire *bash* de GNU et permettant à un attaquant de forcer *bash* à interpréter une chaîne de caractères comme une fonction, et donc exécuter des commandes shell, c'est-à-dire des commandes comme `cat`, `ls` ou bien des binaires quelconques présents sur la machine attaquée.

— *The Heartbleed Bug* [5] affectant les serveurs utilisant la librairie de chiffrement `OpenSSL`. Cette dernière permet à un attaquant de lire une partie de la mémoire de la machine ciblée à l'aide d'une requête TLS spécifique. Ainsi, un attaquant pourra, potentiellement, obtenir des données sensibles, comme des mots de passe ou identifiants.

Ce type d'analyse a ensuite été appliqué sur les binaires de firmware d'objets IdO comme dans [37] où les auteurs présentent DTaint, un outil capable de désassembler le code d'un binaire pour y générer un graphe de flot de contrôle où les variables et registres de processeur sont simulés par des valeurs symboliques. Cet outil a ensuite été testé sur les firmwares de six objets IdO et des vulnérabilités de type dépassement de tampon (*buffer overflow* en anglais) ou injection de commandes (*command injection* en anglais) ont été détectées. Avec une méthode d'analyse similaire, les auteurs de [136] se sont intéressés aux flux de données échangées par les différents binaires présents dans des firmwares d'objet IdO et ont trouvé 46 vulnérabilités *0-day* de type *buffer overflow* ou déni de service (*Denial of Service attack (DOS)* en anglais).

2.3 Analyse dynamique

Contrairement à la précédente approche, l'analyse dynamique repose sur l'exécution des binaires ou l'émulation des firmwares. Ainsi, des tests à données aléatoires [113] (*fuzzing* en anglais) peuvent être effectués sur les binaires en exécution pour détecter des vulnérabilités. Pour ce faire, le *fuzzing* consiste à analyser le comportement d'un binaire à des entrées utilisateur ne faisant pas partie de l'espace des entrées utilisateur attendu par le binaire et ce, pendant un laps de temps donné ou jusqu'à avoir testé toutes les entrées utilisateur possibles [97, 105].

D'après l'analyse de [36], le service web est l'un des services les plus répandus dans les objets IdO. Cependant, ce dernier est souvent vulnérable à de nombreuses attaques [29, 43, 123] liées à des entrées utilisateur non vérifiées résultant à des accès illégitimes de fichiers ou bien des injections de codes malveillants de type *Cross-Site Scripting (XSS)* ou *Cross-Site Request Forgery (CSRF)*. Ces deux injections de codes malveillants permettent à un attaquant de faire exécuter de manière volontaire ou non un code en `JavaScript` à un ou plusieurs utilisateurs d'un site web afin de récupérer des informations comme des mots de passe ou des identifiants de session.

C'est pourquoi, dans la littérature [77, 21, 62, 22, 50], de nombreuses solutions ont été proposées pour détecter ce genre de vulnérabilités. Néanmoins, les applications web étudiées n'étaient pas nécessairement utilisées dans les objets IdO.

Dans [45], les auteurs effectuent une analyse dynamique de 246 applications web présentes dans des firmwares d'objets IdO et ont trouvé, à l'aide d'outils de test de vulnérabilités open source comme ZAP [3] ou w3af [4], qu'environ un quart des applications souffraient au moins d'une vulnérabilité de type injection de commandes shell, XSS ou CSRF. De même, ces vulnérabilités ont également été détectées par l'analyse statique des codes PHP via l'outil RIPS [50]. Leur méthode d'exécution de binaires souffre de plusieurs limitations, notamment liées aux interactions matérielles (ou *hardware* en anglais). Par exemple, certains binaires requièrent la présence

13. Disponible sur `https://www.cert.ssi.gouv.fr/alerte/CERTFR-2014-ALE-006/`, dernier accès le 01/12/2021

d'interfaces réseau comme `eth1`, `br0`, alors que d'autres ont besoin d'interagir avec la mémoire via les fichiers `/dev/nvram` ou `/dev/mtdblock0`.

Afin de prendre en compte ces contraintes, des études basées sur une émulation partielle [85, 184, 92, 116] ou intégrale [36, 90] des firmwares ont été proposées et sont présentées ci-après.

Initialement présenté en 2014 [184] puis amélioré quatre ans plus tard [116], AVATAR est une infrastructure logicielle (*framework* en anglais) d'analyse dynamique de binaires combinant plusieurs outils comme, entre autres, 1) angr [149] pour l'analyse de teinte ou une exécution symbolique, 2) PANDA [57] pour enregistrer ou rejouer les instructions effectuées par un binaire exécuté par leur version modifiée de l'émulateur QEMU [24], 3) *The GNU Debugger (GDB)*[14] pour mieux comprendre les erreurs durant l'exécution d'un binaire et faire du débogage. AVATAR permet d'alterner l'exécution de code entre un émulateur, PANDA ou QEMU, et une plateforme matérielle, ce qui est particulièrement utile lorsque des firmwares ou binaires nécessitent la présence de composants matériels qui ne peuvent être correctement émulés. Dans [92], les auteurs introduisent un *framework* d'analyse similaire à la première version d'AVATAR optimisant notamment la vitesse du flux de données échangées entre l'émulateur et la plateforme matérielle.

Également proposé en 2014, PROSPECT [85] permet aussi d'alterner l'exécution d'un binaire entre un environnement entièrement virtualisé par QEMU et un objet physique, en interceptant les appels de 28 fonctions, comme `read` et `write`, utilisées pour interagir avec des composants matériels. En effet, un binaire lisant directement la mémoire principale d'une machine, va appeler les fonctions `open` puis `read` sur le fichier `/dev/mem`. Enfin, les auteurs ont évalué leur approche sur une alarme incendie connectée et détecté une vulnérabilité *0-day*.

Ré-utilisant l'idée d'intercepter certaines fonctions, FIRMADYNE [36] est un *framework* permettant d'émuler entièrement (ou *full-system mode* en anglais) un objet IdO à partir de son firmware et de QEMU[24]. Ce dernier requiert pour fonctionner un noyau Linux, les auteurs de FIRMADYNE utilisent un noyau Linux version 2.6.32.68, en incluant un driver capable d'intercepter certains appels systèmes (ou *syscalls* en anglais), de créer de faux fichiers de configuration système pour simuler la présence de certains composants matériels, comme `/dev/nvram` et `/dev/gpio` pour, respectivement, la mémoire RAM non volatile (*Non-volatile random-access memory (NVRAM)* en anglais) et les ports d'entrées-sorties de microcontrôleurs (*General Purpose Input/Output (GPIO)*). De plus, étant donné que la NVRAM peut être utilisée par un objet IdO pour récupérer des paramètres de configuration, comme une adresse IP ou le nom de l'interface réseau WAN, un ensemble de valeurs par défaut est défini pour simuler le contenu de cette dernière.

Sur les plus de 8 000 firmwares testés, environ 2 000 ont été correctement émulés et connectés au réseau. Ensuite, des attaques connues ont été appliquées sur ces derniers via l'outil de test de vulnérabilités Metasploit[100] et, au minimum, une vulnérabilité a été détectée dans 887 firmwares (45%). Afin d'augmenter le nombre d'objets émulés correctement connectés au réseau, [90] améliorent l'algorithme de configuration des interfaces réseau via un script shell exécuté automatiquement par l'objet émulé pendant son démarrage. Ainsi, leur approche est capable d'émuler 892 (79.36%) firmwares contre seulement 183 (16.28%) pour FIRMADYNE, soit environ 5 fois plus, et a notamment permis de détecter 12 vulnérabilités *0-day* à l'aide de l'outil RouterSploit [15].

Dans FirmFuzz [158], les objets IdO sont émulés d'une manière similaire à FIRMADYNE mais la sécurité des interfaces web est évaluée par du *fuzzing* afin de détecter, entre autres, des injections de commandes (IC), des dépassements de tampon (DT) ou des XSS. Pour ce faire, les auteurs ont prédéfini une liste d'entrées utilisateur pouvant générer une de ces vulnérabilités. Par

14. Disponible sur `https://www.sourceware.org/gdb/`, dernier accès le 08/03/2022
15. Disponible sur `https://github.com/threat9/routersploit`, dernier accès le 08/03/2022

exemple `n/bin/ls -al\n` pour faire une IC, `\";alert('XSS');//` pour effectuer une XSS, ou bien des chaînes de caractères dont la taille peut varier jusqu'à plusieurs centaines de caractères pour tester un DT. Enfin, sur un total de 6 427 firmwares téléchargés, leur approche a évalué la sécurité de seulement 32 (0.49%) firmwares et trouvé sept vulnérabilités *0-day*.

Faire du *fuzzing* sur des binaires exécutés depuis des firmwares émulés en *full-system mode* par QEMU est lent [117], notamment à cause de la gestion de la mémoire dans ce mode. C'est pourquoi [185, 89] proposent à l'instar d'AVATAR [184, 116] ou PROSPECT [85], d'alterner les instructions, mais entre :

— un binaire exécuté avec le «mode utilisateur» (ou *user-mode* en anglais) de QEMU qui, dans ce mode, gère plus rapidement les opérations liées aux accès mémoire. Par exemple, un appel à la commande `uptime`, indiquant le temps depuis lequel la machine est active, dure en moyenne 0.89ms en *user-mode* au lieu de 7.48ms en *full-system mode*.

— Un environnement émulé en *full-system mode* par la méthode de FIRMADYNE [36] qui est chargé d'exécuter les instructions interagissant avec des composants matériels, comme des interfaces réseau.

Ces deux méthodes améliorent la vitesse des tests à données aléatoires et sont capables de détecter plus rapidement les vulnérabilités. Par exemple, [185] trouve le dépassement de tampon associé à la CVE-2016-1558 en 2 heures et 32 minutes au lieu de 16 heures et 24 minutes en utilisant uniquement un environnement émulé en *full-system mode*. En moyenne, [185] est 10 fois plus rapide qu'une approche en *full-system mode*. D'un autre côté [89] trouve cette même vulnérabilité en 7 minutes en utilisant notamment DECAF [73] au lieu de QEMU.

2.4 Synthèse

Depuis 2010, les analyses de firmwares d'objets IdO se sont perfectionnées et complexifiées. Les analyses statiques se sont raréfiées au profit d'analyses dynamiques avec, plus particulièrement, la mise au point d'infrastructures logicielles pour des analyses dynamiques où plusieurs tests, par exemple une exécution symbolique, une analyse de teinte ou du *fuzzing*, peuvent y être effectués afin de trouver des vulnérabilités plus complexes et indétectables par une analyse statique.

Que ce soit des analyses par approche statique ou dynamique, la plupart des études citées dans le tableau 2.1 ont reporté l'existence de vulnérabilités de type injection de commandes, XSS ou *buffer overflow*, dans les firmwares d'objets IdO. Naturellement, chaque vulnérabilité détectée a été reportée au fabricant de(s) objet(s) IdO concerné(s) puis une fiche explicative liée à cette vulnérabilité est publiée dans le répertoire public de failles de sécurité informatique *Common Vulnerabilities and Exposures (CVE)* [16].

Les études détaillées précédemment s'attardent principalement sur la détection de nouvelles vulnérabilités. Or, d'après la fondation *Open Web Application Security Project (OWASP)*, l'utilisation de logiciels trop anciens (*deprecated* en anglais) et/ou comportant des vulnérabilités déjà connues fait partie des risques de sécurité les plus importants dans l'Internet des Objets [17].

C'est pourquoi, nos travaux [189, 188] visent à mesurer les risques de sécurité dans les firmwares d'objets IdO par rapport aux vulnérabilités déjà existantes et connues, c'est-à-dire présentes dans le répertoire public de failles de sécurité informatique CVE. Pour ce faire, notre

16. Disponible sur `https://cve.mitre.org/`, dernier accès le 07/12/2021
17. Disponible sur `https://owasp.org/www-pdf-archive/OWASP-IoT-Top-10-2018-final.pdf`, dernier accès le 07/12/2021

<table>
<tr><th>Approche</th><th colspan="3">Type d'analyse</th><th>Références</th></tr>
<tr><td rowspan="3">statique</td><td colspan="3">code source</td><td>[141, 48, 44, 45]</td></tr>
<tr><td colspan="3">exécution symbolique</td><td>[53, 148]</td></tr>
<tr><td colspan="3">analyse de teinte</td><td>[37, 136]</td></tr>
<tr><td rowspan="4">dynamique</td><td rowspan="4">fuzzing</td><td rowspan="2">à partir de binaires émulés en</td><td>user-mode</td><td>[45]</td></tr>
<tr><td>full-system mode</td><td>[36, 90]</td></tr>
<tr><td rowspan="2">en alternant les instructions entre</td><td>deux composants émulés</td><td>[185, 89]</td></tr>
<tr><td>un binaire émulé et du matériel</td><td>[85, 184, 92, 116]</td></tr>
</table>

Tableau 2.1 – Références sur l'analyse de firmwares d'objets IdO triées par approche et type d'analyse

première contribution [189] se rapproche des analyses statiques [48, 44] mais se concentre sur la recherche de vulnérabilités connues et liées aux logiciels capables de déployer des services communément présents dans les objets IdO [36], comme par exemple HTTP, SSH ou bien DNS. Enfin, à partir des informations sur les vulnérabilités et logiciels trouvés dans les firmwares d'objets IdO, cette contribution permet également de mettre en lumière les pratiques de développement des fabricants d'objets IdO en vérifiant notamment si des logiciels ayant des vulnérabilités connues sont continuellement intégrés dans les firmwares ou si, au contraire, les fabricants mettent à jour ces derniers vers des versions sécurisées.

De plus, à l'image de [49] qui démontre en 2010 que de nombreux objets IdO connectés à l'Internet utilisent les identifiants et mots de passe par défaut, idée qui sera reprise ensuite par le botnet Mirai [13, 107] pour compromettre plusieurs centaines de milliers d'objets connectés. Nous proposons également dans une seconde contribution [188], d'utiliser les connaissances présentes dans les firmwares de plusieurs milliers d'objets IdO pour établir une méthode de *fingerprinting* basée sur des algorithmes d'apprentissage automatiques afin de prédire les composants comme les logiciels utilisés pour déployer le serveur HTTP ou bien SSH, ou les noms d'utilisateurs et mots de passes présents dans un firmware.

Chapitre 3

Étude des cyberattaques ciblant l'Internet des Objets à l'aide de pots de miel

3.1 Introduction

Dans le premier chapitre, nous avons vu que des objets IdO connectés à l'Internet peuvent être identifiés par un attaquant puis, dans le second chapitre, nous avons détaillé les différentes méthodes d'analyse de firmwares d'objets IdO permettant de détecter les vulnérabilités présentes dans ces derniers. La conclusion commune à ces analyses est qu'un grand nombre de firmwares d'objets IdO est vulnérable. Par exemple, FIRMADYNE [36] montre que 887 (45%) des objets IdO analysés souffrent au minimum d'une vulnérabilité, comme des injections de commandes (*command injection* en anglais) ou des dépassements de tampon (*buffer overflows* en anglais).

Cependant, ces études ne vérifient pas si les vulnérabilités découvertes sont exploitées ou non par des attaquants pour attaquer des objets IdO connectés à l'Internet comme illustré par le schéma d'attaques dans la figure 3.1. D'ailleurs, pour surveiller les attaques subies par un objet connecté à un réseau informatique, il est nécessaire d'avoir un accès à son trafic réseau ou son fonctionnement interne comme les opérations d'écriture, lecture ou exécution effectuées depuis l'objet concerné.

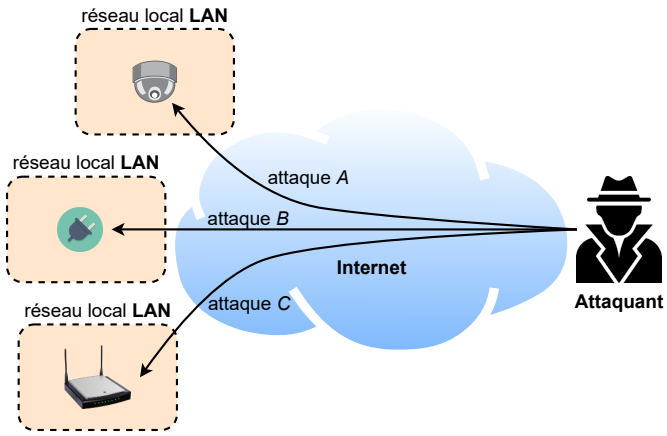


FIGURE 3.1 – Schéma d'attaques d'objets IdO connectés à l'Internet

Or, même en interceptant les paquets échangés entre un objet connecté et un attaquant, l'utilisation de protocoles basés sur le chiffrement comme HTTPS ou SSH, empêche toute tierce partie d'analyser le contenu des données échangées, réduisant ainsi l'intêret de l'observation passive de paquets. Par exemple, dans le cas d'une attaque par injection de commandes sur un serveur HTTPS, les données relatives à l'attaque comme l'URL de la page web demandée ou les entrées utilisateur utilisées ne pourront être directement extraites des paquets réseau.

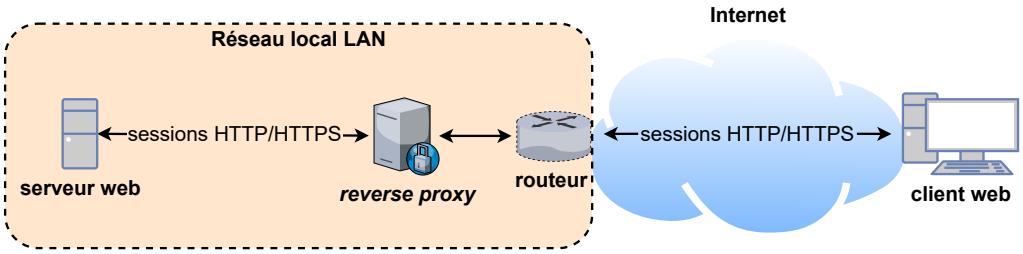


FIGURE 3.2 – *Reverse proxy* d'un serveur HTTP

Comme reporté dans une note technique de l'ANSSI [18], il est possible, par le biais d'un serveur mandataire inverse (ou *reverse proxy* en anglais), placé comme illustré dans la figure 3.2, d'intercepter les données échangées entre deux entités communicantes, même en cas de chiffrement. Dans notre cas d'étude, la faisabilité de cette solution n'est pas garantie en raison de la forte hétérogénéité dans les binaires déployant des services web ou bien dans les configurations réseau c'est-à-dire les technologies et protocoles de communication. Par exemple, dans le cas d'un objet IdO déployant un serveur web HTTPS, pour déchiffrer les requêtes d'attaquants venant de l'Internet vers ce dernier, il est nécessaire que le *reverse proxy* possède une copie de la clé privée utilisée par l'objet IdO, et que l'objet IdO ne propose pas de suites cryptographiques permettant la *Perfect Forward Secrecy (FPS)*, c'est-à-dire la génération de clés de sessions uniques.

L'étude du fonctionnement interne de l'objet nécessite souvent de pouvoir s'y connecter afin d'accéder aux informations telles que la liste des fichiers présents, les processus en cours d'exécution ou bien l'historique des commandes effectuées dans l'objet. Dans le cas d'une cyberattaque, ces informations peuvent être utiles pour mieux comprendre la séquence d'actions effectuées par l'attaquant depuis l'invite de commande ou du *terminal* de l'objet compromis. Ces actions comprennent :

— les commandes de *vérification* de certains prérequis, par exemple pour l'obtention d'une information sur le type de l'objet ou son architecture CPU pour ensuite pouvoir télécharger le virus informatique (ou *malware* en anglais) compilé pour cette dernière,

— la commande effectuant le téléchargement du *malware* auprès d'un serveur externe,

— l'exécution du *malware*,

— les commandes de *camouflage* pour supprimer les traces de l'attaque comme l'historique de commandes ou le *malware*.

D'un point de vue analytique, les informations obtenables depuis l'objet sont d'une importance considérable, car elles permettent d'analyser le fonctionnement global de l'attaque et surtout obtenir le *malware* pour ensuite proposer des contre-mesures. Cependant, il n'est pas toujours nativement possible de se connecter à distance sur un objet ni d'obtenir avec précision les

18. Disponible sur `https://www.ssi.gouv.fr/uploads/IMG/pdf/NP_TLS_NoteTech.pdf`, dernier accès le 08/12/2021

informations précédemment listées. En supposant que ces deux conditions soient réunies, l'étude du fonctionnement interne de l'objet est confrontée à plusieurs obstacles :

— la capacité de l'attaquant à bloquer aux autres utilisateurs l'accès à l'objet,

— selon les commandes de *camouflage* effectuées, il n'est pas toujours possible de récupérer l'historique des commandes et le *malware* intacts,

— suivant les actions effectuées par le virus informatique, l'objet infecté peut être mis hors d'usage via, par exemple, la destruction du contenu de sa mémoire.

Analyser les attaques reçues par un objet connecté requiert plusieurs prérequis afin d'obtenir les informations les plus précises possibles sur ces dernières. En premier lieu, les paquets réseau échangés entre l'objet connecté et l'attaquant doivent être interceptables et interprétables. Deuxièmement, le fonctionnement interne de l'objet connecté doit être également contrôlable, réutilisable et/ou réinitialisable. Néanmoins, ces deux dernières propriétés ne sont pas conformes avec un objet connecté que l'on utiliserait au quotidien.

C'est pourquoi, dès les années 1990, les tentatives d'intrusion sur des machines dédiées spécifiquement à cet effet, c'est-à-dire à recevoir des cyberattaques, ont été analysées [161, 38]. Ces machines dédiées sont configurées avec de faux fichiers ou services afin d'attirer l'attention d'attaquants pour recevoir des attaques et surtout collecter les actions effectuées par les attaquants.

Ce n'est qu'au début des années 2000, que cette approche est formalisée sous le terme pot de miel (ou *honeypot* en anglais), et est définie comme "*A honeypot is a security resource whose value lies in being probed, attacked, or compromised*" [157] en référence au fait que l'on souhaite attirer des attaquants à l'aide d'une "*security resource*" configurée à cet usage comme un serveur, un script simulant un service ou un ensemble de processus isolés (*sandbox* ou *jail*[86] en anglais). Enfin, comme rapporté par [131], d'autres études [151, 23] incluent l'enregistrement des actions effectuées par les attaquants dans leurs définitions.

Afin d'enregistrer les actions d'attaquants, il est nécessaire que le pot de miel réponde avec plus ou moins de fidélité à leurs requêtes afin d'imiter le mieux possible le fonctionnement d'une machine physique ou d'un service. C'est pourquoi, suivant la complexité des attaques que l'on souhaite collecter, les pots de miel peuvent être classés en trois catégories [157] : les pots de miel à faible interaction (ou *low-interaction honeypots (LIH)* en anglais), les pots de miel à moyenne interaction (ou *medium-interaction honeypots (MIH)* en anglais), et les pots de miel à forte interaction (ou *high-interaction honeypots (HIH)* en anglais). Dans les prochaines sections, nous détaillerons chacune de ces trois catégories en nous attardant sur les solutions dédiées à l'Internet des Objets.

3.2 Pot de miel à faible interaction

Un pot de miel à faible interaction, ou *low-interaction honeypot (LIH)* en anglais, est le type de pot de miel le plus simple et rapide à installer mais, comme illustré dans la figure 3.3, ne peut simuler que de manière limitée certains services [157]. En effet, les LIH sont généralement programmés pour répondre de manière déterministe à des requêtes prédéfinies. Ils présentent un risque quasi-nul de se faire compromettre par des attaquants. Le second avantage des LIH réside dans leur maintenance consistant principalement à ajouter ou ajuster les couples «requête, réponse» afin de pouvoir répondre à des requêtes liées à de nouvelles attaques. Par conséquent, les LIH sont parfaitement adaptés à l'analyse des demandes d'accès illégitimes ou les scans de réseau informatique mais ne sont pas adaptés à la détection d'attaques type *0-day* [40, 157].

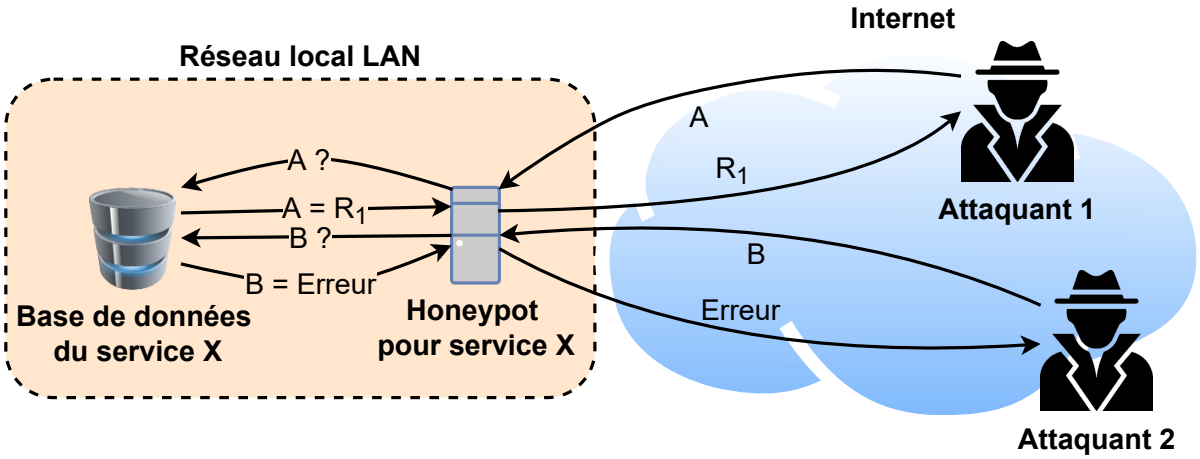


FIGURE 3.3 – Pot de miel à faible interaction (LIH) simulant un service X

Au début des années 2000, [133] présente honeyd, un *daemon*[19] UNIX installé sur une machine servant ainsi d'honeypot, capable de simuler le fonctionnement de différents systèmes d'exploitation (OS) comme Microsoft XP, FreeBSD, etc. en modifiant certains attributs des entêtes TCP, UDP et ICMP présents dans ses paquets réseau. Il est également possible d'associer à honeyd, pour un couple «numéro de port, protocole (UDP ou TCP)», un programme informatique à exécuter afin de simuler un service spécifique. Par exemple, des programmes permettant de simuler des serveurs TELNET, FTP ou SMTP ont été proposés. Dans [19], les auteurs présentent nepenthes, une infrastructure logicielle (ou *framework* en anglais) pour LIH utilisable pour des déploiements à grande échelle, et capable de simuler des services volontairement vulnérables afin d'étudier le déroulement ainsi que la vitesse de propagation d'attaques informatiques. Ce *framework* sera ensuite repris dans Dionaea [1] afin de simuler des serveurs HTTP, FTP, UPnP ou même des services de bases de données comme MySQL ou MongoDB. On pourra également citer, Kojoney [42] et Kippo [2], deux LIH simulant un serveur SSH puis, une fois l'attaquant connecté, un environnement shell. Ces deux solutions peuvent ainsi collecter les commandes de *vérification* effectuées par les attaquants et même, dans [42], déduire s'il s'agit d'un humain ou un robot en inspectant les caractères saisis dans le shell, notamment les frappes de touches comme `SUPPR` ou `BACKSPACE`. Par exemple, la commande `"ls -z[BACKSPACE]a"` indique que l'attaquant a d'abord tapé `"ls -z"` mais a corrigé le `"z"` en `"a"`, ce qui peut indiquer que ce dernier est un humain et non un robot.

Ces environnements ne sont pas spécifiques à ceux présents dans les objets IdO. Ainsi, à partir du code source (HTML) de serveurs web d'objets IdO, les serveurs web de trois objets IdO ont été simulés dans des LIH dont les performances (par exemple, le temps de réponse ou nombre de paquets) sont équivalentes à celles des objets IdO physiques [159] sauf lors de scans avec Nmap [121]. En effet, les LIH semblent plus lents, ce que l'auteur attribue au temps de traitement des requêtes Nmap par honeyd.

D'ailleurs, honeyd ne simulant pas de systèmes d'exploitation propres à des objets IdO, [104] propose une solution capable de simuler le comportement de quatre OS d'objets IdO, par exemple la caméra GoPro Hero3 ou la console Nintendo Wii, utilisable dans le cadre d'un pot de miel à faible interaction.

Afin d'étudier les attaques venant du botnet Mirai [13], les auteurs de [187] proposent un LIH

19. Voir `https://fr.wikipedia.org/wiki/Daemon_(informatique)`, dernier accès le 05/01/2022

simulant le service TELNET d'objets IdO et satisfaisant les commandes de *vérification* effectuées par Mirai. Par exemple, une fois connecté à un objet IdO, Mirai effectue la commande `"nc; wget; /bin/busybox ECCHI"` pour vérifier l'existence d'un binaire, parmi `nc` et `wget`, pour télécharger le *malware*, et dont la réponse attendue par Mirai se termine par `"ECCHI: applet not found"`. Enfin, pour vérifier les performances de leur honeypot, ces derniers ont uniquement vérifié, sur un réseau local, si une machine infectée par Mirai (recompilé à partir de son code source [20]) effectuait effectivement une attaque et un téléchargement du *malware* vers leur honeypot. Une version similaire et open source de ce LIH est proposée par [147].

En complément des attaques sur les objets IdO, les objets de l'Internet industriel des objets (IIdO) et les systèmes de contrôle industriel (*industrial control systems (ICS)* en anglais), sont également sujet à de nombreuses attaques [156, 25, 160, 166].

SCADA HoneyNet [130], basé sur honeyd, simule le fonctionnement d'ICS comme des systèmes de contrôle et d'acquisition de données en temps réel (ou *Supervisory Control And Data Acquisition (SCADA)* en anglais) ou des contrôleurs à logiques programmables (ou *Programmable Logic Controller (PLC)*) où des protocoles propres à ces systèmes comme Modbus, HTTP ou TELNET sont déployés. Les fonctionnalités de ces protocoles sont adaptées par rapport à celles trouvées sur de vrais systèmes ICS, et sont simulées à l'aide de scripts. Par exemple, leur approche peut simuler le serveur web d'un PLC de marque Schneider mais n'est plus maintenue depuis 2005. À l'aide de Dionaea et honeyd, des pots de miel dédiés aux ICS sont présentés dans [176] où des protocoles, comme Modbus et HTTP, sont également simulés. De plus, les auteurs ont déployé leurs pots de miel pendant 28 jours et ont observé 39 attaques, principalement des modifications de paramètres de configuration à l'aide du protocole Modbus ou via le serveur HTTP.

Environ 10 ans après le développement de SCADA HoneyNet [130], un projet open source permettant de simuler des ICS, comme le Siemens PLC S7-200, est proposé dans Conpot [138] en 2014. Ce LIH simule également des protocoles comme, Modbus, SNMP et HTTP. Les données et actions de ces protocoles sont définies dans des modèles (ou *templates* en anglais) facilitant ainsi leur configuration. Un exemple d'utilisation de Conpot est détaillé dans [143] où les étapes nécessaires pour simuler un compteur électrique Schneider ION6200 déployant les services HTTP, SNMP et Modbus, sont décrites. Ainsi, la configuration de SNMP, protocole permettant d'obtenir des informations sur la configuration d'une machine, est grandement facilitée par l'utilisation d'une *template* où chacune des valeurs de configuration devant être gérée par SNMP est simplement listée dans la *template*.

Contrairement à Conpot, HosTaGe [167, 168] peut simuler le service TELNET, mais également collecter plus d'information sur les attaques liées au protocole Modbus. Enfin, ces derniers déploient pendant 12 semaines leur solution et Conpot, et montrent qu'HosTaGe reçoit plus d'attaques que Conpot et également qu'il est pertinent de simuler le protocole TELNET étant donné les nombreuses attaques reçues sur ce dernier (entre 150 et 1 000 par semaine).

Malgré leur faible niveau d'interaction, les pots de miel à faible interaction restent de bons outils de détection de tentatives d'accès illégitimes ou d'attaques connues dont le risque d'être compromis par des attaquants est quasi-nul. Cependant, pour détecter des attaques récentes, il est nécessaire d'ajuster manuellement et régulièrement les services actifs et/ou interactions des LIH par rapport aux commandes de *vérification* effectuées par ces nouvelles attaques. Or, même en étant à jour, un attaquant avisé peut facilement détecter ce type d'honeypot à cause justement des interactions déterministes retournées par ce dernier.

20. Disponible sur `https://github.com/jgamblin/Mirai-Source-Code`, dernier accès le 14/12/2021

3.3 Pot de miel à moyenne interaction

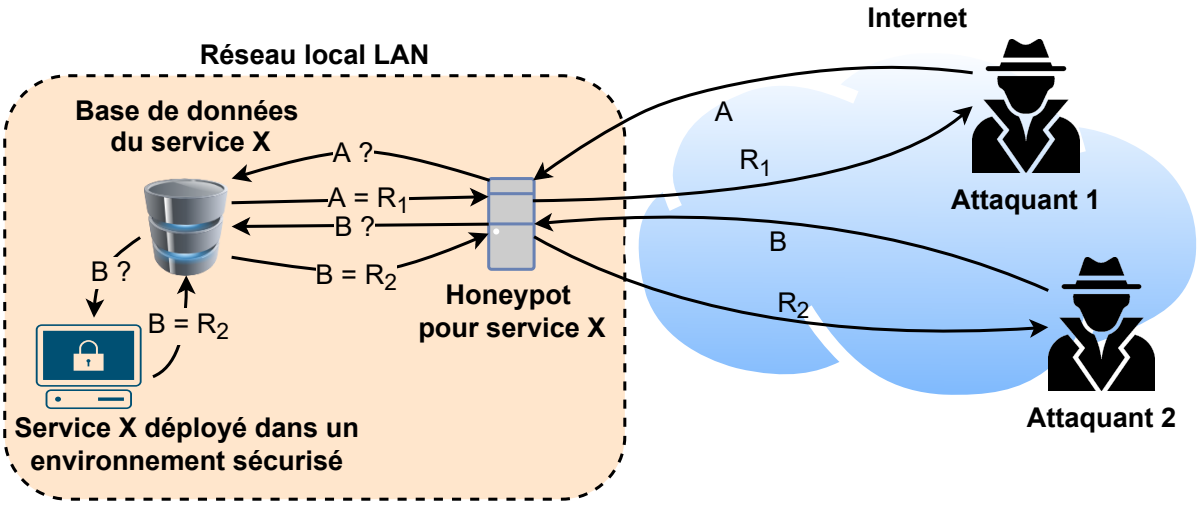


FIGURE 3.4 – Pot de miel à moyenne interaction (MIH) simulant un service X

Contrairement aux pots de miel à faible interaction, les pots de miel à moyenne interaction, ou *medium-interaction honeypots (MIH)* en anglais, peuvent répondre à des requêtes inconnues et donc plus complexes. Pour ce faire et comme illustré par la figure 3.4, un MIH peut transmettre les requêtes reçues à une application émulée et/ou exécutée depuis un environnement limité et sécurisé comme une *sandbox* ou une *jail* [86], pour ensuite renvoyer les réponses aux attaquants.

Ainsi, Cowrie [6] permet de simuler un serveur SSH ou TELNET et peut transmettre les requêtes d'attaquants vers des machines virtuelles utilisant le système d'exploitation (OS) OpenWrt [21] ou Ubuntu [22] émulé via QEMU [24]. OpenWrt pouvant être utilisé comme OS dans certains routeurs. L'utilisation de ce dernier permet ainsi d'analyser les attaques typiques de botnets ciblant les serveurs SSH et TELNET d'objets IdO [150, 99]. Par exemple, [99] ont déployé Cowrie pendant 40 jours entre février et mars 2019, et collecté 236 *malwares* associés au botnet Mirai [13]. Cependant, sur les 84 602 connexions au honeypot, seules 1.55% des connexions ne sont pas liées à Mirai mais plutôt à des attaques SSH non concrétisées dû au niveau d'interaction proposé par Cowrie.

La redirection de requêtes vers des machines virtuelles est également proposée dans IoTPOT [127] où seul OpenWrt est utilisé comme système d'exploitation et émulé, à l'aide de QEMU, sur différentes architectures CPU (par exemple MIPS, ARM, PowerPC) afin de simuler un objet IdO déployant un serveur TELNET. Ces derniers ont déployé 87 instances de leur honeypot pendant 81 jours entre avril et juin 2015, et ont observé 130 314 connexions et collecté 106 *malwares* dont 88 qui n'étaient pas répertoriés dans la base de données de *malwares* VirusTotal [23].

Une approche similaire est employée dans Chameleon [186] où le honeypot consiste à simuler un objet IdO à partir de son équivalent physique, servant notamment à générer les réponses adéquates aux requêtes des attaquants. Afin de ne pas transmettre de requêtes malveillantes à l'objet IdO physique, et donc risquer de le compromettre, les requêtes ne venant pas d'adresses IP appartenant à des réseaux informatiques *de confiance* comme ceux de Google, Amazon ou China

21. Disponible sur `https://openwrt.org/`, dernier accès le 16/12/2021
22. Disponible sur `https://ubuntu.com/`, dernier accès le 16/12/2021
23. Disponible sur `https://virustotal.com/`, dernier accès le 20/12/2021

Telecom, etc. sont bloquées, et une réponse de type erreur peut être retournée. Par exemple, une erreur HTTP 404 dans le cas d'une requête vers le serveur web. Enfin, les auteurs déploient une centaine d'instances de Chameleon, pendant 13 jours, mais montrent uniquement que leurs honeypots ne sont pas détectés comme tels par l'outil de détection d'honeypot `honeyscore`[24] proposé par Shodan[25], et que ces derniers reçoivent continuellement des attaques.

Dans ThingPot [172], l'honeypot présenté simule le fonctionnement du service web d'un contrôleur d'objets IdO Philips Hue mais peut aussi, via le protocole de messagerie instantanée XMPP, interagir avec une vraie ampoule connectée également de marque Philips Hue. Néanmoins, après un déploiement d'un mois et demi, les auteurs ont observé que les attaques reçues étaient génériques et concentrées sur le service web et non sur le client XMPP.

Également massivement déployé dans les objets IdO [114, 36], le protocole de découverte UPnP permet à des objets connectés de communiquer automatiquement entre eux. Cependant, ce dernier peut être utilisé comme vecteur d'attaque [114]. Ainsi, pour étudier les attaques sur ce protocole, [69] présentent U-PoT, une infrastructure logicielle capable, à partir d'un objet physique, d'apprendre puis de construire un binaire simulant les fonctionnalités UPnP de l'objet analysé, comme par exemple retourner la liste des actions implémentées par l'objet ou bien requêter une action. Enfin, les auteurs montrent qu'en exécutant le binaire dérivé d'un objet IdO de marque WeMo depuis une machine accessible depuis l'Internet, cette dernière est correctement identifiée comme un objet UPnP par des moteurs de recherche comme Shodan. Néanmoins, aucune analyse d'attaques n'est présentée.

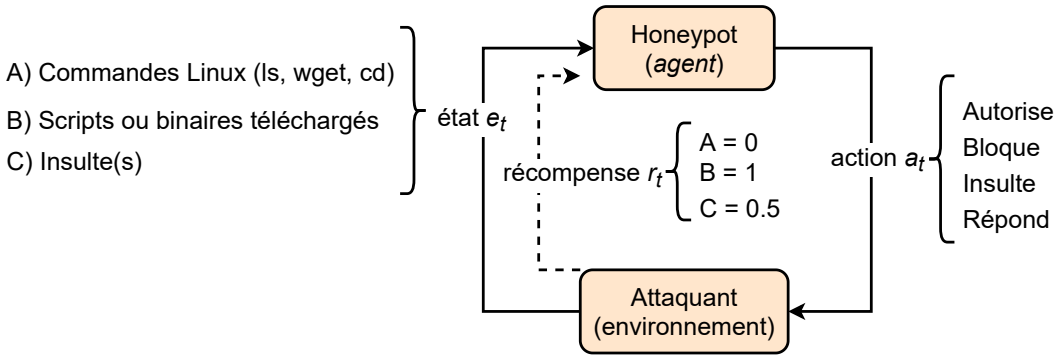


FIGURE 3.5 – Apprentissage par renforcement pour déterminer la meilleure action a_t possible à partir d'un état e_t et de sa récompense associée r_t

Plusieurs études [170, 128, 102, 129] introduisent l'utilisation d'algorithmes d'apprentissage par renforcement afin d'améliorer les interactions proposées par leur honeypot. Ce type d'approche, illustrée par la figure 3.5, consiste à faire apprendre à un agent, les meilleures actions à effectuer en fonction des états observés dans le but de maximiser la somme des récompenses retournées par l'environnement. Dans le cas d'un honeypot, on va donc chercher à obtenir la meilleure séquence d'actions permettant d'observer une demande d'exécution de script ou binaire, c'est-à-dire l'exécution d'un virus informatique.

Ainsi, Heliza [170] est un MIH simulant un serveur SSH capable d'adapter ses interactions avec les attaquants à l'aide de l'algorithme d'apprentissage SARSA qui va déterminer pour chaque état de bloquer, accepter ou modifier les requêtes reçues. Si une requête est acceptée, cette dernière est transmise à un noyau Linux exécuté en espace utilisateur (*User Mode Linux (UML)*[26]

24. Disponible sur `https://honeyscore.shodan.io/`, dernier accès le 21/12/2021
25. Disponible sur `https://www.shodan.io/`, dernier accès le 16/12/2021
26. Disponible sur `http://user-mode-linux.sourceforge.net/`, dernier accès le 17/12/2021

en anglais) afin d'obtenir la réponse correspondante qui sera ensuite renvoyée à l'attaquant. Heliza peut également insulter les attaquants pour déterminer s'il s'agit de robots ou d'humains. Enfin, [170] comparent Heliza avec un LIH et un HIH et montrent qu'une fois suffisamment entrainé, Helisa capte des attaques plus longues. Une amélioration d'Heliza sera proposée dans RASSH [128] qui utilise Kippo [2] pour déployer le serveur SSH servant d'honeypot, et améliore la détection des humains en générant automatiquement des insultes dans la langue correspondante à la géolocalisation de leurs adresses IP. RASSH ajoute également une action introduisant de la latence afin d'analyser si les attaquants adaptent leur attaques selon cette nouvelle variable.

Enfin, [129] présentent QRASSH, une version améliorée de RASSH utilisant Cowrie [6] pour simuler le serveur SSH et l'algorithme de renforcement Q-Learning profond (ou *deep Q-Learning* en anglais) pour déterminer les interactions à faire avec les attaquants. Contrairement à l'algorithme SARSA, l'algorithme Q-Learning profond va choisir, pour un état donné e_1, l'action future qui maximise la récompense afin d'atteindre un nouvel état e_2. Ainsi, [129] vont considérer 57 états (correspondant aux 57 commandes Linux gérées) et les 5 actions c'est-à-dire accepter, bloquer, insulter, produire une fausse réponse et introduire de la latence. Ensuite, l'algorithme de Q-Learning standard est appliqué afin de calculer le score de ces 5 actions pour chacun des 57 états. Enfin, un réseau de neurones est entrainé à partir de ces scores pour prédire la meilleure action à partir d'un état donné. Néanmoins, ces derniers n'ont déployé que pendant un court laps de temps, non détaillé, leur honeypot et ont observé moins de 20 attaques.

L'algorithme Q-Learning a également été utilisé par [102] pour IoTCandyJar, un honeypot capable de simuler de multiples services, par exemple HTTP(S), XMPP, UPnP. Afin d'obtenir des informations sur autant de services et ainsi pouvoir les simuler fidèlement, [102] ont d'abord déployé un honeypot low-interaction et capturé plus de 18 millions de requêtes effectuées par des attaquants. Ces dernières ont été ensuite filtrées afin d'éliminer celles contenant des actions malveillantes ou étant liées à des protocoles non utilisés dans l'Internet des Objets comme BitTorrent ou des protocoles liés à Windows. Par exemple, la détection de requêtes malveillantes est effectuée, entre autres, à l'aide de règles de pare feu ou de systèmes de détection et prévention d'intrusions, comme snort [27], et en inspectant les requêtes HTTP afin de prévenir d'attaques par traversée de répertoires (ou *path traversal* en anglais). Enfin, les requêtes considérées comme sûres sont retransmises à d'autres objets IdO connectés à l'Internet afin de récolter leurs réponses. À partir de cette base de données de «requêtes, réponses», les auteurs utilisent l'algorithme Q-Learning pour générer en temps réel les réponses de leur honeypot et montrent que ce dernier, comparé à des honeypots low-interaction, reçoit des attaques en moyenne trois fois plus longues. Néanmoins, IoTCandyJar est d'un point de vue éthique discutable, car la construction de leur base de connaissance requiert de communiquer avec des objets IdO à travers l'Internet et, potentiellement, relayer des requêtes malveillantes.

Dans cette section, nous avons détaillé les différents pots de miel à moyenne interaction, ou *medium-interaction honeypots (MIH)* en anglais, permettant d'observer des attaques sur différents services comme HTTP, TELNET, SSH ou bien XMPP. Parmi ces MIH, certains permettaient de simuler des objets IdO et donc d'observer des attaques propres à l'Internet des Objets. Pour ce faire, les réponses aux requêtes des attaquants sont principalement générées par :

— l'utilisation de machines virtuelles utilisant un système d'exploitation lié à l'Internet des Objets comme OpenWrt [6, 99, 127]. Cette méthode ne simule que de manière superficielle un objet IdO car OpenWrt n'est pas installé par défaut dans la majorité des objets IdO [173] mais seulement dans certains routeurs et souvent par les utilisateurs eux-mêmes. De plus, les réponses du MIH étant générées par des machines virtuelles, il est possible qu'un attaquant

27. Disponible sur `https://www.snort.org/`, dernier accès le 11/03/2022

expérimenté, en inspectant les composants matériels de l'objet attaqué, détecte que ce dernier soit en réalité une machine virtuelle et donc possiblement un honeypot.

— l'utilisation d'objets IdO physiques permet d'obtenir des réponses plus fidèles et donc correspondantes à un objet IdO spécifique. Cette approche a plusieurs désavantages [186, 172, 69, 102]. Tout d'abord, même en filtrant certains paramètres des requêtes provenant des attaquants, il y a toujours un risque de transmettre une attaque vers l'objet IdO physique qui, à son tour, peut également propager l'attaque à d'autres objets. De plus, le résultat d'une requête filtrée n'est pas forcément celui attendu par l'attaquant qui donc, risque de ne pas continuer son attaque. Cette approche requiert également l'utilisation d'un ou plusieurs objets IdO physiques connectés et isolés du reste du réseau afin d'éviter la propagation de *malwares* si une attaque outrepasse le filtrage. De plus, il n'est pas toujours possible de disposer d'objets IdO physiques automatiquement réinitialisables ou remplaçables rapidement en cas d'attaques infectant ces derniers. Ainsi, cette approche souffre d'un problème de passage à l'échelle.

Malgré un niveau d'interactions sensiblement amélioré par rapport aux pots de miel à faible interaction, il peut ne pas être suffisant pour intercepter des attaques complexes effectuant au préalable des vérifications, notamment sur les composants matériels ou le contenu des réponses retournées par l'honeypot. Par conséquent, pour capturer les attaques les plus perfectionnées, il est nécessaire d'augmenter une nouvelle fois le niveau d'interactions et ainsi, utiliser un pot de miel à forte interaction, ou *high-interaction honeypot (HIH)* en anglais.

3.4 Pot de miel à forte interaction

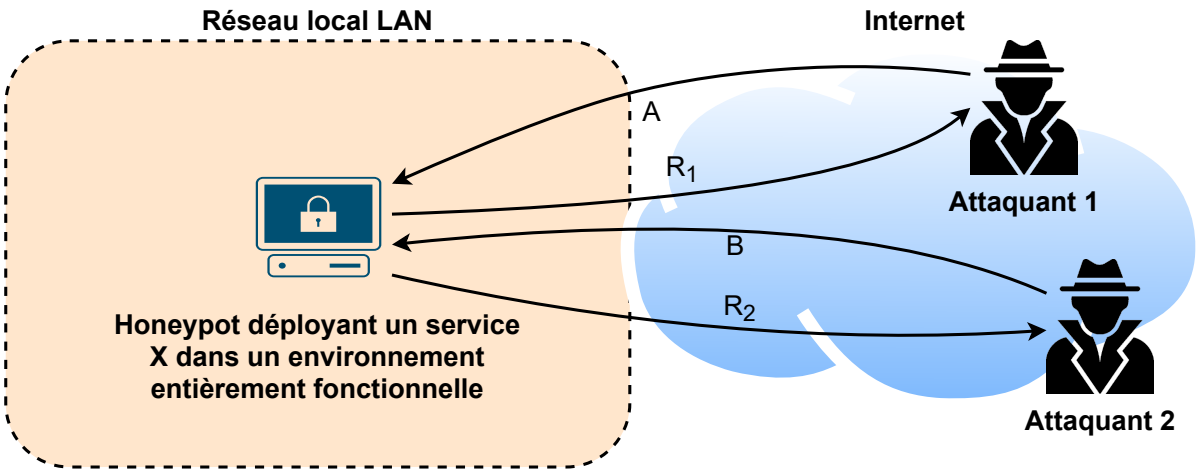


FIGURE 3.6 – Pot de miel à forte interaction (HIH) simulant un service X

Cette dernière catégorie de honeypot propose le niveau d'interactions le plus développé en permettant aux attaquants d'accéder, de manière non simulée, à l'intégralité des fonctionnalités d'une machine qui peut être physique ou virtualisée. Ainsi, des attaques de type *0-day* peuvent y être observées. En contrepartie, le honeypot peut être infecté par un virus informatique et/ou utilisé à son tour pour effectuer des attaques vers d'autres machines connectées à l'Internet. Par conséquent, le déploiement de ce type d'honeypot requiert une attention particulière afin de limiter les risques de propagation des attaques reçues.

Parmi les premières implémentations de pots de miel à forte interaction, ou *high interaction honeypots (HIH)*, on peut citer Sebek [132], un outil permettant d'enregistrer les activités d'attaquants connectés sur une machine virtuelle ou physique fonctionnant sur un système d'exploitation comme Linux, Solaris ou Windows. Dans la suite de ce paragraphe, nous détaillerons uniquement le déploiement de Sebek pour Linux, car ce système d'exploitation est le plus répandu dans l'Internet des Objets [44, 36, 173]. Dans ce cas, il est implémenté sous la forme d'un module pour le noyau Linux [28], ou *Loadable Kernel Module (LKM)* en anglais.

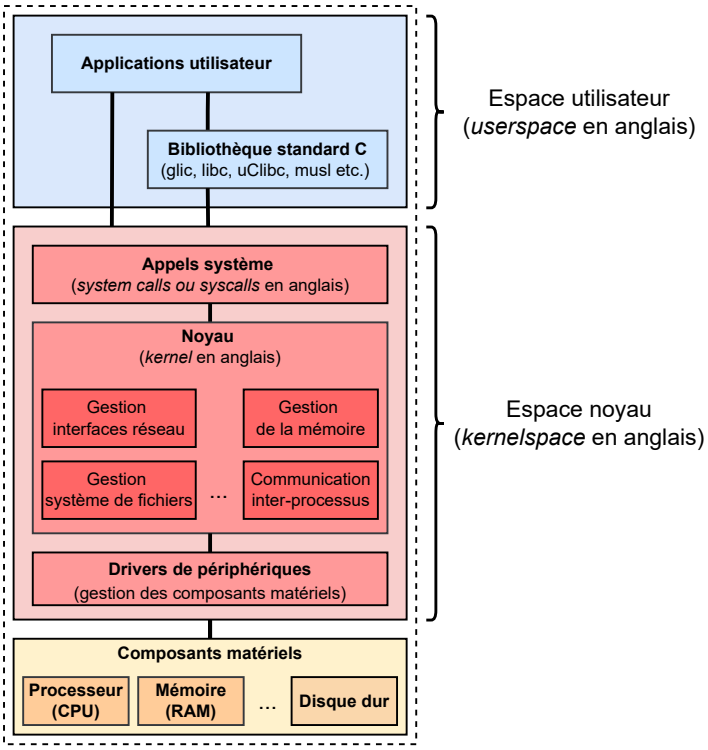


FIGURE 3.7 – Architecture simplifiée d'un système d'exploitation Linux

Comme illustré par la figure 3.7, toutes les activités effectuées dans l'espace utilisateur sont transmises à l'espace noyau afin d'y être interprétée. En effet, la bibliothèque standard C propose des fonctions implémentant des opérations courantes qui, à leur tour, peuvent appeler des appels systèmes. Ainsi, en interceptant directement les appels système, il est possible de déduire les actions effectuées par les utilisateurs en espace utilisateur peu importe la bibliothèque standard C utilisée. Par exemple, pour envoyer et recevoir des paquets réseau, une bibliothèque standard C comme *glibc* implémente généralement les fonctions `send` et `recvfrom` qui appelleront les appels système `sys_send` et `sys_recvfrom`. C'est pourquoi, l'interception de certains appels système est effectuée par Sebek pour analyser l'activité des utilisateurs qui, pour rappel, sont dans le cas d'un honeypot des attaquants.

Pour ce faire, Sebek emploie, via son LKM, la méthode illustrée par la figure 3.8 sur plusieurs appels système, notamment ceux liés aux opérations de lecture et écriture. En effet, dans l'espace noyau, l'adresse mémoire associée à chaque appel système est stockée dans la *syscall table*. Pour intercepter un *syscall* spécifique, il suffit de remplacer l'adresse mémoire originelle de ce dernier par l'adresse d'une autre fonction qui, dans le cas de Sebek, sera implémentée dans le LKM.

28. Disponible sur `https://fr.wikipedia.org/wiki/Loadable_Kernel_Module`, dernier accès le 30/12/2021

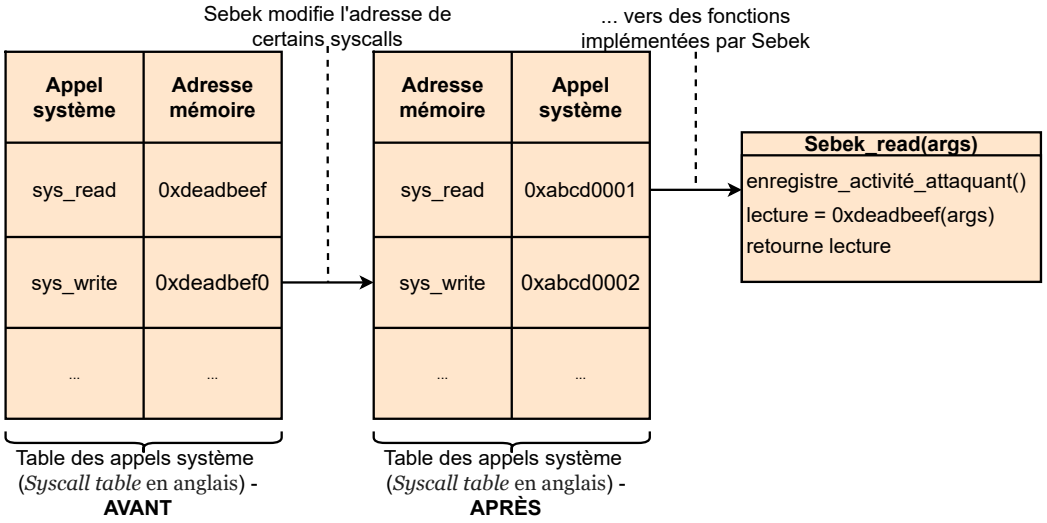


FIGURE 3.8 – Méthode employée par Sebek pour intercepter les appels système en espace noyau

Cette technique permet d'intercepter n'importe quel appel système présent dans la *syscall table*. Par exemple, Sebek intercepte, entre autres, `sys_write`, `sys_read` ou `sys_open` pour, respectivement, observer les opérations d'écriture, de lecture ou d'ouverture de fichiers effectuées par les attaquants.

Enfin, lorsqu'un *syscall* est intercepté par Sebek, les arguments de ce dernier sont analysés, selon différents critères configurables lors de l'insertion du LKM, puis transmis sur le réseau avec d'autres informations comme le numéro du processus (PID) et de l'utilisateur (UID) ainsi que l'heure à laquelle l'exécution est effectuée. Sebek permet simplement en insérant son LKM, de transformer une machine fonctionnant sous Linux en un véritable high-interaction honeypot et ainsi, étudier les activités des attaquants même si ces derniers utilisent des canaux de communication chiffrés lors de leurs attaques.

Dans SIPHON [67], les auteurs présentent un *framework* permettant d'utiliser des objets IdO physiques comme honeypot mais, contrairement au MIH Chameleon [186], les requêtes des attaquants vers les objets IdO ne sont pas altérées ainsi, des attaques peuvent compromettre les objets IdO physiques. Afin de rendre ces derniers accessibles depuis l'Internet, les auteurs utilisent, comme illustré par la figure 3.9, des tunnels SSH entre les objets IdO physiques et des adresses IP publiques appartenant à des fournisseurs cloud. De plus, il est possible de rediriger les requêtes reçues de plusieurs adresses IP vers un seul objet IdO afin d'augmenter les chances de recevoir une attaque mais aussi, en utilisant des adresses IP géolocalisables, d'en déduire les pays les plus attaqués. Pour mesurer les performances de leur honeypot, les auteurs utilisent, pendant deux mois, 39 adresses IP publiques géolocalisées dans 9 pays et reliées par tunnel SSH principalement à des caméras connectées. En effet, en déployant ce type d'objets IdO, complexes à simuler à cause par exemple du flux vidéo ou des mouvements configurables de la caméra, les auteurs espéraient observer des attaques complexes sur les interfaces web des caméras déployées. Les auteurs ont mesuré que la majorité du trafic réseau reçu (>97%) était liée au protocole SSH et non au protocole HTTP comme espéré. Enfin, parmi les attaques reçues sur HTTP, 404 attaques par *brute-force* ont été effectuées dont 11 ont trouvé la bonne combinaison <utilisateur, mot de passe> mais n'ont pas entraîné de téléchargements de *malwares*. Les auteurs ont montré que les attaquants semblaient privilégier certains pays (notamment Francfort, Allemagne ou Londres, Royaume-Uni) et que la grande majorité des attaques était envoyée depuis la Chine (>70%).

D'ailleurs, les auteurs ont également noté que le nombre de connexions vers leur honeypot a quasiment triplé après avoir été listé sur Shodan.

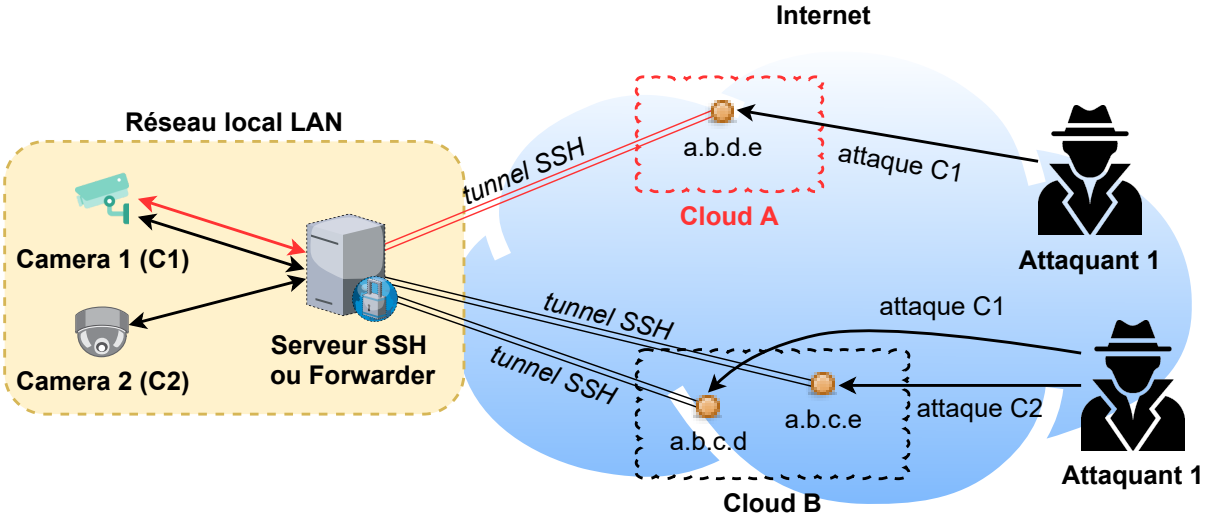


FIGURE 3.9 – Utilisation de tunnels SSH afin de rendre des objets IdO accessibles depuis l'Internet

L'hébergement d'honeypots simulant des objets IdO à partir d'adresses IP publiques appartenant à des fournisseurs cloud est détectable par des attaquants avisés. En effet, il est possible d'identifier le propriétaire d'une adresse IP à l'aide, entre autres, de la commande `whois` de *GNU*. Par exemple, l'exécution de `whois 8.8.8.8` nous informe que l'adresse IP `8.8.8.8` appartient à Google LLC (GOGL). Par conséquent, il est suspect de voir un objet IdO hébergé par un fournisseur cloud, ce qui peut donc indiquer qu'il s'agit, en fait, d'un pot de miel. Ainsi, à l'instar de SIPHON, des objets IdO sont également connectés à des adresses IP publiques mais, appartenant à des fournisseurs de VPN [162]. Pour contrer la censure ou garantir son anonymat sur l'Internet, l'utilisation de VPN s'est largement démocratisée, c'est pourquoi les auteurs supposent que cette approche gagne en furtivité. Entre avril 2017 et août 2018, 44 instances de leur honeypot ont été déployées, et reçues plus de 150 millions de connexions dont la majorité provenait de Chine (64%) puis des États-Unis (14%) mais, contrairement à SIPHON, 54% des connexions étaient destinées au protocole TELNET, alors que les 46% restant au protocole HTTP. De plus, les auteurs ont également observé que les attaquants privilégiaient les interactions avec les objets IdO de marque D-Link (64%) puis Trendnet (14%). Enfin, parmi les attaques reçues, 86 748 tentatives de connexion ont été observées sur HTTP, et 147 663 sur TELNET mais, seules 5 136 d'entres elles ont trouvé la bonne combinaison <utilisateur, mot de passe> et initié une connexion vers une adresse IP externe, typiquement un serveur pour télécharger un *malware*. Hormis pendant une période de 45 jours, les connexions vers des adresses IP externes étaient bloquées, c'est pourquoi seulement 78 *malwares* différents ont été téléchargés.

D'après un rapport de McAfee [93], les attaques de type *fileless*, c'est-à-dire ne nécessitant pas le téléchargement de *malwares* pour exécuter des actions malveillantes sur les machines infectées, ont augmenté de 432% entre 2017 et 2018, notamment dû à l'attaque *Gold Dragon*[29]. Cette dernière, cible le système d'exploitation Windows, et consiste à envoyer par email une pièce jointe contenant un document Microsoft Word qui, une fois ouvert, exécute à l'insu de l'utilisateur

29. Disponible sur `https://mcafee.com/blogs/other-blogs/mcafee-labs/gold-dragon-widens-olympics-malware-attacks-gains-permanent-presence-on-victims-systems/`, dernier accès le 03/01/2022

des macros en *Visual Basic* initiant une communication sécurisée vers un serveur appartenant à l'attaquant. D'autres exemples d'attaques de type *fileless* peuvent être la modification de mots de passe, ou la destruction de données sur la machine infectée. Ainsi, afin d'étudier ce type d'attaques ciblant des objets IdO, [51] déploient deux types d'HIH à partir :

— d'objets IdO physiques fonctionnant avec le système d'exploitation OpenWrt et déployant un serveur TELNET via BusyBox [30] ainsi qu'un serveur SSH via une version modifiée de *Dropbear* [31] permettant de collecter les actions effectuées par les attaquants lorsque ces derniers sont connectés sur l'objet car, contrairement à TELNET, le chiffrement utilisé par SSH ne permet pas d'extraire directement les actions des attaquants à partir des paquets réseau. Chacun des objets IdO est ensuite connecté à l'Internet par un fournisseur d'accès internet, comme Comcast ou Unicom.

— d'objets IdO fonctionnant également avec le système d'exploitation OpenWrt mais émulés par QEMU sont déployés, par centaine, dans huit clouds comme, par exemple, Microsoft Azure ou Google Cloud. Afin d'empêcher les attaquants de détecter que l'objet IdO attaqué est émulé, plusieurs solutions ont été implémentées afin de falsifier les informations liées aux composants matériels. Par exemple, en cas d'émulation, les informations dans `/proc/cpuinfo`, comme le modèle du processeur ou l'identifiant de son fabricant, peuvent indiquer que ce dernier est en fait virtualisé, c'est pourquoi les auteurs modifient le contenu de ce fichier avec des informations d'un vrai processeur utilisé dans des objets IdO.

Dans le cas où un *malware* infecte un des honeypots, les auteurs ont mis en place un mécanisme de réinitialisation automatique d'un HIH physique ou virtuel afin de limiter les risques de voir leur honeypot propager un *malware* et enfin, recapturer de nouvelles attaques. Pour ce faire, chaque honeypot échange périodiquement des messages de type ping-pong avec un serveur qui, en cas de non réception de trois messages consécutifs, réinitialise le honeypot.

Après avoir déployé leurs solutions entre juin 2017 et juin 2018, les auteurs ont observé qu'en plus des protocoles SSH, TELNET et HTTP, le protocole *Server Message Block (SMB)* qui permet à deux machines de partager des ressources (fichiers ou périphériques), était également utilisé de manière non négligeable (entre 6% et 9%) par les attaquants. Enfin, 80 000 attaques (0.05%) étaient de type *fileless*, alors que 750 000 (46%) d'entre elles nécessitaient le téléchargement d'un virus informatique, principalement une variante de Mirai (entre 73.3% et 80.2% des *malwares*). De plus, les attaques vers les objets IdO émulés sont en moyenne 39% moins nombreuses que sur les objets IdO physiques ce qui, selon les auteurs, serait dû :

1. aux suspicions des attaquants de voir des objets IdO hébergés dans un cloud public,
2. aux mécanismes de protection contre les attaques établies par les fournisseurs cloud pouvant donc bloquer directement certaines attaques,
3. aux vérifications effectuées par les attaquants pour détecter que l'objet est exécuté depuis une machine virtuelle.

Contrairement aux pots de miel utilisant des objets IdO physiques, il est difficile de simuler précisément un objet IdO à partir d'un environnement émulé, notamment via QEMU. Par exemple, dans [6, 127, 51], les objets IdO émulés utilisent le système d'exploitation OpenWrt. Or, une analyse des systèmes d'exploitation présents dans l'Internet des Objets conclut que le noyau Linux version 2.6.36 est le plus répandu [173]. De plus, OpenWrt est principalement déployé dans des routeurs, ou installé par les utilisateurs eux-mêmes. Ainsi, un attaquant ne pourra pas cibler une vulnérabilité particulière à un modèle ou marque d'objet IdO. De plus, pour masquer

30. Disponible sur `https://busybox.net`, dernier accès le 03/01/2022
31. Disponible sur `https://matt.ucc.asn.au/dropbear/dropbear.html`, dernier accès le 03/01/2022

l'environnement émulé aux attaquants, il est nécessaire de modifier le contenu de certains fichiers de configuration [51]. L'usage d'objets IdO physiques dans des pots de miel propose l'environnement le plus réaliste mais souffre d'un problème de passage à l'échelle, et requiert de déployer régulièrement de nouveaux objets IdO pour capturer de nouvelles attaques.

Afin de proposer un environnement émulé aussi proche possible qu'un objet IdO physique, Honware [169] propose d'utiliser l'outil d'émulation d'objets IdO, FIRMADYNE [36], afin de déployer des pots de miel à forte interaction à partir de firmwares d'objets IdO. Ainsi, une fois l'objet IdO émulé, il suffira ensuite de le connecter à l'Internet. Pour ce faire, les auteurs ont amélioré l'algorithme configurant les interfaces réseau présentes dans l'objet IdO émulé, et aussi entre QEMU et la machine hôte, c'est-à-dire celle exécutant QEMU. Cet algorithme consiste, comme initialement dans [36], à analyser les opérations réseau interceptées par le module de FIRMADYNE en espace noyau puis, dériver les commandes `iptables` ou `ip` permettant de reproduire la configuration réseau attendue mais aussi, compléter ces dernières en assignant, par exemple, une adresse IP aux interfaces réseau trouvées. L'ensemble de ces commandes est ensuite disposé dans un script shell qui sera automatiquement exécuté par l'objet émulé lors de son démarrage. Ainsi, l'exécution de ce script permet de configurer les interfaces réseau de l'objet IdO émulé et rendre accessible ce dernier depuis le réseau. Il est également possible de placer dans ce script shell des commandes initiant certains services, comme des serveurs web [90]. De plus, afin de détecter les erreurs lors de l'émulation d'un firmware d'objet IdO, les signaux d'erreurs comme `SIGSEGV`, `SIGABRT` ou bien `SIGFPE` retournés par les programmes sont analysés par Honware afin de localiser le moment où une erreur se produit, et ainsi proposer une correction afin de pouvoir émuler l'objet IdO sans erreurs critiques.

Par exemple, la mémoire RAM non volatile (*Non-volatile random-access memory (NVRAM)* en anglais) peut être utilisée par un objet IdO pour récupérer certains paramètres de configuration, comme son adresse IP ou son modèle. À l'instar de [36, 90], le fonctionnement de la NVRAM est simulé à partir d'une librairie partagée connaissant un ensemble de couples <clé, valeur> prédéfini. Or, dans le cas où la clé demandée est inconnue, une valeur par défaut, comme NULL, peut être retournée, et malencontreusement générer une erreur. Ainsi, via les signaux d'erreurs, Honware peut détecter ces dernières et générer une nouvelle version de la librairie partagée incluant la clé manquante avec une valeur par défaut différente.

Grâce à ces améliorations, les auteurs montrent qu'Honware est capable d'émuler environ trois fois plus d'objets IdO que FIRMADYNE, et que ces derniers sont accessibles sur le réseau. Enfin, les auteurs montrent que les temps de réponse entre un objet IdO physique et sa version émulée par Honware sont similaires ainsi, les attaquants ne pourront pas détecter leur HIH en analysant ces métriques. Enfin, plusieurs instances d'Honware ont été déployées et des attaques sur les protocoles DNS et UPnP ou plus traditionnellement HTTP(S) y ont été observées. Les auteurs montrent également qu'en 14 jours de déploiement, un de leur honeypot simulant un routeur Tp-Link TD-8960N a reçu 566 attaques sur son interface web, et téléchargé 49 *malwares* dont 16 (32.7%) d'entres eux n'étaient pas répertoriés dans la base de données de *malwares* VirusTotal. Contrairement à [51], aucun mécanisme n'est présenté pour masquer l'environnement émulé aux attaquants. De plus, il est à noter que le code source de ce pot de miel n'est pas disponible.

3.5 Synthèse

Dans ce chapitre, nous avons détaillé les solutions, listées dans le tableau 3.1, permettant de déployer des pots de miel à faible interaction, moyenne interaction et forte interaction, en nous attardant sur les solutions dédiées à l'Internet des Objets. De plus, un résumé des fonctionnalités

Pot de miel	**Cible**	**Interactions**			**Références**
		Prédéfinies	Générées par des machines		
			physiques	virtuelles	
à faible interaction	toutes machines	✓	×	×	[133, 19, 1, 42, 2]
	objets IdO	✓	×	×	[104, 187, 159]
	ICS	✓	×	×	[130, 176, 138, 143, 167, 168]
à moyenne interaction	toutes machines	×	×	✓	[6, 170, 128, 129]
	objets IdO	×	×	✓	[127, 150, 99]
	objets IdO	×	✓	×	[102, 186, 172, 69]
à forte interaction	toutes machines	×	✓	✓	[132]
	objets IdO	×	✓	×	[67, 162, 51]
	objets IdO	×	×	✓	[51, 169]

Tableau 3.1 – Références sur les pots de miel triées selon le type de machine ciblée et la méthode de générations des interactions utilisée

de chacun de ces trois types d'honeypot est également présenté dans le tableau 3.2. Néanmoins, les particularités de l'Internet des Objets, notamment le large nombre d'objets IdO connectés à des réseaux privés ou à l'Internet et surtout l'hétérogénéité de leurs fonctionnalités, font qu'il est souvent nécessaire d'adapter les pots de miel traditionnels à ce nouveau paradigme.

Pot de miel	**Installation**	**Maintenance et configuration**	**Information sur les attaques**	**Risque de compromission**
à faible interaction	Facile	Facile	Limitée	Faible
à moyenne interaction	Compliqué	Compliqué	Variable	Moyen
à forte interaction	Difficile	Difficile	Détaillé	Élevé

Tableau 3.2 – Comparaison des trois types de pot de miel selon [157]

Dans le cas des pots de miel à faible interaction (ou *low interaction honeypot(LIH)* en anglais), nous avons vu que les solutions proposées, afin d'intercepter les attaques ciblant les objets IdO, se focalisaient sur une simulation *limitée* des protocoles les plus attaqués ou présents dans les objets IdO comme TELNET [130, 176, 167, 168, 147, 187], HTTP [1, 138, 159] ou SSH [42, 2]. Les solutions présentées n'ont pas toutes été déployées. Néanmoins, celles déployées ont effectivement intercepté des attaques dont certaines ciblaient spécifiquement les objets IdO comme celles liées au botnet Mirai [13]. Toutefois, pour intercepter des attaques spécifiques, il est nécessaire de connaître à l'avance les commandes de *vérification* effectuées par cesdites attaques. Ainsi, un LIH n'est pas adapté pour intercepter des attaques *0-day* ou effectuant des commandes de *vérification* non gérées par le LIH. C'est pourquoi, une fois déployé dans un réseau privé, ce type d'honeypot reste un bon outil de détection de tentatives d'accès illégitimes ou d'attaques connues tout en présentant un faible risque d'être compromis ce qui réduit ainsi, le temps de maintenance de ce dispositif.

De plus, compte tenu de leur niveau d'interactions limité, il est donc possible pour un attaquant avisé de détecter un LIH par le biais de commandes de *vérification* mais aussi en analysant les paquets réseau envoyés par le LIH. C'est pourquoi, certaines solutions [159, 104] s'inspirent d'honeyd [133] et modifient les paquets envoyés par le LIH, plus précisément la valeur de certains attributs présents dans les entêtes de protocoles comme TCP ou UDP, afin de faire identifier ce dernier comme étant un objet IdO standard en cas d'analyse par un logiciel comme Nmap [121]. Cette approche implique une analyse préalable des objets IdO que l'on souhaite imiter ce qui n'est pas exploitable à grande échelle au vu du grand nombre d'objets IdO présents sur le marché.

Ainsi, dans l'objectif de capturer des attaques complexes ou *0-day* ciblant des objets IdO, l'usage d'un LIH n'est pas souhaitable. En effet, pour simuler un environnement semblable à celui d'un objet IdO déployant un ou plusieurs protocoles, il est nécessaire de générer manuellement, et à l'avance, les réponses censées être générées par l'objet IdO aux requêtes des attaquants. Par conséquent, au vu du grand nombre d'objets IdO, de protocoles et de requêtes possibles, cette approche requerrait trop de temps de mise en place pour être capable de capturer les attaques listées ci-avant. C'est pourquoi, il est préférable d'utiliser soit un pot de miel à moyenne ou forte interaction.

Parmi les MIH présentés, deux méthodes ont été présentées pour, à partir de requêtes des attaquants, générer des réponses semblables à celles d'un objet IdO que l'on souhaite simuler. Tout d'abord, plusieurs travaux [6, 2, 170, 128, 127, 129, 150, 99] utilisent des machines virtuelles fonctionnant avec le système d'exploitation OpenWrt, notamment déployé dans des routeurs [173]. Même si cette première approche permet, en effet, de générer des réponses semblables à celles d'un routeur, il est uniquement possible de simuler des objets IdO génériques car, d'après les analyses de firmwares d'objets IdO [44, 189, 173], de nombreux logiciels sont en réalité utilisés pour déployer les interfaces web, les serveurs SSH ou FTP, etc. Néanmoins, cette méthode s'est montrée efficace pour capturer des attaques venant de robots [127, 150, 99]. À l'instar des LIH, l'utilisation d'un environnement virtuel peut être détectée par les attaquants via des commandes de *vérification*, notamment pour lister les composants matériels (processeur ou mémoire) de l'objet.

C'est pourquoi, d'autres travaux [102, 69, 172, 186] utilisent de vrais objets IdO afin de permettre à leur MIH de simuler exactement le comportement de ces derniers, mais aussi de pallier aux détections citées ci-avant. Ces travaux s'apparentent principalement à des preuves de concepts [32] où les auteurs présentent une infrastructure logicielle permettant de transmettre les requêtes des attaquants vers un objet IdO physique, puis renvoyer les réponses générées par ce dernier aux attaquants. Ainsi, des attaques sur des protocoles comme UPnP ou XMPP qui étaient jusqu'alors négligées par rapport aux protocoles comme TELNET, SSH ou HTTP, peuvent être interceptées.

Néanmoins, afin d'éviter aux objets physiques d'être compromis et/ou inutilisables au cours d'une attaque, [186] propose de bloquer les requêtes d'attaquants ne provenant pas de *réseaux de confiance*, ce qui est contre-intuitif pour un honeypot. Cependant, cette mesure peut s'expliquer par la difficulté ou l'impossibilité pour certains objets IdO d'être réinitialisés, sur demande, vers un état stable. Par conséquent, la maintenance des objets IdO utilisés peut être un frein au déploiement de ce genre d'honeypot. D'un autre côté, la contribution de [102] n'a pas ces problèmes car elle ne nécessite pas d'avoir un accès physique aux objets IdO. En effet, leur solution utilise les objets IdO accessibles depuis l'Internet pour générer les réponses aux requêtes des attaquants, ce qui est discutable éthiquement et présente un risque de propager des requêtes malveillantes et donc, des attaques.

32. Voir `https://fr.wikipedia.org/wiki/Preuve_de_concept`, dernier accès le 05/01/2022

Comme détaillé ci-dessus, les MIH existant et s'appuyant sur des objets IdO physiques ou virtuels semblent peu adaptés au contexte de l'Internet des Objets car d'un côté, la virtualisation des objets IdO reste trop générique et ne permet donc pas de simuler des objets IdO de marques spécifiques alors que d'un autre côté, l'utilisation d'objets IdO physiques permet la simulation la plus fidèle d'objets IdO mais nécessite d'être capable de gérer automatiquement les objets s'ils deviennent compromis et, évidemment, laisser les attaquants interagir pleinement avec ces derniers afin de pouvoir intercepter plus d'attaques. De plus, déployer une multitude de MIH où chaque instance simule un objet IdO différent requerrait de posséder une large collection d'objets IdO ce qui n'est pas, logistiquement ou financièrement, toujours faisable.

Ainsi, en considérant les avantages et inconvénients des LIH et MIH détaillés ci-dessus, notre troisième contribution consiste, à l'image d'Honware [169], en une implémentation d'un pot de miel à forte interaction, ou *high interaction honeypot (HIH)* en anglais, à partir de l'émulation de firmwares d'objets IdO. En effet, même s'il est également possible d'utiliser des objets IdO physiques dans des HIH [67, 51], nous pensons que ces derniers souffrent de problèmes similaires à leurs équivalents en moyenne interaction notamment, la difficulté de déployer rapidement et maintenir des instances d'honeypots simulant des objets IdO différents. Néanmoins, cette approche peut s'avérer particulièrement pertinente si utilisée par des fabricants d'objets IdO eux-mêmes pour tester la sécurité de leurs produits.

D'un autre côté, Honware [169], en améliorant FIRMADYNE [36], permet d'émuler via QEMU un objet IdO à partir de son firmware et de le déployer comme HIH. Ainsi, l'objet émulé aura les mêmes propriétés logiciels que son homologue physique à la différence du système d'exploitation qui sera nécessairement un noyau Linux avec une version fixe dans l'implémentation d'Honware. Contrairement aux approches utilisant des objets IdO physiques, cette méthode ne nécessite qu'un firmware et un noyau Linux ce qui, est avantageux, car certains fabricants d'objets IdO mettent à disposition en ligne les firmwares de leurs produits. Ainsi, cette méthode nous paraît la mieux adaptée pour évaluer la sécurité des objets IdO, car elle permet de déployer, en quelques minutes, un HIH à partir de n'importe quel firmware d'objet IdO et ainsi, vérifier si ce dernier est vulnérable ou non à des attaques connues ou *0-day*.

Néanmoins, contrairement à Honware dont le code source n'est pas disponible, notre contribution améliore la technique d'émulation proposée par FIRMADYNE, détaillée dans le chapitre 2, afin de permettre une meilleure évolutivité du code et intégrer de nouvelles fonctionnalités, notamment les mécanismes permettant de cacher la partie émulation aux attaquants lorsque ces derniers sont connectés sur le honeypot. En effet, les études présentées notent que, durant des attaques, les attaquants effectuent des commandes de *vérifications* qui, selon les réponses de honeypot à ces dernières, décident de poursuivre ou non leurs attaques. De plus, cet aspect est peu ou pas développé dans les articles présentés alors qu'il s'agit pourtant d'un point déterminant dans le déroulement d'une attaque. Par exemple, un attaquant peut détecter un HIH utilisant Sebek [132] en vérifiant les adresses mémoires de certains appels systèmes localisées en espace noyau comme `sys_read` ou `sys_write`.

C'est pourquoi, dans notre contribution, nous nous attardons sur les mesures à effectuer en espace utilisateur et noyau afin de masquer aux attaquants les fichiers que FIRMADYNE et Honware ajoutent au firmware de l'objet IdO à émuler. Enfin, puisque le risque est élevé de voir notre HIH se faire compromettre par des attaques qui, possiblement, seront propagées à d'autres machines connectées au réseau local ou à l'Internet. Notre contribution détaille également les techniques employées pour isoler notre pot de miel du reste du système ainsi que la configuration d'un système de prévention d'intrusion (IPS) déployé pour justement limiter la propagation d'attaques.

Deuxième partie

Analyse du trafic réseau chiffré d'objets IdO

Chapitre 4

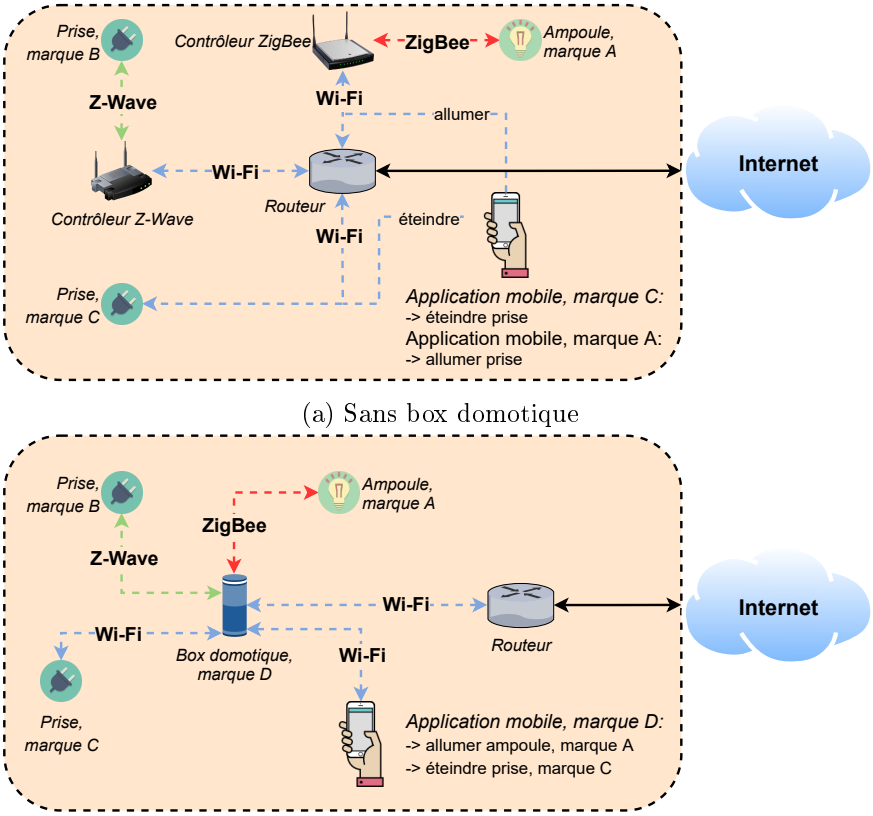
Analyse en boîte-grise des communications d'une box domotique

Sommaire

4.1 Introduction

Afin de rendre une *maison intelligente* à l'image de celle illustrée par la figure 4.1, les objets IdO peuvent être déployés et accessibles suivant plusieurs méthodes. La première, et la plus simple, consiste simplement à brancher un objet IdO puis utiliser l'application mobile ou interface web associée pour interagir avec ce dernier. Si l'on observe l'exemple de *maison intelligente* en figure 4.1(a), plus le nombre d'objets déployés de marques différentes est grand, plus le nombre d'applications mobile et/ou interfaces web à utiliser pour interagir avec ces derniers est important. C'est pourquoi, afin de d'interagir plus facilement avec un ensemble d'objets IdO, des box domotiques existent, notamment la Google Home ou Amazon Alexa. Ces dernières permettent, comme illustré par la figure 4.1(b), d'interagir avec des objets IdO de marques ou de protocoles de communication différents via une unique application mobile. Cette propriété permet de maximiser le nombre d'objets avec lesquels la box domotique peut communiquer car de nombreux

protocoles de communication principalement sans-fil, comme Z-Wave, 6LoWPAN, ZigBee ou Wi-Fi [140], sont utilisés dans l'Internet des Objets. Certains protocoles peuvent également être propriétaires, c'est-à-dire implémentés par des fabricants eux-mêmes et dont la documentation n'est pas toujours publiquement accessible.



(a) Sans box domotique

(b) Avec box domotique

FIGURE 4.1 – Exemple de déploiement d'objets IdO dans une *maison intelligente*

Cette pluralité de protocoles de communications fait qu'il est souvent compliqué d'inspecter le trafic réseau sans-fil d'un ensemble d'objets IdO connectés à un réseau informatique pour ensuite détecter si des activités malveillantes vers ou depuis ces objets sont en cours, ou inférer les actions effectuées par les utilisateurs sur ces derniers. En effet, l'utilisation de protocoles de communication sans-fil impose aux attaquants et/ou aux systèmes de détection ou prévention d'intrusions (IDPS) d'être à proximité géographique de ce type d'installation et de savoir interpréter les données échangées par ces protocoles, ce qui n'est pas toujours possible.

Une fois associé à l'application mobile de son fabricant, il est probable de voir un objet IdO communiquer avec un des serveurs de son fabricant, généralement localisé dans le cloud, et qui permet ainsi d'interagir avec cet objet à distance.

Comme illustré par la figure 4.1(b), en associant des objets IdO à une box domotique, cette dernière peut servir de passerelle de communication entre ces objets, l'utilisateur et un serveur localisé dans le cloud. En d'autres termes, la box domotique va recevoir, depuis un des serveurs de son fabricant, les actions demandées par l'utilisateur puis, va les retransmettre en utilisant le protocole de communication approprié vers les bons objets IdO. Dans ce chapitre, nous utiliserons la notation A-B pour désigner les communications de A vers B, et l'expression action utilisateur pour qualifier l'action demandée par l'utilisateur.

Ainsi, par rapport à la section 1.3 de l'état de l'art où les études sont focalisées sur les communications objet IdO-serveur, nous nous intéressons aux communications d'une box domotique contrôlant plusieurs objets IdO afin de mesurer les informations qu'un attaquant localisé à l'extérieur du réseau privé pourrait inférer uniquement à partir d'une écoute passive des communications serveur-box domotique.

Afin de détailler notre analyse d'une box domotique [191], nous avons découpé ce chapitre en trois sections en commençant dans la section 4.2 par la description du fonctionnement d'une box domotique et de ses différentes interactions possible avec ses objets associés. En effet, étant en analyse de type boîte grise, nous avons le contrôle de la box domotique et pouvons donc transmettre des actions vers ses objets associés. Néanmoins, nous n'avons pas accès aux codes sources de la box domotique (c'est-à-dire son firmware) ni du serveur utilisé par le fabricant pour relayer les requêtes utilisateur envoyées depuis l'Internet vers la box domotique. Puis dans la section 4.3, nous détaillons comment notre approche peut être utilisée pour inférer les actions demandées par les utilisateurs même si les communications box domotique serveur ou serveur-box domotique sont chiffrées. Enfin, nous discutons dans la section 4.4 des performances de notre approche pour inférer les actions utilisateur dans un environnement domotique où une box domotique peut contrôler jusqu'à 16 objets IdO.

4.2 Modélisation de l'environnement domotique

Dans cette section, nous présentons l'environnement domotique étudié, puis nous discuterons des challenges inhérents à ce type d'installation, ainsi que des hypothèses émises afin de formaliser notre approche.

4.2.1 Environnement domotique

L'environnement domotique étudié repose sur l'utilisation d'une box domotique installée comme illustré par la figure 4.2. Nous supposons que la box domotique ainsi que ses objets IdO associés appartiennent au même fabricant. Dans le cas illustré, nous avons quatre objets IdO : deux ampoules et deux prises connectées, liés à une box domotique branchée au routeur de son réseau local et qui peut communiquer avec un ou des serveurs de son fabricant à travers l'Internet.

Le fonctionnement général de ce type d'environnement domotique est également montré dans la figure 4.2 où les quatre étapes permettant à un utilisateur d'envoyer des actions sur ses objets IdO peuvent être décrites comme suit :

1 Par le biais de l'application mobile du fabricant de la box domotique, l'utilisateur u choisit les objets IdO sur lesquels effectuer une action. Dans notre exemple, u souhaite allumer l'ampoule numéro 1 et la prise numéro 2.

2 Une requête contenant ces deux actions est ensuite envoyée au serveur appartenant au fabricant.

3 Le serveur va ensuite reformater cette première requête pour en envoyer une seconde, équivalente, mais à destination de la box domotique appartenant à l'utilisateur.

4 Enfin, cette dernière va ensuite interpréter la requête puis envoyer chaque action au bon objet IdO, c'est-à-dire l'ampoule numéro 1 et la prise numéro 2, en utilisant le protocole de communication approprié.

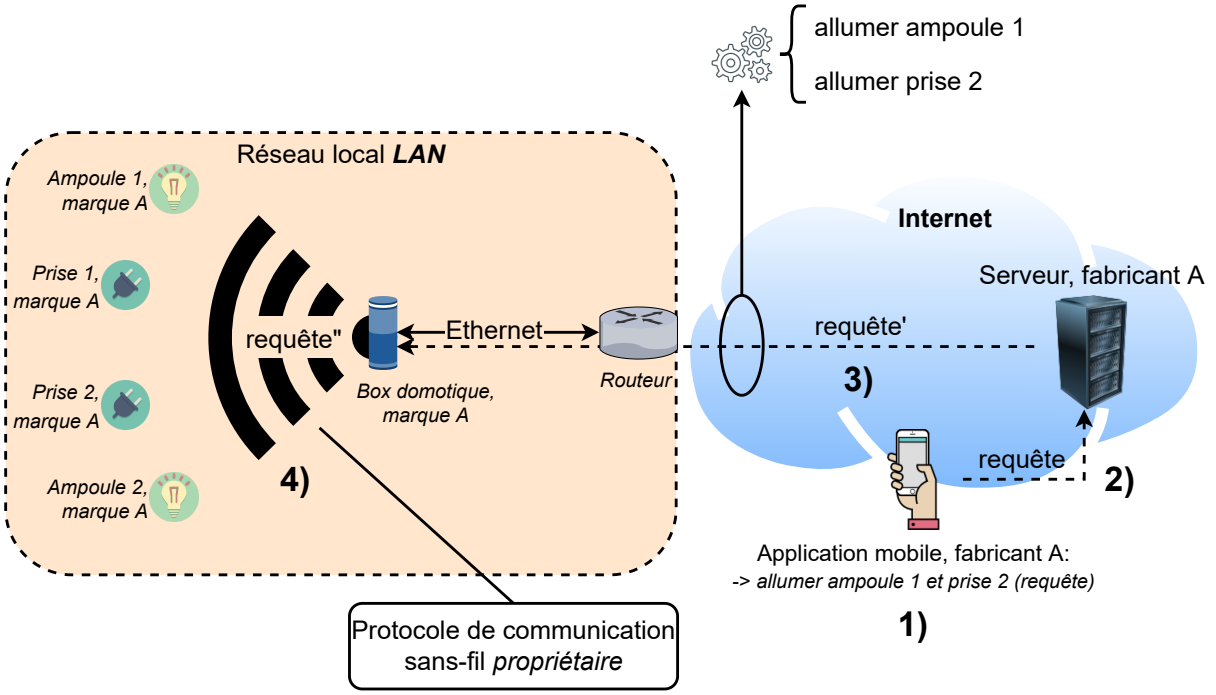


FIGURE 4.2 – Exemple de scénario où un utilisateur commande, via une application mobile, une action à deux objets associés à une box domotique

Lors de l'étape **3**, la requête est donc envoyée d'un serveur localisé dans le cloud vers la box domotique de l'utilisateur. C'est à ce moment-là, que des attaquants pourraient intercepter cette requête et tenter de l'interpréter. Nous allons chercher à vérifier cette possibilité.

4.2.2 Challenges et hypothèses

Le positionnement des attaquants à l'extérieur du réseau privé et l'utilisation de la box domotique rendent l'interprétation de la requête envoyée lors de l'étape **3**) plus complexe dû, tout d'abord, aux contraintes suivantes :

C1 Les communications box domotique-serveur et serveur-box domotique traversent l'Internet. Il est probable que ces dernières soient chiffrées. Dans le cas contraire, la box domotique et le serveur pourraient être sujets à de nombreuses attaques. Pour ce faire, l'établissement d'un canal de communication chiffré entre la box domotique et son serveur se fait traditionnellement de la manière suivante :

1 La box domotique va tout d'abord utiliser le protocole DNS afin d'obtenir l'adresse IP d'un serveur à partir d'un nom de domaine dont la valeur est généralement insérée dans la mémoire de la box domotique lors de sa fabrication, et est donc utilisable par la box une fois cette dernière en fonctionnement.

2 Une fois la résolution DNS effectuée et l'adresse IP obtenue, la box domotique va pouvoir entamer une connexion sécurisée avec le serveur via l'utilisation de protocoles, comme *Transport Layer Security (TLS)* [55, 56, 137]. Avant de commencer à échanger des données, la box et le serveur vont négocier les paramètres de ce nouveau canal de communication. Durant cette négociation ou *handshake* en anglais, les deux participants vont s'authentifier mutuellement afin de s'assurer que l'identité du serveur et/ou

de la box domotique n'a été usurpée par un attaquant lors d'une attaque homme du milieu ou *man-in-the-middle (MITM)* en anglais puis, vont s'accorder sur l'algorithme de chiffrement à utiliser. C'est seulement après ce *handshake* que la box et le serveur vont pouvoir échanger des données dans ce nouveau canal de communication chiffré.

C2 Notre analyse étant en boîte grise, nous ne pouvons pas connaître exactement le fonctionnement de la box domotique, ni le contenu des données échangées entre cette dernière et le serveur de son fabricant.

La quantité d'information qu'un attaquant peut extraire d'un canal de communication sécurisé est limitée. En effet, les données échangées étant chiffrées, elles ne sont plus interprétables. C'est pourquoi, notre approche doit résoudre les challenges suivants :

D1 Parmi les attributs extractibles des communications box domotique-serveur ou serveur-box domotique, nous avons vu dans la section 1.3 du chapitre 1, que des attributs temporels comme le temps d'arrivé entre deux paquets ou le temps entre le premier et dernier paquet échangés pouvaient être utilisés pour identifier une action effectuée par un objet IdO ou l'objet lui-même. Dans notre cas, la box peut recevoir des actions pour plusieurs objets en même temps. Il n'est pas nécessairement possible de déduire, à partir des attributs temporels d'une requête, le nombre d'objets IdO concernés par cette dernière. C'est pourquoi, les attributs utilisables avec notre approche sont restreints aux tailles des données chiffrées et au nombre de sessions TLS ouvertes observées dans les requêtes serveur-box ou box-serveur.

D2 Déjà évoqué dans la difficulté **D1**, la box domotique peut recevoir des actions pour plusieurs objets IdO en même temps. En effet, même si ces objets IdO sont hétérogènes de part leur fonctionnalité, leur marque ou leur protocole de communication, ces différences ne seront pas perceptibles sur les communications entre la box domotique et son serveur étant donné que ces dernières sont effectuées via les mêmes protocoles réseau, notamment IP, TCP ou TLS. Ainsi, déterminer exactement le nombre ou les modèles des objets IdO associés à la box domotique est complexe, voire impossible. Cependant, certaines fonctionnalités peuvent être associées à un type d'objet IdO en particulier. En inférant les actions utilisateur, notre approche peut déduire les types de certains objets IdO mais également borner le nombre de ces objets associés à la box domotique.

D3 Une box domotique peut interagir avec un ou plusieurs objets IdO pouvant délivrer une seule ou plusieurs fonctionnalités, c'est-à-dire actions. Lorsqu'un utilisateur effectue une action sur un objet IdO, la box domotique recevra la requête associée à ladite action puis la retransmettra à l'objet en question. Ainsi, une approche naïve permettant de déduire les actions demandées par l'utilisateur consisterait à effectuer toutes les combinaisons d'actions possibles sur les objets IdO d'un environnement domotique puis étudier les attributs présents dans les requêtes reçues par la box. Cette approche ne passe pas à l'échelle. C'est pourquoi notre approche consiste plutôt à utiliser un ensemble réduit de connaissances sur les relations entre les actions et leurs requêtes associées.

Compte tenu des contraintes et des challenges précédemment listés, nous formulons les hypothèses suivantes vis-à-vis du fonctionnement de la box domotique et de son serveur :

H1 La taille des données envoyées par le serveur vers la box domotique lors de l'étape **3**) dans la figure 4.2, est liée à son contenu, c'est-à-dire les actions. La taille est spécifiquement proportionnelle au nombre d'actions.

H2 Lorsqu'un utilisateur effectue $m > 1$ fois la même action sur un objet IdO alors, les m requêtes reçues par la box domotique auront des tailles des données similaires.

H3 La box domotique et l'utilisateur pouvant interagir avec un ou plusieurs objets IdO, nous supposons que lorsqu'un utilisateur souhaite effectuer une action a sur $n > 1$ objets IdO alors, la box domotique recevra une seule requête du serveur contenant n couples $< a_i,\ objet_i >$ avec $i \in [1, n]$ et a_i l'action à effectuer sur l'objet $objet_i$.

H4 La taille de la requête, notée $|requete|$, serveur-box suit une relation linéaire avec les couples $< a_i, objet_i >$ qu'elle contient. Par exemple, $|requete| = 1\times < a_1,\ objet_1 > +d$, contient l'action a_1 à effectuer sur l'objet $objet_1$ avec d pouvant être une entête contenant, l'heure de la requête ou des informations associées à l'utilisateur.

H5 Nous supposons qu'il existe une similarité, dans leur structure, entre les données échangées par l'utilisateur, via l'application mobile ou un service web, et le serveur, puis entre le serveur et la box domotique. En formalisant cette hypothèse, l'utilisateur va 1) sélectionner l'action *allumer* sur les objets IdO $prise_2$ et $ampoule_1$ sous la forme

$$requete = \{allumer(prise_2), allumer(ampoule_1), d\} \tag{4.1}$$

Une fois la requête *requete* reçue, 2) le serveur va interpréter cette dernière puis 3) en générer une nouvelle, notée $requete'$ où nous supposons que les informations présentes dans la requête initiale *requete* sont transformées par une fonction inconnue g comme formulé dans l'équation 4.2.

$$requete' = \{g(allumer(prise_2)), g(allumer(ampoule_1)), g(d)\} \tag{4.2}$$

La box domotique interprétera ensuite $requete'$ et, 4) transmettra l'action *allumer* à la $prise_2$ et l'$ampoule_1$.

H6 Notre méthode requiert d'effectuer des actions utilisateur ainsi, nous supposons qu'un attaquant possède une box domotique ainsi que des paires d'objets IdO associables avec la box domotique.

Malgré les contraintes évoquées ci-dessus, notre approche a pour objectif d'inférer les actions demandées par un utilisateur à partir des communications serveur-box domotique, comme la requête $requete'$ visible à l'étape **3**) de la figure 4.2. Pour ce faire, nous présentons dans la section 4.3 notre méthode permettant d'analyser $requete'$.

4.3 Analyse du trafic réseau de la box domotique

Dans cette section, nous détaillons les différentes étapes définies dans notre approche pour dériver, à partir des requêtes reçues par la box domotique à l'étape **3**), les actions demandées par un utilisateur sur ses objets IdO. Pour ce faire, notre démarche se compose de trois étapes :

1. Étude des relations entre les données utilisateur-serveur et serveur-box, c'est-à-dire les données envoyées par l'utilisateur, via une application mobile ou un service web, vers le serveur contrôlant la box domotique et, les données transmises par ce dernier vers la box domotique ;
2. Formalisation des relations déduites à l'étape 1,
3. Résolution de ces équations.

Chaque étape est ensuite développée dans les sections ci-dessous. Néanmoins, nous ne rentrerons pas dans les détails sur la manière dont un attaquant peut intercepter une requête traversant l'Internet, ni comment ce dernier est capable d'identifier cette requête comme étant destinée à une box domotique. À titre d'exemple, les requêtes DNS effectuées par la box domotique pourraient être utilisées pour l'identifier.

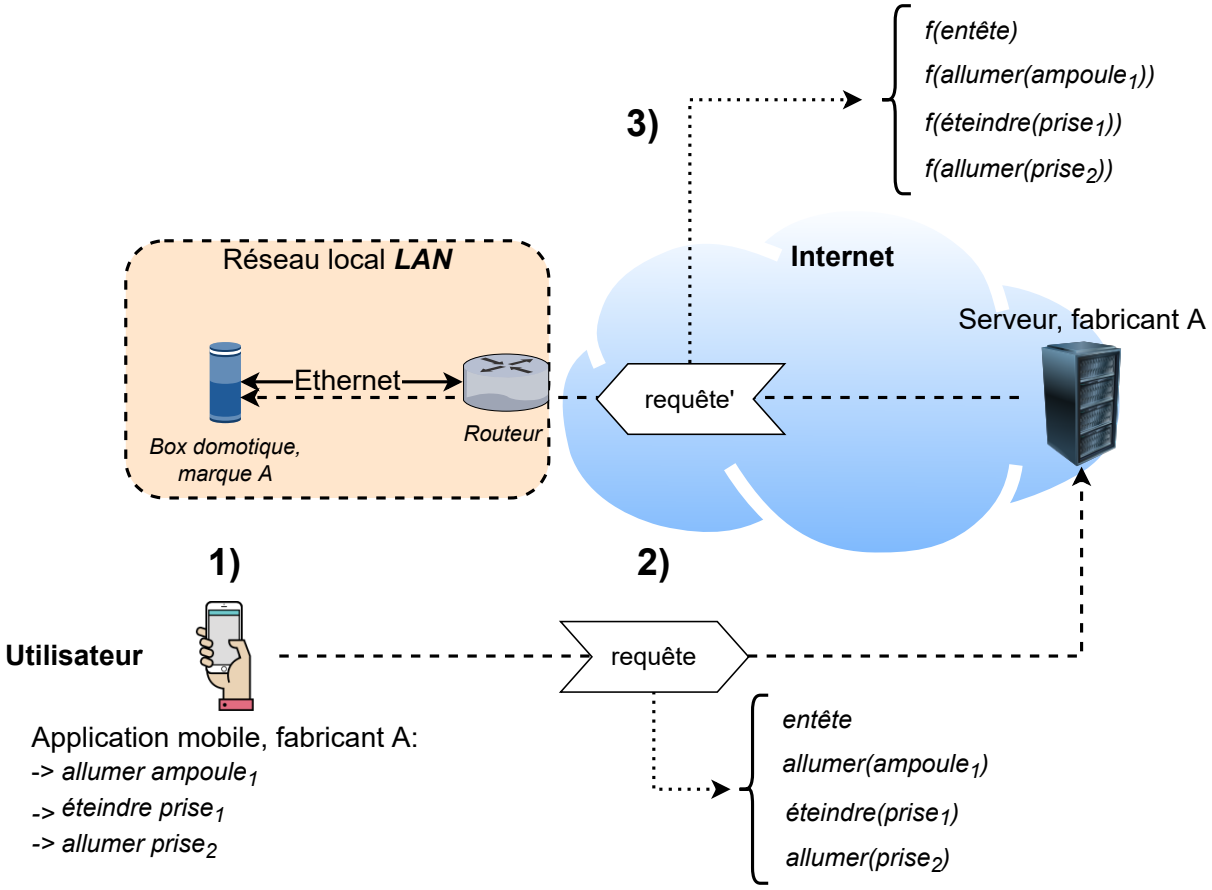


FIGURE 4.3 – Évolution d'une requête contenant les données pour une exécution de trois actions vers trois objets IdO

4.3.1 Étude des relations entre les données utilisateur-serveur et serveur-box

L'utilisation du chiffrement des données serveur-box (**C1**) ainsi que la méconnaissance du fonctionnement respectif du serveur et de la box domotique (**C2**) rendent l'analyse des données serveur-box plus complexe. Cependant, même en cas de chiffrement, la taille des données chiffrées peut être facilement extraite des paquets réseau, et nous supposons que leurs tailles ne varient peu ou pas si la même action est effectuée à plusieurs reprises (**H1** et **H2**). De plus, dans le cas où un utilisateur ordonne plusieurs actions sur ses objets IdO alors, une seule requête contenant les données associées aux actions sera transmise du serveur à la box domotique (**H3**). Ces données sont dérivables de celles initialement envoyées par l'utilisateur au serveur lors de l'étape **2**) de la figure 4.3 (**H5**).

À partir de nos hypothèses, nous proposons d'étudier la relation entre les données utilisateur-serveur et serveur-box domotique en interagissant tout d'abord avec la box domotique de manière légitime, c'est-à-dire ordonner des actions types aux objets IdO associés à la box, mais aussi de manière illégitime en modifiant les paramètres des requêtes avec des valeurs non standards comme par exemple, des actions ou des identifiants d'objets IdO vides.

Cette approche est réalisable, car le serveur avec lequel interagissent les utilisateurs est généralement une interface de programmation d'applications [33], ou *Application Programming Interface*

33. Voir `https://fr.wikipedia.org/wiki/Interface_de_programmation`, dernier accès le 24/01/2022

(API) en anglais, suivant les contraintes du style d'architecture *Representational State Transfer (REST)*[34]. Cette API REST permet au serveur de fournir des services aux utilisateurs, comme requêter des actions sur des objets IdO ou connaître l'état des objets IdO (par exemple ampoule numéro 1 allumée). Ainsi, l'utilisateur et le serveur vont devoir échanger des ressources dont le format des données peut être, par exemple, *eXtensible Markup Language (XML)*, *Yet Another Markup Language (YAML)* ou *JavaScript Object Notation (JSON)*. Ces formats de données étant libres, il est donc possible d'interpréter leur contenu. Cependant, compte tenu que les communications utilisateur-serveur sont probablement chiffrées, éditer les ressources échangées requiert d'accéder à ces dernières avant leur chiffrement ce qui est généralement faisable depuis un navigateur internet moderne.

Ainsi, nous proposons la méthode ci-après pour étudier la relation entre les actions effectuées par un utilisateur et la taille des données chiffrées observées à l'étape **3**) de la figure 4.3. Dans le reste de ce chapitre, nous utiliserons les expressions taille des données et taille des données chiffrées indistinctement. Supposons que l'on souhaite déduire la taille des données correspondantes à n actions $a_1, a_2, ..., a_n$, notre approche effectue les étapes suivantes :

e1 à l'aide de l'application mobile ou de l'interface web associée à la box domotique, nous effectuons une action a_i sur un seul objet IdO,

e2 nous récupérons les données correspondantes à cette première requête,

e3 nous extrayons la taille des données s_i de la requête serveur-box domotique correspondante.

e4 Nous répétons ensuite l'étape **e1** en utilisant deux objets IdO.

Une fois ces quatre étapes effectuées, nous obtenons les trois équations ci-dessous :

$$s_i = 1 \times a_i + d \qquad s_{2i} = 2 \times a_i + d \qquad |a_i| = s_{2i} - s_i \tag{4.3}$$

La taille d'une action a_i, notée $|a_i|$ est donc dérivée comme illustré par l'équation 4.3. Comme évoqué dans **H4** et **H5**, nous supposons qu'il existe dans la structure de données envoyée à l'étape **e2**, un élément d pouvant contenir des informations sur les objets IdO ou l'utilisateur. D'ailleurs, la présence de d ainsi que la relation linéaire entre les actions présentes dans les requêtes seront vérifiées expérimentalement. Enfin, à partir de $|a_i|$, la taille de d peut être calculée via l'équation $|d| = s_i - 1 \times |a_i|$.

Dans le cas où d contiendrait des informations sur l'utilisateur, il est alors nécessaire de répéter l'étape **e1**, mais en changeant le contenu de d afin de vérifier si ces modifications sont perceptibles sur la taille des données s_i observées durant l'étape **e3**. Si tel est le cas, notre méthode n'est pas implémentable.

Une fois les tailles des n actions calculées, nous formalisons dans la section suivante, les requêtes reçues par la box domotique.

4.3.2 Modélisation de la requête serveur-box domotique

À partir de la taille des métadonnées $|d|$, des n actions a_i, notée $|a_i|$, ainsi que du nombre d'occurrences nb_a_i d'une action a_i, nous pouvons calculer la *taille des données estimée* d'une requête serveur-box domotique s à l'aide de l'équation 4.4.

$$s = |d| + \sum_{i=1}^{n} |a_i| \times nb\_a_i \tag{4.4}$$

34. Voir `https://www.redhat.com/fr/topics/api/what-is-a-rest-api`, dernier accès le 24/01/2022

Nous utilisons le terme *taille des données estimée* car, comme énoncé dans l'hypothèse **H2**, il est possible que dans certains cas, notamment les requêtes composées de $m > 3$ actions, la taille des données réellement extraite de la requête envoyée par le serveur vers la box domotique diffère de la *taille des données estimée* à cause, par exemple, de l'algorithme de chiffrement utilisé. En effet, même si le chiffrement ne modifie pas le contenu des données, la taille de ces dernières peut varier de quelques octets lorsque l'algorithme de chiffrement est par bloc et utilise le remplissage, ou *padding* en anglais [34]. Par conséquent, si nous voulons correctement déduire les actions utilisateur, il est nécessaire d'inférer la différence ϵ attendue entre la taille des données observée, notée so, et la *taille des données estimée*, notée s, calculée par l'équation 4.4.

Étant donné que cette différence ϵ peut être calculée à partir des tailles des données so et s telle que $\epsilon = s - so$, nous proposons donc de construire un ensemble de données où pour un couple $< s, so >$, la différence ϵ est calculée. Par conséquent, nous devons générer de nombreuses valeurs de so et s, c'est-à-dire requêter une ou plusieurs actions aux objets IdO associés à la box domotique. Dans le reste de ce chapitre, nous utiliserons l'expression combinaison d'actions pour qualifier les requêtes décrites ci-avant. En considérant que m objets IdO sont associés à la box domotique étudiée, nous générons aléatoirement j combinaisons d'actions c_i (avec $j >= m$ et $i \in [1, j]$) constituées de n actions ($n \in [1, m]$). Ces j combinaisons sont ensuite transmises au serveur suivant le format de données attendu par ce dernier comme discuté dans la section 4.3.1. Une fois ces j combinaisons d'actions envoyées, nous interceptons les j requêtes correspondantes et reçues par la box domotique pour y récupérer les tailles des données correspondantes. Enfin, pour chaque c_i avec $i \in [1, j]$, nous calculons à l'aide de l'équation 4.4 sa *taille des données estimée* s_i et la valeur de sa taille des données observée so_i puis calculons sa différence ϵ_i.

À partir de cet ensemble de données composé de j tuples $(c_i, s_i, so_i, \epsilon_i)$ avec $i \in [1, j]$, il est possible d'utiliser un algorithme d'apprentissage automatique, noté A, afin d'inférer la différence ϵ à partir de la taille observée so tel que $A(so) = \epsilon$. Le choix cet algorithme d'apprentissage dépend du volume de données disponible et la distribution des différences ϵ, et sera discuté dans la section 4.4.

En raison des différences de tailles des données précédemment énoncées, il est nécessaire de modifier l'équation initiale 4.4 en y intégrant la différence attendue ϵ.

$$s - \epsilon - |d| = \sum_{i=1}^{n} |a_i| \times nb\_a_i \tag{4.5}$$

La partie gauche de l'équation pouvant être facilement calculée, et les valeurs des $|a_i|$ dérivées dans l'équation 4.3, il reste donc à trouver les valeurs des nb_a_i satisfaisant l'égalité. Comme $nb\_a_i \in \mathbb{N}$, notre problème devient similaire à celui du rendu de monnaie ou *change-making problem* [178] dont nous expliquons la méthode de résolution dans la prochaine section.

4.3.3 Résolution de l'équation

Similairement au problème du rendu de monnaie [178], résoudre l'équation 4.5 revient à trouver les tuples $(nb\_a_1, ..., nb\_a_n) \in \mathbb{N}^n$ tels que $\sum_{i=1}^{n} |a_i| \times nb\_a_i = M$ où $M = (s - \epsilon - |d|)$ avec $M \in \mathbb{N}$. Il n'est pas nécessaire de trouver les solutions $(nb\_a_1, ..., nb\_a_n)$ minimisant le nombre d'actions, c'est-à-dire minimisant $\sum_{i=1}^{n} nb\_a_i$. En effet, un utilisateur pouvant effectuer les actions qu'il souhaite sur les objets IdO de son choix, chercher à minimiser ou maximiser le nombre d'actions n'a pas de sens. C'est pourquoi, contrairement au problème du rendu de monnaie, nous cherchons tous les tuples $(nb\_a_1, ..., nb\_a_n)$ satisfaisant l'équation 4.5.

Ces derniers sont calculés par l'algorithme 1 où l'équation 4.5 est résolue par programmation dynamique. Dans cet algorithme, la ligne 4 permet d'initialiser le tableau des solutions avec des

listes vides afin de pouvoir stocker les combinaisons d'actions futures. Ensuite, nous assignons une première combinaison d'actions vide, équivalente au 0 de l'approche initiale, à la ligne 5. Enfin, au lieu de calculer $min_{1 \leq i \leq n}(M - |a_i|)$ pour chaque élément $A[i]$, la ligne 16 permet de concaténer la nouvelle action disponible aux combinaisons d'actions déjà existantes dans $A[i]$. À l'aide de ces ajustements, l'algorithme 1 retourne une liste $L = \{S_1, ..., S_m\}$ de taille m, avec $m \in \mathbb{N}$, contenant toutes les solutions $S_j = (nb\_a_1, ..., nb\_a_n)$ avec $j \in [1, m]$ satisfaisant l'équation 4.5.

Enfin, et comme discuté dans la section 4.3.2, notre approche suppose l'interception préalable d'une requête envoyée par le serveur vers la box domotique pour ensuite, déduire les actions utilisateur. C'est pourquoi, nous regroupons dans l'algorithme 2 les quatres étapes effectuées par notre méthode pour atteindre cet objectif. Parmi ces étapes, nous avons :

1. l'extraction à la ligne 4 de la taille des données observée, notée *so*, à partir d'une requête r envoyée par le serveur vers la box domotique.

2. Suivant l'algorithme de chiffrement utilisé par la box domotique et le serveur, la taille calculée par notre méthode peut être différente de *so*. C'est pourquoi, la valeur de cette différence ϵ est dérivée à partir de *so* à la ligne 8.

3. La taille observée *so* est ensuite ajustée à la ligne 11 par rapport aux valeurs d'ϵ et des métadonnées $|d|$.

4. Enfin, à la ligne 15, nous calculons toutes les combinaisons d'actions pouvant correspondre à cette nouvelle valeur de *so* à l'aide de l'algorithme 1 décrit ci-avant.

Notre méthode permet, en plus d'obtenir les combinaisons d'actions effectuées par un utilisateur, de déduire d'autres informations sur la composition des objets IdO associés une box domotique. Cependant, il n'est pas toujours possible de confirmer l'exacte combinaison d'actions effectuée par un utilisateur à un instant donné. Ainsi, nous discutons des avantages et inconvénients de notre approche dans la section 4.3.4.

Algorithme 1 Résolution par programmation dynamique de l'équation 4.5

fonction RESOUDRE($M = (s - \epsilon - |d|), actions = [|a_1|, ..., |a_n|]$)
  $actions \leftarrow$ TrierParOrdreCroissant($actions$)
  $nb\_actions \leftarrow$ RecupereNbElements($actions$) // Retourne le nombre d'actions
  $solutions \leftarrow [[\,]_0, ..., [\,]_M]$ // Créé un tableau contenant $M + 1$ listes vides
  $solutions[0] \leftarrow solutions[0] \bigcup [\,]$ // Initialise une 1re solution vide à l'indice 0 du tableau

  // Initialise un 1^{er} indice pour parcourir la liste des actions
  $i \leftarrow 1$
  tant que $i \leq nb\_actions$ **faire**
    // Initialise un 2^{nd} indice pour parcourir le tableau des solutions
    $j \leftarrow actions[i]$
    tant que $j \leq M$ **faire**
      pour tout $solution \in solutions[j - actions[i]]$ **faire**
        /* Ajoute l'action $actions[i]$
          aux solutions présentes à l'indice $j - actions[i]$ */
        $solution \leftarrow solution \bigcup actions[i]$
      fin pour
      // Incrémente j pour parcourir le tableau des solutions
      $j \leftarrow j + 1$
    fin tant que
    // Incrémente i pour sélectionner l'action suivante
    $i \leftarrow i + 1$
  fin tant que
  /* Retourne la liste des solutions à l'indice M,
    c'est-à-dire les solutions satisfaisant $M = \sum_{i=1}^{nb\_actions} nb\_a_i \times a_i$ */
  retourne $solutions[M]$
fin fonction

Algorithme 2 Infère une liste de combinaisons d'actions à partir d'une requête serveur-box notée r, de la liste des tailles d'actions $ta = [|a_1|, ..., |a_n|]$, et de la taille, constante, des métadonnées $m = |d|$

```
1: fonction InfereCombinaisonsDActions(r, ta = [|a_1|, ..., |a_n|], m = |d|)
2:     /* À partir de la requête r,
3:        on extrait la taille des données*/
4:     so ← ExtraitTailleDonnee(r)
5:
6:     /* Comme discuté dans 4.3.2,
7:        on va prédire la différence attendue ϵ à partir de so. */
8:     ϵ ← PredireDifference(so)
9:
10:    // On soustrait à so les valeurs de m et ϵ
11:    so ← so − m − ϵ
12:
13:    /* À partir de cette nouvelle valeur de so, on peut appeler l'algorithme 1
14:       pour obtenir les combinaisons d'actions pouvant être contenues à so */
15:    actions ← Resoudre(so, ta)
16:
17:    // Enfin, on peut retourner la liste des combinaisons d'actions
18:    retourne actions
19: fin fonction
```

4.3.4 Commentaires sur notre méthode de résolution

Dans cette section, nous discutons des avantages et inconvénients de notre approche par rapport aux méthodes proposées dans la section 1.3 du chapitre 1 de l'état de l'art.

4.3.4.1 Avantages

Notre méthode décompose une taille des données observée so en une liste de $m \geq 1$ combinaisons d'actions. Par conséquent, cette méthode de résolution peut fonctionner sur n'importe quelle taille des données observée so' même si so' n'a jamais été interceptée auparavant. Une méthode basée uniquement sur un algorithme d'apprentissage automatique supervisé A aurait plus de difficulté que notre approche. En effet, pour que A puisse prédire une combinaison d'actions tel que $A(so) = (nb\_a_1, ..., nb\_a_n)$, il est nécessaire, durant le processus d'apprentissage, de fournir à ce dernier autant de couples $< so_i, S_i >$ avec $S_i = (nb\_a_1, ..., nb\_a_n)$ que possible. Or, plus le nombre d'objets IdO associés à une box domotique est grand, plus le nombre de combinaisons d'actions possibles est important. Ainsi, compte tenu de l'aléa induit par le nombre d'objets IdO pouvant être associés à une box domotique, il ne semble pas réaliste de construire un tel ensemble de données. Supposons un tel ensemble de données, il n'est pas garanti que A retourne la combinaison d'actions correcte, ni que A soit capable de dériver l'ensemble des combinaisons d'actions correspondantes. Notre méthode semble donc plus appropriée, car elle ne requiert qu'une phase d'apprentissage limitée où seul un petit ensemble de connaissances est appris puis utilisé pour décomposer n'importe quelle taille des données observée so" en un ensemble de combinaisons d'actions possible (**D1** et **D3**).

Comme évoqué dans la difficulté **D2**, les actions peuvent être utilisées pour déduire le type des objets IdO associés à une box domotique, par exemple une action assignant une intensité

lumineuse est associée à une ampoule connectée. Supposons qu'un objet IdO de type T soit associé aux actions a_p avec $p \in [1, n]$, et que notre méthode retourne une liste de combinaisons d'actions L composée d'une ou plusieurs combinaisons d'actions $S_j = (nb\_a_1, ..., nb\_a_n)$ avec $S_j \in L$. Alors, si toutes les $S_j \in L$ contiennent nb_a_p fois les actions a_p, nous pouvons déduire qu'au moins nb_a_p objets IdO de type T sont associés à la box domotique. Cependant, nous ne pouvons pas conclure qu'exactement nb_a_p objets de type T sont associés à la box. D'ailleurs cette remarque s'applique également pour le nombre d'objets IdO. De manière générale, notre méthode permet de borner le nombre ou le type des objets IdO associés à la box domotique étudiée.

4.3.4.2 Inconvénients

Comme discuté dans la section 4.3.2, il peut être nécessaire de soustraire une différence ϵ à la taille des données observée *so*. Cette dernière est prédite uniquement à partir de *so*. Les performances de notre méthode vont donc dépendre de la prédiction de ϵ. Par conséquent, l'algorithme d'apprentissage automatique supervisé sélectionné doit être le plus performant possible afin de rendre les performances globales de notre approche optimales. Le choix de ce dernier dépend donc du volume de données disponibles et de la distribution des différences.

Suivant la box domotique étudiée, il est possible que des combinaisons d'actions différentes soient transmises par des requêtes serveur-box domotique avec des tailles des données observées identiques, notée so_i. Notre méthode de résolution décompose so_i en une liste de $m \geq 1$ combinaisons d'actions $L = \{S_1, ..., S_m\}$ où $S_j = (nb\_a_1, ..., nb\_a_n)$ et $j \in [1, m]$. Ainsi, il ne sera pas possible d'affirmer quelle est la combinaison d'actions réellement demandée par l'utilisateur parmi toutes les possibilités.

4.4 Expérimentation et résultats

Dans cette section, nous détaillons l'environnement domotique étudié, c'est-à-dire la box domotique et ses objets IdO associés, puis discutons de la méthodologie employée pour mettre en place l'approche décrite dans la section 4.3 sur cet environnement IdO. Et enfin, nous présentons les performances de notre méthode sur la déduction des actions demandées par un utilisateur.

4.4.1 Environnement domotique étudié

Nous nous intéressons au cas d'une *maison intelligente* à l'image de celle illustrée dans la figure 4.4 où une box domotique, notée B, permet de contrôler un total de 16 objets IdO, dont 12 prises et 4 ampoules connectées. De plus, tous appartiennent à la même marque française que l'on nommera M. Enfin, comme énuméré dans le tableau 4.1, selon l'objet IdO sélectionné, un utilisateur peut demander l'une des actions suivantes :

- **On** : l'objet IdO passe en état de marche.
- **On {1, ..., 10} minutes** : effectue l'action **On** pendant une période de temps limitée entre {1, ..., 10} minutes.
- **Off** : l'objet IdO devient hors service en attendant une action future.
- **P** : cette action permet d'allumer l'ampoule connectée avec une intensité préconfigurée par l'utilisateur.
- **P {1, ..., 10} minutes** : effectue l'action **P** pendant une période de temps limitée entre {1, ..., 10} minutes.

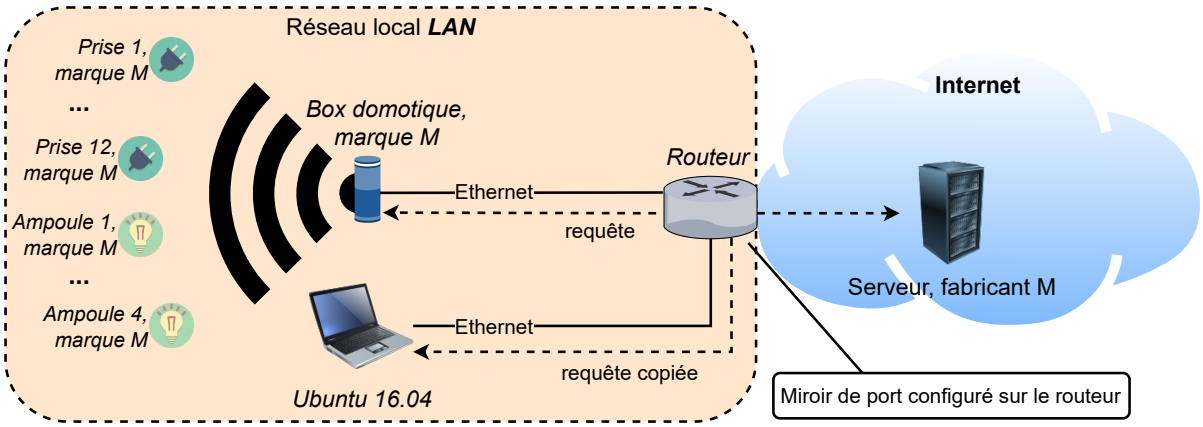


FIGURE 4.4 – Environnement domotique étudié

Objet IdO	Actions
Box domotique	∅
Prise connectée	On, Off, On {1, ..., 10} minutes
Ampoule connectée	On, Off, On {1, ..., 10} minutes, P, P{1, ..., 10} minutes

Tableau 4.1 – Répartition des actions disponibles par objet IdO

Par le biais d'une application mobile et/ou un service web, M propose les deux méthodes décrites ci-après pour interagir avec la box domotique et donc transmettre des actions aux objets IdO associés :

— la méthode **type spécifique** va permettre d'exécuter la même action sur un ou plusieurs objets IdO de même type, par exemple exécuter l'action **On** sur les 12 prises connectées.

— La méthode **scénario** autorise l'exécution d'actions différentes sur n'importe quels objets IdO en deux étapes. L'utilisateur créé d'abord un **scénario** en lui assignant un nom et, évidemment, les actions à effectuer sur les objets IdO. Après sa création, il sera possible de l'exécuter.

Comme expliqué dans la section 4.2.2, notre méthode nécessite de connaître la structure des données échangées lors des requêtes utilisateur-serveur. Pour ce faire, nous utilisons le service web, noté SW, proposé par M à l'aide du navigateur web Mozilla Firefox [35] version 64.00 installé sur une machine Ubuntu [36] version 16.04 *Xenial Xerus*. En effet, ce navigateur inclut l'outil *moniteur réseau* qui permet d'observer le contenu des requêtes échangées depuis le navigateur et ce, avant leurs chiffrements.

Comme illustré dans la figure 4.4, nous interceptons les requêtes échangées entre la box domotique et le serveur à l'aide d'un miroir de port [37], configuré sur le routeur de notre réseau privé. Comme son nom l'indique, cette fonctionnalité permet de copier le trafic réseau traversant un port du routeur vers un autre port où nous branchons notre ordinateur afin d'effectuer une écoute passive du trafic réseau de la box domotique.

35. Disponible sur `https://www.mozilla.org/fr/firefox/`, dernier accès le 30/01/2022
36. Disponible sur `https://ubuntu.com/`, dernier accès le 30/01/2022
37. Voir `https://fr.wikipedia.org/wiki/Miroir_de_port`, dernier accès le 30/01/2022

4.4.2 Rétro-ingénierie de protocole

Dans cette section, nous détaillons les étapes effectuées pour analyser le fonctionnement de la box domotique B et du serveur S. En effet, avant d'appliquer notre méthode, il est nécessaire de s'assurer que la box domotique et le serveur vérifient les hypothèses présentées dans la section 4.2.2.

Nous possédons les objets IdO associés à une box domotique et sommes capable d'effectuer des actions sur ces objets (**H6**). Ensuite, à l'aide du service web SW, nous requêtons des combinaisons aléatoires d'actions via les méthodes **type spécifique** et **scénario**, puis avons étudié le trafic réseau entre B et S, et observé les comportements suivants :

C1 B et S échangent continuellement des paquets UDP contenant des données non chiffrées composées d'un mot-clé humainement compréhensible mais suivi d'une chaîne de caractères non interprétable. Selon notre analyse, ces paquets UDP remplissent deux fonctions :

(a) vérifier l'état de la connectivité entre B et S grâce aux mots-clés *PING* et *PONG*, et

(b) forcer B à initier une session sécurisée avec S si le paquet UDP reçu par B contient le mot-clé *OPEN*.

Cette dernière fonctionnalité est observée peu de temps après qu'un utilisateur ait demandé à exécuter une combinaison d'actions en utilisant la méthode **type spécifique** ou **scénario**.

C2 C'est seulement après avoir reçu le paquet UDP avec le mot-clé *OPEN* que B va initier par le biais du protocole TLS en version 1.2[56] une session sécurisée, c'est-à-dire chiffrée, avec S. Parmi les paramètres associés à cette session, nous avons remarqué que B et S s'accordent toujours à employer la suite cryptographique, ou *ciphersuite* en anglais, `TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256`, c'est-à-dire chiffrer les données à l'aide de l'algorithme de chiffrement par bloc *Advanced Encryption Standard (AES)* avec le mode *Galois/Counter Mode (GCM)*. Ce dernier va itérativement découper les données par bloc de 128 bits puis les chiffrer avec AES. Qui plus est, cet algorithme est préconisé pour sa sûreté par les normes de cybersécurité IEEE 802.1AE[38] ou NIST SP 800-38D[39]. Une fois cette session TLS établie, S envoie à B les données associées à la combinaison d'actions demandée par l'utilisateur puis clôt la session.

C3 Peu de temps après, et sans recevoir de paquets UDP avec le mot-clé *OPEN*, B va initier nb_s sessions TLS 1.2 avec S.

Tout d'abord, nous remarquons que, peu importe le nombre d'objets IdO concernés par une action, le serveur S envoie une seule requête r à B, ce qui nous permet de valider l'hypothèse **H3**. De plus, la taille des données de cette dernière dépend de l'action demandée ou du nombre d'actions, c'est-à-dire objets IdO (**H1**), mais ne varie pas lorsqu'une même action, ou combinaison d'actions, est effectuée plusieurs fois (**H2**). Enfin, à l'aide de l'outil *moniteur réseau*, nous avons observé que la structure des données envoyées par l'utilisateur au serveur contient des métadonnées, ainsi que la liste des couples $< a,\ objet >$ avec a l'action à effectuer sur l'objet *objet* (**H4**). La modification du contenu des métadonnées ou des informations de l'utilisateur n'entraînent pas de changement dans les requêtes envoyées de S à B.

Nous remarquons également que la box domotique B interagit avec les objets IdO à l'aide d'un protocole de communication sans-fil propriétaire à M. Par conséquent, un attaquant capable d'intercepter ces communications devra également effectuer de la rétro-ingénierie sur ce dernier tout en devant être à proximité de B. D'un autre côté, notre approche permet de déduire les actions utilisateur directement depuis l'Internet.

38. Voir `https://1.ieee802.org/security/802-1ae/`, dernier accès le 31/01/2022
39. Voir `https://csrc.nist.gov/publications/detail/sp/800-38d/final`, dernier accès le 31/01/2022

4.4.2.1 Mesure du nombre d'objets IdO à partir des sessions TLS entre la box domotique et le serveur

Un utilisateur peut obtenir, par le biais de SW, plusieurs informations sur son environnement domotique comme :

1. l'état de la connectivité entre sa box domotique B et le serveur S. Pour ce faire, nous supposons que S mesure cette valeur à partir des paquets UDP contenant les mots-clés *PING* et *PONG* échangées entre B et S, comme abordé lors de l'étape **C1** de la section 4.4.2.
2. L'état de chacun des objets IdO associé à B, c'est-à-dire connaître l'action actuellement exécutée par chacun d'entre eux. Après avoir déclenché une combinaison d'actions C telle que $C = \{nb\_a_1, nb\_a_2, ..., nb\_a_7\}$ avec $\sum_{i=1}^{i\leq 7} nb\_a_i = nb\_IoT$, et nb_a_i le nombre de répétition de l'action a_i où $a_i \in \{On, Off, ..., P\}$; nous remarquons à l'étape **C3** de la section 4.4.2 que B va initier nb_s sessions TLS 1.2 consécutives avec S où $nb_s = nb\_IoT+1$. Les données échangées étant chiffrées, nous supposons que ces nb_s sessions sont utilisées par B pour informer S du déroulement de C et l'état des nb_IoT objets IdO suite aux actions demandées. De plus, l'envoi de ces nb_s sessions s'effectue de manière très rapide par B, c'est-à-dire que le temps entre les deux premiers paquets *Client Hello* de deux sessions TLS 1.2 consécutives est court.

Ainsi, l'observation décrite à l'étape 2 nous permet de déduire l'équation 4.6 où le nombre d'objets IdO nb_IoT associé à une combinaison d'actions C est dérivé du nombre de sessions TLS 1.2 nb_s initié par B.

$$nb\_IoT = nb_s - 1 \tag{4.6}$$

Les sessions TLS 1.2 initiées par B sont comptabilisées uniquement après que B ait reçu de S la requête contenant la combinaison d'actions C, et si le temps inter-sessions est inférieur à 2.5 secondes, c'est-à-dire le temps entre deux paquets *Client Hello* de deux sessions TLS 1.2 consécutives.

Enfin, c'est seulement après avoir décomposé une taille de données observées so en une liste L de $m \geq 1$ combinaisons d'actions telle que $L = \{C_1, ..., C_m\}$ avec $C_i = \{nb\_a_1, nb\_a_2, ..., nb\_a_7\}$ où $i \in [1, m]$, que notre approche va utiliser la valeur de nb_IoT calculée par l'équation 4.6 pour retirer les combinaisons d'actions C_i ayant une cardinalité différente de nb_IoT, c'est-à-dire $\sum_{j=1}^{j\leq 7} nb\_a_j \neq nb\_IoT$.

Dans la suite de cette section, nous utiliserons la notation L' pour qualifier la liste des combinaisons d'actions où chaque combinaison d'actions C_j a une cardinalité égale à nb_IoT.

4.4.3 Application de notre approche sur la box domotique

À l'aide de notre approche détaillée dans la section 4.3, la taille de chacune des actions listée dans le tableau 4.1 est dérivée dans le tableau 4.2 où nous indiquons également les types des objets IdO associés à chaque action.

Comme abordé en section 4.3.1, il existe dans les requêtes transmises de S à B, en plus des données liées aux actions à effectuer, des métadonnées d dont notre approche estime la taille à 216 octets. Ainsi, une requête so de taille des données observée $|so|$ envoyée de S à B peut être modélisée par l'équation 4.7 où nb_a_i correspond au nombre de répétitions de l'action a_i avec $a_i \in \{On, Off, ..., P\}$.

$$\begin{aligned}|s_o| = \epsilon + 216 + 221 \times nb\_a_1 + 238 \times nb\_a_2 + 240 \times nb\_a_3 + 489 \times nb\_a_4 + 490 \times nb\_a_5 \\ +491 \times nb\_a_6 + 492 \times nb\_a_7\end{aligned} \tag{4.7}$$

Objet(s) IdO	Actions	Taille (octets)
Ampoule connectée	P	$\|a_1\| = 221$
Prise et ampoule connectées	On	$\|a_2\| = 238$
Prise et ampoule connectées	Off	$\|a_3\| = 240$
Prise et ampoule connectées	On 1 minute	$\|a_4\| = 489$
Prise et ampoule connectées	On {2, ..., 10} minutes	$\|a_5\| = 490$
Ampoule connectée	P 1 minute	$\|a_6\| = 491$
Ampoule connectée	P {2, ..., 10} minutes	$\|a_7\| = 492$

Tableau 4.2 – Répartition de la taille des actions par objet IdO

Dans le reste de cette section, nous utiliserons la notation a_i où $i \in [1, 7]$ pour faire référence aux actions listées dans le tableau 4.2.

4.4.4 Déduction des actions utilisateur

Afin de résoudre l'équation 4.7, il est d'abord nécessaire de prédire la valeur ϵ correspondante à la différence attendue entre la *taille des données estimée* $|s|$ et la taille des données observée $|so|$ extraite de la requête envoyée par S à B, notée so, contenant les informations sur la combinaison d'actions à exécuter.

Ainsi, et comme expliqué dans la section 4.3.2, nous avons construit un ensemble de données D composé de 307 tuples $(C_i, |s_i|, |so_i|, \epsilon_i, nb\_IoT_i)$ avec $C_i = \{nb\_a_1, nb\_a_2, ..., nb\_a_7\}$ et $i \in [1, 307]$. Pour ce faire, nous avons itéré sur le nombre d'objets IdO de un à 16, c'est-à-dire $\sum_{i=1}^{i\leq 7} nb\_a_i$, et généré à chaque itération 20 combinaisons d'actions aléatoires. Cependant, seules 7 actions sont possibles avec un objet IdO, c'est pourquoi nous obtenons seulement 307 combinaisons au lieu de 320. La répartition moyenne des prises et ampoules dans ces combinaisons d'actions est illustrée dans la figure 4.5

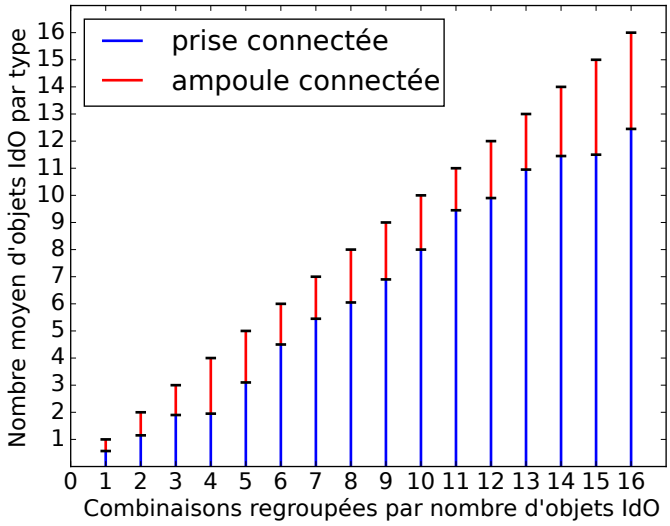


FIGURE 4.5 – Répartition des prises et ampoules connectées dans nos 307 combinaisons d'actions

Enfin, nous utilisons les données présentes dans D pour tester notre approche et prédire les

valeurs d'ϵ à l'aide de l'algorithme des k plus proches voisins (kNN) paramétré avec $k = 2$.

4.4.4.1 Métriques d'évaluation

Nous introduisons les métriques suivantes afin d'évaluer les résultats obtenus par notre approche :

- *Pred_ devices* représente le ratio des tests ayant une prédiction de nb_IoT qui correspond effectivement au nombre d'objets IdO présents dans la combinaison d'actions réellement effectuée $C_i = \{nb\_a_{1i}, nb\_a_{2i}, ..., nb\_a_{7i}\}$, c'est-à-dire satisfaisant l'équation $nb\_IoT_i = \sum_{j=1}^{j\leq 7} nb\_aji$.
- $P(C_i \in L)$ ou $P(C_i \in L')$ indique simplement si la combinaison d'actions réellement effectuée est présente dans la liste des combinaisons d'actions L retournée par notre approche ou, dans sa version réduite L'.
- $card(L)$ ou $card(L')$ retourne la taille respective de la liste des combinaisons d'actions L ou L'. En effet, même si dans la métrique précédente nous comparions leurs contenus, il est également pertinent de comparer leurs tailles afin de confirmer ou non la déduction de la valeur de nb_IoT. Plus la taille de L ou L' est faible plus la certitude de retourner la bonne combinaison d'actions est grande.
- $P(C_i \in L'|Pred\_devices)$ correspond à la probabilité de trouver la vraie combinaison d'actions effectuée C_i dans la liste réduite L' sachant que la prédiction de nb_IoT_i est correcte.

De plus, étant donné que plusieurs combinaisons d'actions peuvent être présentes dans les listes L et L', nous introduisons la métrique définie par la formule 4.8 permettant de mesurer la similarité entre deux combinaisons d'actions $p = \{p_0, .., p_n\}$ et $q = \{q_0, .., q_n\}$. Cette métrique permet de prendre en compte 1) les erreurs d'affectation pour chaque action, et 2) l'erreur globale sur le nombre d'actions entre deux combinaisons d'actions. Une distance $dist(p, q) = 0$ correspond à la valeur optimale où p et q sont identiques. Par exemple, en posant $p = \{1, 0, 1\}$ et $q = \{1, 1, 0\}$, nous obtenons une distance $dist(p, q) = 1$ correspondant à l'action mal placée à l'indice deux de q rapport à p. Dans le cas où les combinaisons n'ont pas la même taille comme $p = \{1, 1, 1\}$ et $q = \{3, 2, 1\}$, nous obtenons bien de même une distance $dist(p, q) = 3$, correspondant bien aux deux actions supplémentaires à l'indice un et à l'action supplémentaire à l'indice deux dans la combinaison q.

$$dist(p, q) = \frac{|\sum_{p_i \in p} p_i - \sum_{q_i \in q} q_i| + \sum_{i=1}^{n} |p_i - q_i|}{2} \tag{4.8}$$

Nous utiliserons cette métrique pour mesurer les similarités entre les combinaisons d'actions présentes dans les listes L et L' avec la vraie combinaison, puis nous calculerons la moyenne de ces valeurs pour chacune de ces deux listes. Ainsi, plus la moyenne de ces distances tend vers zéro plus les combinaisons d'actions retournées sont proches de la vraie combinaison d'actions effectuée.

4.4.4.2 Résultats d'expérimentation

Nous avons appliqué une validation croisée par la méthode *k-fold* avec $k = 4$ sur l'ensemble de données D et obtenu les résultats visibles dans le tableau 4.3.

Métrique	**Précision**
$P(C_i \in L)$	98.4
$P(C_i \in L')$	91.2
$Pred\_devices$	91.8
$P(C_i \in L' \mid Pred\_devices)$	99.2

Tableau 4.3 – Résultats obtenus par métrique

Tout d'abord, nous observons que notre approche est capable de trouver la combinaison d'actions réellement effectuée C_i dans la liste de combinaisons d'actions L dans 98.4% des cas. Cependant, comme illustré par la figure 4.6(a), nous observons que L possède environ cinq fois plus de combinaisons d'actions par rapport à la liste L' dont le nombre de combinaisons a été réduit par la méthode présentée en 4.4.2.1. Ainsi, même si $C_i \in L$, il est souvent difficile de déduire précisément la combinaison d'actions effectuée par l'utilisateur parmi toutes les possibilités. En effet, en moyenne nous trouvons 39 combinaisons d'actions dans L et seulement huit dans L'. Cependant, plus le nombre d'objets IdO assujettis à une action est grand, plus le nombre de combinaisons d'actions trouvées par notre approche sera grand. Par exemple, en considérant les combinaisons d'actions pour 12 objets IdO, notre approche retourne, en moyenne, 56 combinaisons dans L et seulement neuf dans L'. Qui plus est, avec 16 objets IdO, jusqu'à 465 combinaisons d'actions peuvent être présentes dans L mais «seulement» 107 dans L'.

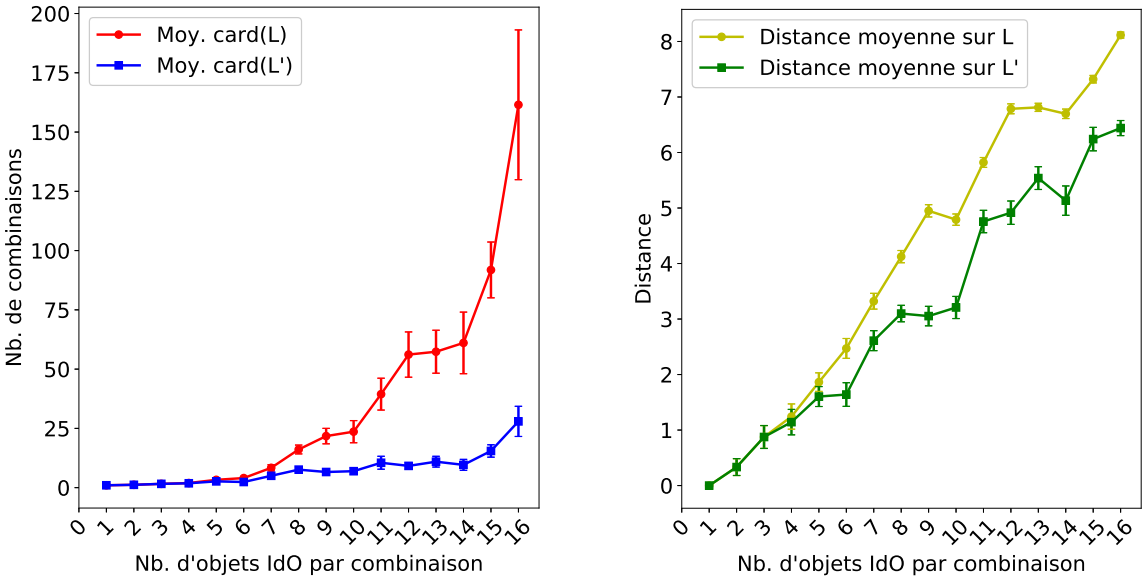
Malgré une probabilité plus faible de 91.2% de trouver $C_i \in L'$, il semble plus pertinent d'utiliser la liste L' au lieu de L par rapport au premier critère qu'est le nombre de combinaisons d'actions présentes dans la liste. Nous remarquons aussi grâce à la valeur de $P(C_i \in L' \mid Pred\_devices) = 99.2\%$ que trouver C_i dans L' est fortement lié aux performances de la métrique $Pred\_devices$ (91.8%).

Un second avantage à l'utilisation de L' au lieu de L est illustré par la figure 4.6(b) où nous mesurons les distances entre la vraie combinaison d'actions effectuée C_i et chacune des autres combinaisons présentes dans les listes L et L' en suivant la formule 4.8. Globalement, nous pouvons voir que les combinaisons d'actions présentes dans la liste L' sont plus proches de C_i que celles présentes dans L, c'est-à-dire les combinaisons d'actions dans L' ont plus d'actions communes avec C_i. Par exemple, pour les combinaisons d'actions avec 10 objets IdO, nous obtenons une distance moyenne d'environ quatre pour L et trois pour L' soit, en d'autres termes, six actions communes avec C_i pour L et sept pour L'. De manière générale, nous observons que :

— plus le nombre d'objets IdO par combinaison d'actions est conséquent, plus les distances dans L et L' sont grandes.

— En moyenne, les combinaisons d'actions dans L' ont une distance d'environ trois alors que celles dans L ont une distance d'environ quatre donc, les combinaisons d'actions dans L' sont plus proches de C_i.

En utilisant L ou L', il est difficile, voir impossible, de retourner la combinaison d'actions exacte demandée par un utilisateur. En effet, compte tenu des tailles de chaque action listées dans le tableau 4.2, plusieurs d'entre elles sont similaires, comme a_6 et a_7 ou bien a_4 et a_5. Par conséquent, il est en effet possible d'observer, pour plusieurs combinaisons d'actions, une taille des données observée identique. Ainsi, notre approche remplit parfaitement sa fonction en nous retournant effectivement toutes les combinaisons d'actions possibles et, à l'aide de la méthode 4.4.2.1, nous pouvons filtrer ces dernières par rapport au trafic réseau que génère la box domotique à l'instant observé. D'un autre côté, atteindre des performances similaires avec une méthode basée uniquement sur une analyse du trafic réseau de la box domotique (par exemple



(a) Nb. moyen de combinaisons d'actions trouvées dans les listes L et L' selon le nombre d'objets IdO concernés par une combinaison. L'intervalle de confiance est configuré à 68%.

(b) Distance moyenne des combinaisons d'actions trouvées dans les listes L et L' selon le nombre d'objets IdO concernés par une combinaison. L'intervalle de confiance est configuré à 68%.

FIGURE 4.6 – Évaluation des améliorations introduites par notre méthode de réduction

nombre de paquets échangés, délai entre deux sessions TLS 1.2) nécessiterait de déclencher plusieurs millions de combinaisons d'actions auprès du service web associé à la box domotique puis construire une signature à partir du trafic réseau de chacune d'entre elles. Par exemple, le nombre total de combinaisons d'actions sur uniquement 12 prises connectées est de $4^{12} = 16777216$.

4.5 Synthèse

Dans ce chapitre, nous nous sommes intéressés aux environnements domotiques où une box domotique contrôle un ensemble d'objets IdO. Ce type d'installation permet de simplifier les interactions vers les objets IdO. En effet, une box domotique est généralement capable d'interagir avec de nombreux objets IdO via des protocoles de communication sans-fil comme Z-Wave, ZigBee, Wi-Fi mais aussi des protocoles propriétaires, c'est-à-dire implémentés par les fabricants d'objets IdO eux-mêmes. Ainsi, en associant des objets IdO à une box domotique, un usager n'aura qu'à utiliser l'application mobile ou le service web proposé par le fabricant de la box domotique pour interagir avec les objets IdO associés à cette dernière.

Cependant, étant donné qu'il est possible de contrôler des objets IdO à distance, c'est-à-dire sans être connecté au réseau informatique de son domicile, les actions que souhaite effectuer un utilisateur sur ses objets IdO sont transmises à la box domotique via un des serveurs appartenant au fabricant de cette dernière localisé dans le cloud.

Ainsi, notre première contribution [191] s'intéresse aux informations qu'un attaquant pourrait justement inférer à partir des requêtes envoyées par le serveur à la box domotique, ou requêtes serveur-box. Contrairement aux travaux existants où il est supposé qu'un attaquant est capable

d'intercepter les communications des objets IdO, notre approche est originale, car les objets IdO sont associés à une box domotique et notre point d'observation se place dans l'Internet. De plus, les communications individuelles de chaque objet IdO ne sont pas non plus accessibles, c'est pourquoi nous utilisons uniquement les requêtes serveur-box qui, elles, sont généralement chiffrées.

Face au problème du chiffrement dans les requêtes serveur-box, notre méthode utilise la taille des données chiffrées pour déterminer les actions utilisateur. Pour ce faire, nous avons émis plusieurs hypothèses sur le fonctionnement de la box domotique et de son serveur. Parmi ces dernières, nous avons supposé que la box domotique recevait, dans une seule requête serveur-box, les actions demandées par l'utilisateur, et que la taille des données chiffrées de cette requête suit une relation linéaire avec les actions présentes dans cette dernière.

À l'aide de ces hypothèses, notre méthode détermine la taille des données chiffrées correspondante à chacune des actions utilisateur possibles afin de dériver la relation linéaire entre la taille des données de la requête serveur-box et les actions utilisateur qu'elle contient. Cependant, les tailles calculées à partir de la formule dérivée ci-avant peuvent différer des tailles des données chiffrées réellement observées. C'est pourquoi, notre méthode intègre également dans la relation linéaire une variable permettant d'évaluer cette différence.

Enfin, nous avons appliqué notre approche à une box domotique contrôlant jusqu'à 16 objets IdO, et montré qu'il était effectivement possible pour un attaquant localisé dans l'Internet, de trouver la combinaison d'actions demandée par un utilisateur dans 98.4% des cas mais parmi d'autres combinaisons d'actions candidates. De plus, en analysant le trafic réseau de la box domotique, nous avons remarqué que cette dernière, après avoir reçu une combinaison d'actions, initiait un nombre de connexions avec le serveur proportionnel au nombre d'objets IdO présents dans la combinaison. Ainsi, nous avons adapté notre méthode par rapport à cette nouvelle connaissance et limité le nombre de combinaisons d'actions retourné. Malgré une baisse de performance, notre méthode trouve la combinaison d'actions demandée par l'utilisateur dans 91.2% des cas. Même si nous ne pouvons déterminer l'exacte combinaison, les combinaisons d'actions retournées par notre approche sont plus proches de la vraie combinaison demandée par l'utilisateur, c'est-à-dire ont des actions communes.

Dans ce chapitre, nous avons montré que la présence d'une box domotique rendait, certes, l'inférence des actions utilisateur plus complexe mais pas totalement impossible malgré l'utilisation du chiffrement. En effet, nous avons vu que d'autres informations comme la taille des données chiffrées de certaines requêtes ou le nombre de requêtes pouvaient, involontairement, donner des informations sur les objets IdO associés à la box domotique et leurs actions reçues.

Troisième partie

Analyse de firmwares d'objets IdO

Chapitre 5

Analyse hybride de firmwares d' objets IdO

Sommaire

5.1 Introduction

Dans les chapitres 1 et 4 nous avons vu que l'analyse de trafic réseau, passive ou active, d'un objet connecté permettait de détecter des fuites de données et donc déduire certaines propriétés d'un objet comme ses services déployés, son modèle ou bien les actions qu'il effectue. De plus, certains travaux montrent qu'il est possible de déceler les attaques en cours, ou les objets compromis par une attaque.

Ce type d'analyse ne permet d'obtenir qu'une vue limitée de la sécurité d'un objet connecté. En effet, il est compliqué de déduire les vulnérabilités d'un objet à partir de son trafic réseau même si, certains travaux présentés dans la section 1.2.2 du chapitre 1, peuvent trouver les services déployés ainsi que le nom et la version des binaires utilisés par un objet connecté.

Cependant, suivant les binaires utilisés ou les interactions possibles avec l'objet, ces travaux ne retournent pas nécessairement l'intégralité de ces informations d'où la nécessité d'implémenter une autre méthode plus efficace afin trouver les vulnérabilités connues présentes dans un objet connecté.

D'un autre côté, à l'aide du nom et de la version des binaires, il est possible de retrouver les vulnérabilités connues associées à ces binaires afin 1) d'exploiter ces dernières ou 2) les corriger via des patches de sécurité. Par exemple, si le port 80 (HTTP) d'une machine est ouvert, et le binaire utilisé est `lighttpd` en version `1.4.35`, alors la vulnérabilité connue CVE-2019-11072 peut être exploitée pour réaliser une attaque par déni de service. C'est pourquoi, notre approche propose d'analyser la sécurité d'un objet connecté, en associant au nom et à la version des binaires présents dans ce dernier, la liste de ces vulnérabilités connues correspondantes.

Dans ce chapitre, nous nous focalisons sur les objets de l'Internet des Objets (IdO), et plus particulièrement aux systèmes embarqués comme des routeurs, passerelles ou prises connectés. Dans le contexte de l'IdO, dont le nombre est estimé à plus de 43 milliards d'ici 2023 [40], il est nécessaire d'élaborer une méthode satisfaisant le passage à l'échelle. Pour qu'une analyse ait des résultats significatifs, c'est-à-dire basée sur un large nombre d'objets, elle ne peut utiliser des objets IdO physiques. En effet, l'obtention d'un grand nombre d'objets physiques n'est pas toujours financièrement et/ou techniquement possible. D'un autre côté, la mise en place d'analyses de vulnérabilités par des exécutions symboliques, analyses de teinte ou des tests à données aléatoires, ou *fuzzing* en anglais, requiert souvent un temps non négligeable et a pour objectif de détecter de nouvelles vulnérabilités. Or, nous nous focalisons sur les vulnérabilités connues afin de déduire si un objet IdO est, dès sa commercialisation, vulnérable.

Pour faire face à ce problème, notre méthode, à l'image de certains travaux présentés dans le chapitre 2, utilise les firmwares d'objets IdO. Ces derniers, généralement accessibles en téléchargement depuis l'Internet, contiennent tous les fichiers utilisés par un objet IdO. Par exemple, nous pouvons en déduire les services déployés ou analyser spécifiquement n'importe quel binaire présent dans un firmware.

Notre contribution [189] a pour but d'évaluer le niveau de sécurité d'un firmware d'objet IdO à l'aide de deux mesures que sont 1) le niveau d'exposition du firmware, c'est-à-dire le nombre de binaires capables de déployer un service et, 2) la présence de vulnérabilités connues ciblant les binaires de ce firmware. Cette dernière mesure permet également de mettre en lumière les firmwares contenant des vulnérabilités lors de leurs publications, et en comparant les résultats au cours de plusieurs années, de conclure si les fabricants d'objets IdO continuent ou non à déployer des binaires vulnérables dans leurs objets. Ainsi, contrairement aux travaux présentés dans le chapitre 2, notre approche se base sur les vulnérabilités connues car, elles sont généralement publiées, et décrites, dans une base de données de vulnérabilités publique. De plus, un attaquant peut privilégier ces dernières au lieu d'utiliser une vulnérabilité inconnue, ou *0-day* en anglais, qui nécessite, elle, plus d'efforts pour la découvrir.

Le chapitre est découpé en quatre sections. Dans la section 5.2, nous discutons des challenges liés à une analyse de firmwares, et justifions notre choix des logiciels étudiés. Dans la section 5.3, nous détaillons notre méthode d'analyse de firmwares, puis présentons dans la section 5.4, les métriques utilisées pour évaluer la sécurité d'un firmware. Enfin, les résultats de notre analyse sont présentés dans la section 5.5.

40. Disponible sur `https://www.mckinsey.com/industries/private-equity-and-principal-investors/our-insights/growing-opportunities-in-the-internet-of-things`, dernier accès le 10/02/2022

5.2 Objectifs de notre analyse de firmwares

Notre contribution mesure la sécurité d'un objet IdO à partir de la composition de son firmware. Dans le reste de ce chapitre et compte tenu des statistiques sur la répartition des systèmes d'exploitation dans l'Internet des Objets détaillées dans le chapitre 2 de l'état de l'art, le terme firmware fera donc référence à un système de fichiers nécessaire au fonctionnement d'un système d'exploitation de type UNIX.

Comme illustré dans la figure 2.1 du chapitre 2 de l'état de l'art, un firmware d'objet IdO est composé de plusieurs fichiers et répertoires, notamment les fichiers de configuration des différents services comme dans `/etc/ssh/sshd_config` pour le service SSH ou bien les binaires employés pour déployer un service comme `/usr/sbin/lighttpd` pour un serveur HTTP. Par conséquent, les analyses de firmwares permettent d'obtenir un meilleur aperçu de la sécurité globale d'un objet IdO en vérifiant d'un côté les binaires ou le contenu des fichiers de configuration. Il sera donc possible d'exécuter, et analyser, n'importe quel binaire présent dans un firmware depuis un environnement externe, et donc sans risquer d'endommager l'objet IdO physique.

Contrairement aux travaux existants, notre analyse se focalise sur les vulnérabilités connues afin de mettre en lumière certaines pratiques de développement utilisées par les fabricants d'objets IdO, comme par l'exemple l'utilisation d'un binaire spécifique pour déployer un service ou bien le nombre de services déployés par objet IdO. Ce choix est motivé par le classement des dix plus importants risques de sécurité dans l'Internet des Objets proposé par le *Open Web Application Security Project (OWASP)* qui, dans son classement en 2018 [41], place en seconde position les *Insecure Network Services* que sont le démarrage de services non nécessaires ou vulnérables, et en cinquième position les *Use of Insecure or Outdated Components*, c'est-à-dire l'utilisation de binaires, de systèmes d'exploitation ou de composants matériels vulnérables.

Notre approche inspecte les deux aspects ci-après dans un firmware d'objet IdO :

1. le niveau d'exposition de l'objet, c'est-à-dire le nombre de binaires capables de déployer un service et pouvant être exploités par un attaquant. Par exemple, si nous trouvons dans un firmware les deux binaires `lighttpd` et `dropbear` alors nous conclurons que les services HTTP et SSH sont déployés par l'objet IdO.
2. À partir de ces binaires, nous vérifions si ces derniers sont sujets à des vulnérabilités connues, et plus particulièrement au moment de la publication du firmware. Par exemple, le binaire `dropbear` avec une version inférieure à 2016.74 est sujet à la vulnérabilité CVE-2016-7406 publiée en 2017 ainsi, si nous trouvons ce binaire dans un firmware distribué après 2017 alors nous pourrons conclure que le fabricant n'a pas tenu compte des vulnérabilités existantes et a publié un firmware vulnérable.

Afin de mener à bien ces analyses, nous discutons dans la section suivante de la principale hypothèse posée par notre approche, ainsi que des challenges soulevés par l'analyse de firmware appliquée à l'Internet des Objets.

5.2.1 Challenges et hypothèse

À partir des binaires et fichiers de configuration présents dans un firmware d'objet IdO, il est souvent compliqué de savoir, par défaut, quels sont les binaires exécutés. Une des approches serait de trouver le script d'initialisation, comme `/etc/init.d/rcS` ou `/init`, exécuté par le noyau Linux lors du démarrage de l'objet. Or, obtenir cette connaissance est souvent compliqué.

41. Disponible sur `https://owasp.org/www-pdf-archive/OWASP-IoT-Top-10-2018-final.pdf`, dernier accès le 11/02/2022

Par exemple, dans le chapitre 2, les approches effectuant les émulations complètes de firmwares d'objet IdO choisissent 1) le premier script d'initialisation trouvé ou 2) le script démarrant le plus de binaires. Supposons que le script d'initialisation est connu, il est alors nécessaire d'analyser ses instructions afin d'identifier les binaires démarrés qui, à leur tour, peuvent exécuter d'autres binaires. C'est pourquoi, nous supposons dans notre approche que les binaires présents dans un firmware sont, à un moment ou un autre, exécutés par l'objet IdO.

Une analyse à large échelle de firmwares d'objets IdO soulève plusieurs challenges comme :

C1 la collecte de firmwares d'objets IdO depuis l'Internet. En effet, chaque fabricant d'objets IdO implémente son site web d'une manière différente. Il est donc nécessaire d'inspecter manuellement chaque plan de site afin de dériver une suite d'instructions permettant d'accéder à la page web d'un des produits puis télécharger le ou les firmwares disponibles et extraire les informations associées aux firmwares téléchargés et au produit. Enfin, une fois les firmwares téléchargés, une opération d'extraction est nécessaire afin d'extraire les fichiers présents dans ces derniers. Or, cette dernière peut malheureusement échouer ou monopoliser les ressources d'un serveur sur une période de temps indéterminée car certains firmwares peuvent 1) être chiffrés, ce qui empêche une tierce personne de les déchiffrer sans effectuer au préalable de la rétro-ingénierie, ou 2) utiliser un format de données non standard ne permettant pas une extraction sans un algorithme spécifique.

C2 Le système de fichiers extrait d'un firmware peut contenir plusieurs incohérences. Par exemple, il est possible qu'un ou plusieurs systèmes de fichiers complets, ou incomplets, soient présents dans un seul firmware. Ainsi, les méthodes d'analyse dynamique de firmwares reposant sur l'exécution de binaires pour tester leurs fonctionnalités via du *fuzzing* ou des exécutions symbolique, peuvent ne pas être applicables.

C3 Obtenir un système de fichiers complet ne garantit pas non plus de pouvoir exécuter un binaire. En effet, un binaire peut utiliser certaines ressources matérielles pour, par exemple, récupérer des données de capteurs via le fichier `/dev/gpio`, ou obtenir des paramètres de configuration système en lisant le contenu de la mémoire RAM non volatile (NVRAM) à l'aide du fichier `/dev/nvram`. Or, il est compliqué de correctement répondre à ces besoins matériels afin de pouvoir exécuter correctement les binaires présents dans un système de fichiers.

C4 Les objets IdO présents sur le marché proposent des fonctionnalités variées. Les composants matériels présents dans ces derniers sont hétérogènes [18] et de nombreuses architectures processeur (CPU) comme ARM, MIPS ou PowerPC sont utilisées. Or, les serveurs et ordinateurs personnels vendus de nos jours utilisent majoritairement des architectures CPU de type x86, ce qui signifie que, par défaut, ces derniers peuvent uniquement interpréter, ou exécuter, des binaires compilés pour du x86. Notre approche requiert d'exécuter des binaires compilés pour des architectures CPU différentes.

En prenant en compte ces challenges et notre hypothèse, nous justifions dans la section suivante notre choix d'une analyse de firmware hybride, c'est-à-dire combinant une analyse statique et/ou dynamique.

5.2.2 Analyse hybride de firmwares

Afin de retrouver les vulnérabilités connues associées à un binaire, il est souvent nécessaire de connaître le nom ainsi que la version du binaire. Ainsi, pour extraire ces informations d'un binaire, notre approche permet d'effectuer une analyse statique ou dynamique décrites, respectivement, ci-après.

— L'analyse statique d'un binaire ou d'un fichier de configuration repose entièrement sur l'analyse de son contenu sans exécuter de binaires. Cette analyse fonctionne même si le firmware contient un système de fichiers incomplet (**C2**) et ne dépend pas de composants matériels spécifiques (**C3**). De plus, dans notre cas, nous souhaitons uniquement extraire le nom et la version du binaire qui peuvent être directement présents dans le code de ce dernier. Donc nous n'avons pas besoin d'analyser les instructions (**C4**).

— L'analyse dynamique consiste à exécuter un binaire puis analyser son fonctionnement. L'extraction du nom et de la version sont souvent obtenables à partir d'une exécution partielle, c'est-à-dire une exécution demandant au binaire de décrire son fonctionnement. Par exemple, exécuter un binaire avec les arguments `-h` ou `-help` peut résulter à un message indiquant le nom et la version du binaire puis la description des autres paramètres. Par conséquent, demander ce type de message au binaire ne requiert souvent pas de dépendances à des composants matériels (**C3**) et, à l'aide de QEMU [24], il est possible d'exécuter des binaires initialement compilés pour d'autres architectures CPU (**C4**).

D'ailleurs ces analyses sont combinables. Par exemple, une analyse dynamique peut être d'abord effectuée puis, en cas d'échec de cette dernière, l'extraction du nom et de la version d'un binaire peut être effectuée via l'analyse statique. Dans la section suivante, nous discutons de notre choix des services, et donc binaires ou logiciels, analysés par notre approche.

5.2.3 Services et logiciels analysés

Nous focalisons notre analyse de firmwares sur les logiciels associés aux services listés dans le tableau 5.1. Nous justifions le choix de ces services par rapport aux attaques répandues dans l'Internet des Objets. Tout d'abord, nous étudions les services SSH et TELNET utilisés notamment par des botnets comme Mirai [91, 13] ou Kaiji[42] pour infecter des objets IdO. En effet, ces botnets utilisent ces deux services pour se connecter à une machine distante à l'aide d'une méthode *brute-force* sur les identifiants de connexion. Néanmoins, les logiciels déployant SSH peuvent également contenir des vulnérabilités comme les CVE-{2016-7406, 2015-5600} où respectivement, un attaquant peut effectuer à distance une injection de commande sur le binaire `dropbear` et, causer une attaque par déni de service sur le binaire `OpenSSH`.

Le service web HTTP(S) déployé dans certains objets IdO peut également être utilisé par des attaquants pour 1) compromettre l'objet depuis lequel le service est déployé ou 2) utiliser les fonctionnalités disponibles pour obtenir des informations compromettantes sur l'utilisateur, par exemple enregistrer le flux vidéo d'une caméra connectée. Parmi les vulnérabilités associées au service web, on pourra citer des dépassements de tampon ou *buffer overflow* en anglais dans les CVE-{2013-2028, 2018-16119, 2019-11072}, des injections de commandes dans les CVE-{2016-1555, 2017-14250,2018-14067}, ou bien des attaques par traversée de répertoires ou *path traversal* en anglais comme les CVE-{2015-3035, 2018-19052, 2021-41449} permettant à un attaquant d'accéder de manière illégitime à des fichiers présents sur la machine déployant le serveur HTTP(S).

42. Disponible sur `https://www.zdnet.com/article/new-kaiji-malware-targets-iot-devices-via-ssh-brute-force-attacks/`, dernier accès le 14/02/2022

Services	Logiciels
DNS	bind8, bind9, dnsmasq, tinydns, knotd, nsd, maradns, deadwood, pdnsd, posadis, pdns, unbound, yadifad
FTP	ftpd, vsftpd, uftpd, uftp, bftpd, proftpd
HTTP	httpd, nginx, boa, lighttpd, hiawatha, mongoose, thttpd, jjhttpd, mini_httpd, uhttpd, cherokee, jetty, monkey, hfs, navi, shttpd
NTP	ntpd, openntpd, sntp, sntpd, ntimed, ntpsec, chronyd
RPC	rpc.statd, rpcd
SNMP	snmpd, net-snmp, nagios, prometheus, zabbix, icinga, cacti
SSDP/UPnP	ssdp, upnpd, miniupnpd, minissdpd, minidlna, ushare, mediatomb, gmediaserver, gerbera
SSH	dropbear, openssh, apache mina, copssh, teleport, wolfssh
TELNET	telnetd, utelnetd

Tableau 5.1 – Liste des services considérés et des logiciels, ou binaires, analysés

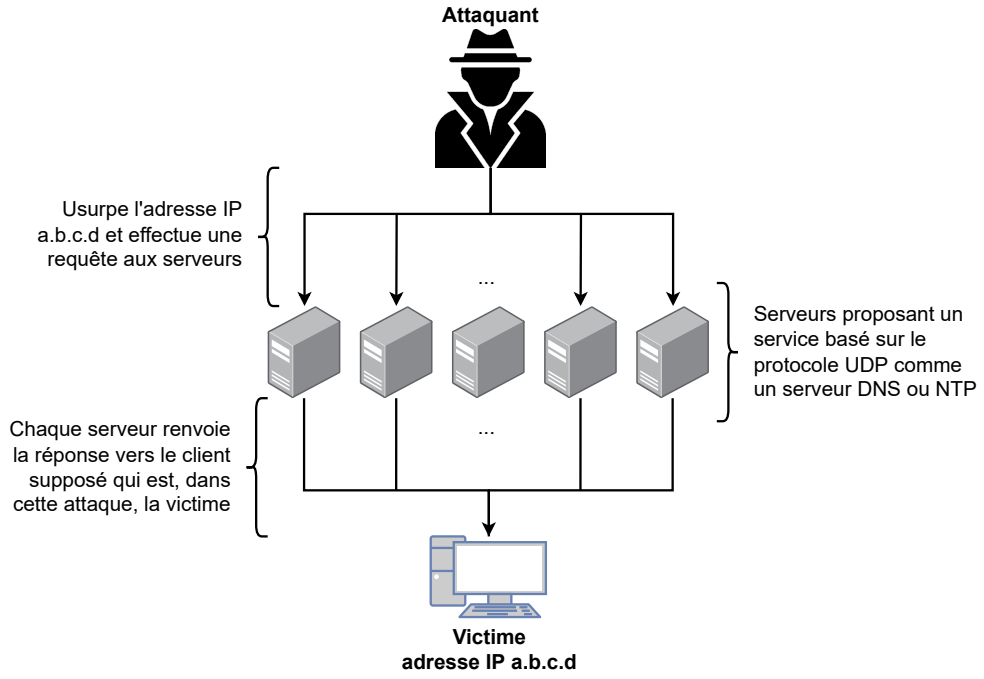


FIGURE 5.1 – Exemple d'une attaque par réflexion sur une victime localisée dans l'Internet à l'adresse IP a.b.c.d

Les protocoles DNS, NTP, RPC, SNMP, SSDP/UPnP utilisent le protocole UDP qui, contrairement à TCP, ne permet pas de vérifier l'identité de l'initiateur d'une requête. C'est pourquoi ces derniers peuvent également être utilisés par des attaquants pour effectuer une attaque par réflexion, voire amplification, sur une cible distante [39, 65, 78] comme illustrée dans la figure 5.1. Une attaque par amplification est une attaque par réflexion à la différence que, comme son nom le laisse entendre, la taille de la requête initiale effectuée par l'attaquant résulte en une réponse par le serveur d'une taille supérieure. Ainsi, cette dernière permet d'utiliser moins de ressources intermédiaires, c'est-à-dire des serveurs, que l'attaque par réflexion *basique* et ainsi submerger plus efficacement le trafic réseau de la victime, à l'image d'une attaque par déni de service. Par

exemple, une attaque par amplification sur le protocole NTP [43] a permis à des attaquants de générer des attaques par déni de service générant plus de 400 gigaoctets (Go) de données par seconde en utilisant 4 529 serveurs NTP. En moyenne, une attaque par amplification à l'aide du protocole NTP a un coefficient d'amplification de 200 [174], c'est-à-dire la taille de la requête générée par le serveur est 200 fois plus grande que la taille de la requête initiale. D'un autre côté, une attaque par amplification sur le protocole DNS génère *seulement* des réponses huit fois plus grandes que les requêtes initiales alors que, théoriquement, le protocole SNMP serait capable d'atteindre un coefficient d'amplification de 650 [44]. En plus de ces attaques par réflexion, les binaires utilisés pour déployer ces protocoles sont eux aussi sujets à de nombreuses vulnérabilités comme des dépassements de tampon (CVE-{2012-5959, 2019-14363,2020-5591}), des injections de commandes (CVE-{2007-2447, 2018-1000116, 2019-3925}) ou des élévations illégitimes de privilèges (CVE-{2016-0727, 2020-15861}).

Nous nous intéressons également au protocole FTP car ce dernier, peut être utilisé par les objets IdO pour stocker des données liées aux utilisateurs, par exemple le flux vidéo d'une caméra connectée ou des données personnelles comme des photos, vidéo ou documents divers.

Ainsi, lors de l'analyse d'un firmware, notre approche va donc rechercher à identifier les binaires listés dans le tableau 5.1 puis, mesurer les risques de sécurité associées au firmware et aux binaires trouvés à l'aide des métriques détaillées dans la section 5.4.

5.3 Méthode

Dans cette section, nous détaillons les étapes illustrées dans la figure 5.2 permettant à notre approche de télécharger des firmwares d'objets IdO depuis l'Internet puis analyser les binaires présentés dans la section 5.2.3 afin de construire un ensemble de données sur les compositions de ces firmwares. Chacune de ces étapes est disposée dans l'un des deux modules suivants :

(A) le module d'obtention de firmwares présenté en section 5.3.1 effectue premièrement l'étape 1) téléchargeant des firmwares d'objets IdO depuis l'Internet, puis l'étape 2) associant à chacun des firmwares téléchargés un type d'objet IdO. Par exemple, déterminer s'il s'agit d'un firmware d'une prise connectée ou d'un routeur.

(B) Le module d'analyse de firmwares décrit en section 5.3.2 se compose de l'étape 3) dont le rôle est d'extraire d'un firmware le ou les systèmes de fichiers présents dans ce dernier, et de l'étape 4) effectuant l'analyse du firmware à l'aide d'une analyse statique ou dynamique afin de récupérer certaines informations sur les binaires ou fichiers de configuration trouvés dans le ou les systèmes de fichiers extraits. En cas d'échec de l'analyse dynamique, le module bascule automatiquement vers l'analyse statique.

Enfin, à partir de notre ensemble de données sur les compositions des firmwares, nous mesurons le niveau d'exposition ainsi que les risques de sécurité.

La base de données sur les compositions de ces firmwares ainsi obtenue nous permet de mesurer les risques de sécurité via les métriques présentées dans la section 5.4. Nous détaillons dans les prochaines sections chacune des étapes implémentée dans les deux modules présentés ci-avant.

43. Disponible sur `https://blog.cloudflare.com/technical-details-behind-a-400gbps-ntp-amplification-ddos-attack/`, dernier accès le 23/03/2022

44. Disponible sur `https://blog.cloudflare.com/understanding-and-mitigating-ntp-based-ddos-attacks/`, dernier accès le 23/03/2022

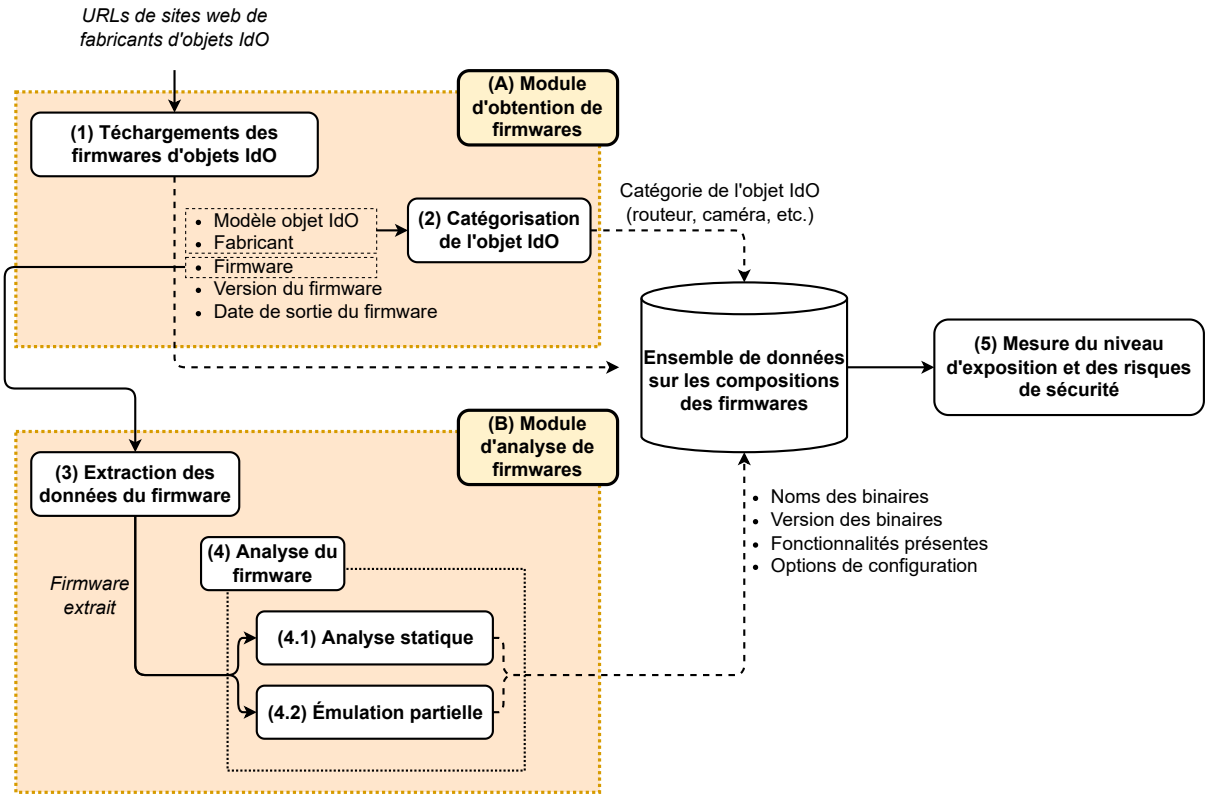


FIGURE 5.2 – Étapes suivies par notre approche pour construire une base de données sur les compositions de firmwares d'objets IdO

5.3.1 Obtention de firmwares

Dans cette section, nous détaillons le module d'obtention de firmwares illustré dans la figure 5.2. Nous présentons dans la section 5.3.1.1 la première étape de ce module consistant à télécharger des firmwares d'objets IdO à partir de sites internet, puis discutons dans la section 5.3.1.2 de l'étape associant à un firmware une catégorie d'objet IdO.

5.3.1.1 Téléchargement des firmwares depuis l'Internet

Notre analyse étant focalisée sur les objets IdO, il est nécessaire que les firmwares téléchargés soient liés à des objets IdO. Une approche naïve consisterait à obtenir un accès au serveur FTP appartenant à un fabricant d'objets IdO puis télécharger l'intégralité des fichiers proposés. Cette approche a plusieurs désavantages :

— il n'est pas garanti qu'un fabricant d'objet IdO met à disposition un serveur FTP où seuls des firmwares sont disponibles.

— L'organisation des fichiers dans le serveur FTP ne permet pas nécessaire d'obtenir de manière exacte les informations comme le produit, la date de sortie ou la version associée à un chaque firmware.

— Certains fabricants d'objets IdO peuvent également proposer d'autres produits n'appartenant pas à l'Internet des Objets. Ainsi, ignorer les firmwares appartenant à ces derniers nécessite des traitements complémentaires.

C'est pour ces raisons que notre approche suit la méthode introduite par [36] où les sites de fabricants d'objets IdO sont parcourus à l'aide d'intructions spécifiques, propres à chaque site, afin de trouver la page web associée à chacun des produits disponibles, c'est-à-dire les objets IdO, pour ensuite télécharger les firmwares et extraire les informations sur le produit comme le nom du modèle ou la version du firmware téléchargée. Pour ce faire, nous utilisons l'outil Scrapy [45] dont nous expliquons le fonctionnement dans l'annexe A.

À l'aide de cet outil, notre approche est capable d'explorer les sites web de grands fabricants d'objets IdO, comme NETGEAR ou D-Link, afin de télécharger les firmwares et extraire les informations associées. Parmi ces informations, chaque objet IdO est modélisé par :

— son modèle,

— le nom de son fabricant,

— l'URL de téléchargement de son firmware,

— la version du firmware,

— la date de publication du firmware.

Comme certains fabricants proposent également d'autres types de produits (par exemple ASUS avec des cartes mères), notre approche ignore ces derniers et se focalise uniquement sur les systèmes embarqués comme des routeurs, caméras, prises connectées, etc.

5.3.1.2 Catégorisation des objets IdO

Notre analyse se focalise sur les objets de l'Internet des Objets (IdO), et plus particulièrement aux systèmes embarqués. En effet, dans les chapitres 2 et 3 de l'état de l'art, le terme IdO englobe généralement des objets comme des routeurs, commutateurs ou des caméras connectées. Plus globalement, ce terme englobe des appareils réseau et des objets IdO. Or, les appareils réseau proposent généralement plus de fonctionnalités que les objets IdO, comme des prises ou caméras connectées.

Afin de comparer les résultats, notamment le niveau d'exposition, entre les objets IdO axés réseau et les *vrais* objets IdO, nous utilisons la méthode décrite dans l'annexe B pour assigner à chaque objet présent dans notre base de données, une catégorie principale pouvant être `appareil réseau` ou `objet IdO`, et une ou plusieurs sous-catégories parmi celles listées dans le tableau B.1. Par exemple, la caméra connectée `DCS-2310L` de marque D-Link se voit affectée la catégorie `objet IdO` et la sous-catégorie `caméra`.

Bien que cette approche a été créée et configurée grâce à plusieurs itérations manuelles, nous discutons de ces performances dans la section 5.5.1.

5.3.2 Analyse de firmwares

Dans cette section, nous présentons le module d'analyse de firmwares visible dans la figure 5.2. Ce dernier se charge de l'extraction des données présentes dans le firmware téléchargé dont nous décrivons le fonctionnement dans la section 5.3.2.1. À partir des données extraites, nous détaillons l'analyse statique dans la section 5.3.2.2, et l'analyse dynamique dans la section 5.3.2.3.

5.3.2.1 Extraction des données du firmware

Afin de limiter l'espace de stockage utilisé par les firmwares sur les serveurs, les fabricants compressent généralement ces derniers sous forme d'archives au format TAR, ZIP ou Gzip.

45. Disponible sur `https://scrapy.org/`, dernier accès le 13/02/2022

En plus d'un ou plusieurs systèmes de fichiers, un firmware peut contenir d'autres éléments, comme un noyau Linux, une documentation expliquant le processus d'installation du firmware, ou bien un chargeur d'amorçage (*bootloader* en anglais) dont le rôle est, entre autres, d'assurer la mise à jour du firmware sur l'objet IdO.

Dans le chapitre 2 de l'état de l'art discutant de l'analyse de firmwares, les trois logiciels suivants : Binary Analysis Toolkit (BAT) [72], Binwalk [46] ou FRAK [47] sont notamment utilisés pour décompresser les archives téléchargées, et ainsi retrouver les différents fichiers présents dans ces dernières. Pour ce faire, ces méthodes identifient récursivement les fichiers présents dans un firmware à l'aide de signatures et extraient leur contenu à l'aide de l'algorithme spécifique au format de données détecté. Par exemple, Binwalk identifie une archive ZIP si les quatre premiers octets d'un fichier correspondent à la signature `"PK\003\004"`, puis utilise `unzip` pour extraire son contenu.

Nous utilisons Binwalk car, contrairement à FRAK, Binwalk est un logiciel open source bénéficiant, grâce à une communauté très active, de nombreuses mises à jours et ajout de fonctionnalités. De plus, Binwalk a été largement amélioré par [36] afin de rendre l'outil plus évolutif et plus performant que BAT, et ainsi capable de gérer certains systèmes de fichiers présents dans les objets IdO comme JFFS2 ou SquashFS.

Dossier	**Description**
`bin`	Binaires pour les commandes utilisateurs (par exemple cat, ls, cd)
`boot`	Fichiers associés au démarrage du système
`dev`	Fichiers de gestion de périphériques (par exemple clavier, souris)
`etc`	Fichiers de configuration
`lib`	Librairies partagées et modules pour le noyau Linux
`media`	Point de montage (par exemple pour les CD-ROM ou disquettes)
`mnt`	Point de montage (par exemple pour les clés USB ou disques dur externes)
`opt`	Fichiers nécessaires au fonctionnement de certains logiciels
`run`	Information sur les processus en cours d'exécution
`sbin`	Binaires essentiels au fonctionnement du système
`srv`	Données nécessaires au fonctionnement de certains services
`tmp`	Stockage des fichiers temporaires
`usr`	Binaires, fichiers ou librairies partagées disponibles pour les utilisateurs
`var`	Stockage des variables associées aux processus ou certains programmes en cours d'exécution

Tableau 5.2 – Liste des dossiers présents dans un système de fichiers racine selon [177]

Après avoir extrait les éléments présents dans une archive à l'aide de Binwalk, il est nécessaire de localiser le firmware, c'est-à-dire le système de fichiers contenant les binaires et fichiers de configuration. Comme, plusieurs systèmes de fichiers peuvent être trouvés ainsi, il est nécessaire de sélectionner le système de fichiers le plus complet, c'est-à-dire le système de fichiers racine ou *root filesystem* en anglais. Pour ce faire, nous considérons le système de fichiers contenant le nombre maximum de dossiers listés dans le tableau 5.2 comme étant le système de fichiers racine.

46. Disponible sur `https://github.com/ReFirmLabs/binwalk`, dernier accès le 14/02/2022

5.3.2.2 Analyse statique

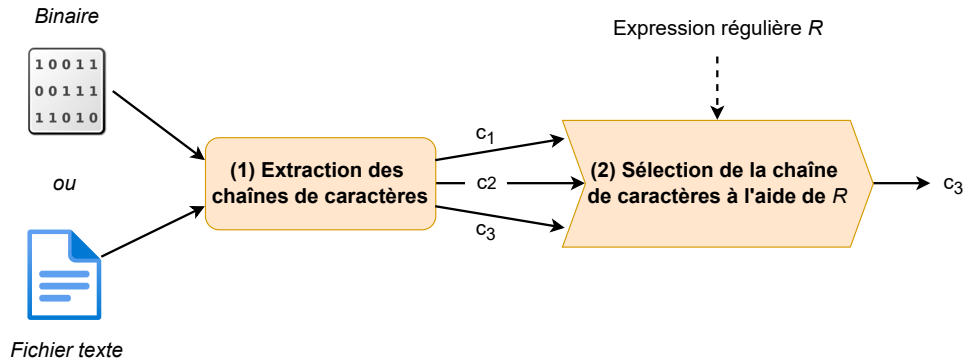


FIGURE 5.3 – Analyse statique d'un binaire ou d'un fichier texte composée de 1) l'extraction de l'ensemble des chaînes de caractères présentes dans ce dernier, puis de 2) la sélection de la chaîne de caractères par rapport à l'expression régulière fournie R

Notre processus d'analyse statique (figure 5.3) consiste premièrement à extraire l'ensemble des chaînes de caractères présentes dans un fichier de configuration ou binaire, puis à sélectionner la chaîne de caractères recherchée par l'analyse selon des critères prédéfinis présentés dans la section 5.3.2.4. Avant de rentrer dans les détails de notre analyse statique, nous détaillons tout d'abord l'intérêt de cette approche sur des binaires qui sont censés être composés uniquement d'instructions au format d'une architecture processeur donnée.

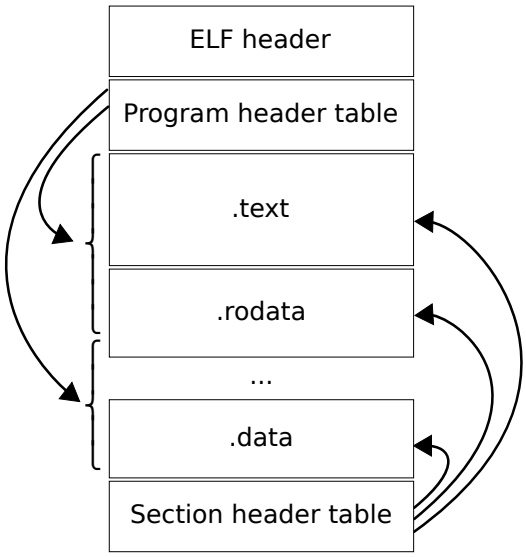


FIGURE 5.4 – Sections composant un fichier binaire au format *Executable and Linkable Format (ELF)* d'après [175]

Un binaire capable de déployer un service est généralement structuré au format *Executable and Linkable Format (ELF)* lors de sa compilation. Durant ce processus, un compilateur comme *GNU Compiler Collection (gcc)*[47], va interpréter le code source puis assigner son contenu dans

47. Disponible sur `https://gcc.gnu.org/`, dernier accès le 16/02/2022

les différentes sections visibles dans la figure 5.4. Ainsi, certaines données présentes dans le code source comme des chaînes de caractères peuvent être assignées à une de ces trois sections : `.text`, `.data` ou `.rodata`. Par conséquent, si le code source contient des chaînes de caractères comme des mots de passe ou bien le message d'information contenant notamment la version et le nom associés à ce code source, alors ces dernières seront visibles depuis l'une des trois sections listée ci-dessus. D'ailleurs, l'extraction de chaînes de caractères est utilisée, entre autres, pour analyser des *malwares*[96, 58, 87].

Ainsi, nous utilisons l'outil `strings`, proposé librement par GNU dans son ensemble d'outils de développement logiciel *GNU Binary Utilities*, capable de détecter automatiquement les chaînes de caractères humainement lisibles dans un fichier en vérifiant s'il existe des suites d'octets $o_1, o_2, .., o_n$ tel que chaque octet o_i avec $1 \leq i \leq n$ appartienne à un intervalle de valeurs prédéfini suivant l'encodage utilisé par la machine (par exemple `UTF-8`).

5.3.2.3 Analyse dynamique

Contrairement à l'analyse statique où des informations peuvent être également extraites de fichiers textes, cette approche, illustrée dans la figure 5.5, se focalise sur les binaires via leur exécution. Cependant, comme évoqué dans la difficulté **C4** de la section 5.2.1, pouvoir exécuter des binaires compilés pour des architectures processeurs différentes de celle de la machine utilisée pour l'analyse requiert l'usage d'un émulateur. Notre choix s'est porté sur l'émulateur QEMU en mode utilisateur, ou *user mode* en anglais, car ce dernier :

1. supporte les architectures CPU majoritairement observées dans l'Internet des Objets [45].
2. QEMU est open source [48] et bénéficie de patchs proposés par la communauté dont certains permettant l'émulation de nouvelles architectures CPU [79, 179].
3. Enfin, QEMU est également utilisé par les travaux rencontrant des problèmes similaires [36, 45, 90].

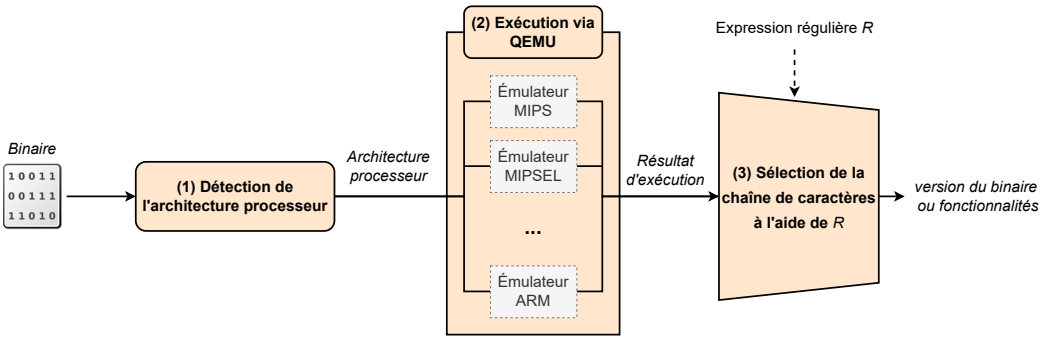


FIGURE 5.5 – Extraction de la version ou des fonctionnalités d'un binaire par l'approche dynamique

Avant d'exécuter le binaire étudié b, la première étape consiste à identifier l'architecture processeur de b pour sélectionner l'émulateur QEMU compatible, b étant au format *Executable and Linkable Format (ELF)*, des informations présentes dans l'entête ELF, illustrée dans la figure 5.4, permettent de dériver l'architecture processeur attendue. Parmi ces informations, nous avons :

— Le nombre de bits correspondant à la taille des mots [49] utilisée par le processeur, et qui

48. Disponible sur `https://github.com/qemu/qemu`, dernier accès le 17/02/2022
49. Voir `https://fr.wikipedia.org/wiki/Mot_(architecture_informatique)`, dernier accès le 17/02/2022

peut prendre deux valeurs possibles 64 bits ou 32 bits.

— Le boutisme ou *endianness* en anglais indiquant l'ordre des octets utilisé par le processeur dont les deux valeurs possibles sont 1) le gros-boutiste, ou *big-endian (MSB)* en anglais, ou 2) le petit-boutiste, ou *little-endian (LSB)* en anglais.

— La machine cible, ou *instruction set architecture (ISA)* en anglais, pouvant prendre des valeurs comme MIPS, ARM, PowerPC ou x86.

Afin d'extraire ces trois valeurs, nous utilisons l'outil `file`[50] sur le binaire à exécuter. Enfin, c'est à partir de ces trois valeurs que notre approche dérive l'émulateur QEMU à utiliser. Par exemple, le triplet <32-bit, MSB, MIPS> va entraîner l'utilisation de l'émulateur `qemu-mips-static`.

Cependant, exécuter naïvement un binaire avec QEMU peut résulter en des erreurs liées, par exemple, à des ouvertures de fichiers non existants ou des chargements de librairies partagées effectués par le binaire [45, 36, 90]. Pour pallier à ces erreurs, nous utilisons l'option `-L` proposée par QEMU pour spécifier le chemin à préfixer lorsque des chargements de librairies partagées effectués par le binaire durant son exécution. Ainsi, il suffit de spécifier à l'aide de cette option le chemin vers le système de fichiers racine du firmware analysé R pour que les binaires exécutés chargent les librairies partagées présentes dans R.

Enfin, les arguments avec lesquels exécuter un binaire lui sont propres, même s'il est commun de voir des arguments comme `-h`, `-v` ou `-help`. Ainsi, il est possible de paramétrer notre approche afin d'assigner un ou plusieurs arguments lors de l'exécution d'un binaire.

Dans le cas où l'exécution échoue, ce que nous caractérisons par une exécution ne retournant pas de données, notre approche bascule vers l'approche statique.

5.3.2.4 Extraction de l'information

À partir des chaînes de caractères obtenues par l'analyse statique ou dynamique, il est désormais nécessaire d'isoler la chaîne de caractères contenant l'information recherchée, comme la version du binaire ou une option dans un fichier de configuration.

Notre approche associe au binaire ou fichier de configuration analysé, une expression régulière (regex) ou *regular expression* en anglais. Par exemple, dans la figure 5.6, la version de BusyBox est extraite d'une chaîne de caractères à l'aide de l'expression régulière `(v\d.\d{2}.\d)`.

BusyBox v1.27.2 (Ubuntu 1 :1.27.2-2ubuntu3.4) multi-call binary.

$\downarrow (v\backslash d.\backslash d\{2\}.\backslash d)$

v1.27.2

FIGURE 5.6 – Extraction de la version de BusyBox à partir d'une chaîne de caractères et de l'expression régulière `(v\d.\d{2}.\d)`

Il est donc nécessaire d'établir manuellement une expression régulière pour chacune des propriétés que l'on souhaite extraire. À titre d'exemple, deux expressions régulières sont associées à chaque binaire étudié afin d'extraire 1) son nom et 2) sa version.

50. Disponible sur `http://darwinsys.com/file/`, dernier accès le 17/02/2022

5.4 Mesure des risques de sécurité

Nous détaillons dans cette section les trois métriques calculées à partir des firmwares extraits. La première de ces métriques, le niveau d'exposition, mesure le nombre de services que peut déployer un firmware et est présentée dans la section 5.4.1. La seconde métrique, discutée dans la section 5.4.2, permet d'évaluer le niveau d'obsolescence des binaires, c'est-à-dire mesurer si les binaires présents dans un firmware étaient, au moment de la publication de ce dernier, distribués avec une version récente. Enfin, dans la section 5.4.3, nous introduisons notre troisième métrique analysant les vulnérabilités connues, c'est-à-dire listées dans la base de données de vulnérabilités CVE, et présentes dans les firmwares afin de déduire si, à la date de publication du firmware, ces dernières étaient connues ou non.

Ces métriques se basent sur les services et binaires présentés dans le tableau 5.1 de la section 5.2.3, dont les relations peuvent être formalisées par i couples $<$ service S_i, binaires B_i $>$ où $Si \in [DNS, FTP, ..., TELNET]$ et B_i représente la liste des binaires $b_1, b_2, ..., b_n$ capable de déployer S_i. De plus, B représente l'ensemble des binaires étudiés, c'est-à-dire $B = B_{DNS} \cup B_{FTP} ... \cup B_{TELNET}$.

5.4.1 Niveau d'exposition

Le contenu du système de fichiers racine extrait d'un firmware f associé à un objet IdO d permet d'obtenir de nombreuses informations, notamment les services S_d que d peut déployer. En termes de risques de sécurité informatique, plus le nombre de services déployés par un objet connecté est grand, plus la surface d'attaque disponible pour un attaquant est grande. En effet, un attaquant pourra effectuer plus d'attaques et ainsi augmenter ses chances de compromettre l'objet ciblé. Par exemple, si un objet déploie un serveur TELNET et un service web HTTP alors, un attaquant peut d'abord initier une attaque *brute-force* sur le service TELNET puis, en cas d'échec, essayer de compromettre le service web à l'aide d'attaques comme des injections de commandes ou des traversées de répertoires illégitimes.

Le niveau d'exposition d'un objet IdO d, noté $exp(d)$, correspond au nombre de binaires dans d capables de déployer un des neuf services étudiés. En d'autres termes, $exp(d)$ se calcule de la manière suivante :

$$exp(d) = |Bin_d \cap B| \tag{5.1}$$

avec Bin_d, la liste des binaires présents dans le firmware associé à d, et obtenable à l'aide des analyses détaillées dans la section 5.3.2.

5.4.2 Obsolescence des binaires

Selon l'OWASP, l'utilisation de binaires ou systèmes d'exploitation obsolètes se place en cinquième position des risques de sécurité les plus présents dans l'Internet des Objets. Par exemple, [173] montre que certains objets IdO utilisent des versions du système d'exploitation Linux datant de plus de dix ans. Nous définissons cette métrique pour mesurer si un binaire b utilisé par un objet connecté d est à jour au moment où d est disponible sur le marché. De plus, cette métrique requiert au préalable pour chacun des binaires b étudiés, la liste L_b des versions avec leurs dates de publications associées.

À partir de L_b, nous mesurons l'obsolescence d'un binaire b utilisé par un objet IdO d de deux manières. La première mesure $\delta^d_{dver}(d, b)$ se calcule par l'équation 5.2 et mesure le nombre de jours séparant la version du binaire b trouvée dans le firmware de d, notée ver^d_b, et la dernière

version de b disponible au moment où le firmware est publié, notée ver_b^{last}. L'opération $date(v)$ permet de retourner la date de publication de la version v.

$$\delta^d_{dver}(d,b) = date(ver_b^{last}) - date(ver_b^d) \tag{5.2}$$

Cependant, suivant la cadence de développement d'un binaire b, il est possible que la valeur retournée par l'équation 5.2 soit importante et pas nécessairement représentative du niveau d'obsolescence de b. C'est pourquoi, nous calculons dans l'équation 5.3 l'équivalent de la précédente mesure mais en utilisant comme unité de mesure, le nombre de versions qui sépare la dernière version de b disponible au moment où le firmware de d est publié et la version du binaire b présente dans d.

$$\delta^d_{\#version}(d,b) = ver_b^{last} - ver_b^d \tag{5.3}$$

Ainsi, à l'aide de ces deux équations, dont les résultats sont des nombres entiers, nous mesurons le niveau d'obsolescence d'un binaire b en nombre de jours ou de versions. Cependant, un binaire non à jour, n'implique pas nécessairement qu'il soit vulnérable à une ou plusieurs vulnérabilités d'où l'introduction de la troisième métrique.

5.4.3 Vulnérabilités des binaires

Cette troisième métrique indique pour un binaire b utilisé par un objet connecté d, si ce dernier souffrait ou non d'une ou plusieurs vulnérabilités connues au moment de sa publication. Nous utilisons la base de données de vulnérabilités CVE pour connaître les vulnérabilités dont ver_b^d peut souffrir. De plus, nous sélectionnons la CVE la plus ancienne que l'on notera $cve_{ver_b^d}$. Ainsi, ce délai post-vulnérabilité, se calcule comme suit :

$$\delta^d_{dvuln}(d,b) = date(d) - date(cve_{ver_b^d}) \tag{5.4}$$

où l'opération $date(d)$ retourne la date de sortie de l'objet connecté d, alors que $date(cve_{ver_b^d})$ retourne la publication de la CVE $cve_{ver_b^d}$.

Cependant, chaque CVE est associée à un score appelé *Common Vulnerability Scoring System (CVSS)*[51] indiquant si les dommages occasionnés par la vulnérabilité sont minimes ou importants. Notre analyse se focalise sur les CVE classées comme *High* ou *Critical*, c'est-à-dire permettant de compromettre un objet ou accéder à ce dernier de manière illégitime. De plus, pour exploiter une CVE, il peut être nécessaire d'avoir un accès local à l'objet ou bien de pouvoir communiquer avec ce dernier depuis l'Internet ainsi, nous considérons uniquement les CVE pouvant être exploitées par un attaquant depuis l'Internet ou, dans la syntaxe CVSS, les CVE ayant un *Access Vector (AV)* assigné à *Network (N)*.

Contrairement aux deux précédentes métriques, l'équation 5.4 peut retourner des valeurs négatives. Si tel est le cas, cela indique que la première vulnérabilité $cve_{ver_b^d}$ a été découverte après que l'objet d ait été mis en vente sur le marché. Par conséquent, cela indique que d ne comportait pas de vulnérabilité associée au binaire b et donc, était considéré comme sécurisé au moment de sa publication.

51. Voir `https://nvd.nist.gov/vuln-metrics/cvss`, dernier accès le 21/02/2022

5.5 Expérimentation et résultats

Dans cette section, nous présentons la composition de la base de données de firmwares construite, puis discutons de l'évolution du niveau d'exposition des objets IdO analysés, et ensuite détaillons les répartitions des binaires déployant les services HTTP et SSH ainsi que leurs vulnérabilités associées.

5.5.1 Description de la base de données de firmwares

Nous avons implémenté un *Spider* Scrapy, comme détaillé dans la section 5.3.1.1, pour chacun des 11 fabricants d'objets IdO suivants : Asus, Belkin, D-Link, Edimax, Linksys, NETGEAR, Reolink, Tp-Link, Trendnet, Ubiquiti et 360. Dans la suite de cette section, nous utilisons le terme produit pour référer un couple < nom de l'objet IdO, version du firmware>.

Nous avons téléchargé, durant le mois de mai 2020, 9 106 firmwares, équivalent à 152 Go de données, correspondant à 12 047 produits. La répartition de ces derniers par marque est donnée dans le tableau 5.3. Nous observons qu'il existe plus de produits que de firmwares, car certains fabricants utilisent le même firmware dans plusieurs produits. En analysant manuellement les scripts présents dans des firmwares utilisés par plusieurs objets IdO, nous remarquons qu'ils utilisent les ressources matérielles, notamment la NVRAM, pour identifier le modèle de l'objet IdO à partir duquel ils sont exécutés pour ensuite adapter leurs comportements.

Marque	**Nb. produits**	**Nb. firmwares**	**Nb. firmwares extraits**
360	5	5	ND
Asus	406	406	336 (82.76%)
Belkin	50	49	49 (98%)
D-Link	1603	1611	700 (43.45%)
Edimax	260	220	148 (67.27%)
Linksys	162	129	83 (64.34%)
NETGEAR	3611	3303	2233 (67.61%)
Reolink	43	42	42 (97.67%)
Trendnet	1678	1183	681 (57.57%)
Tp-Link	1464	1357	857 (63.15%)
Ubiquiti	2765	801	767 (95.76%)
Total	12047	9106	5896 (64.75%)

Tableau 5.3 – Répartition par marque du nombre de produits et firmwares téléchargés et extraits

Compte tenu du faible nombre de firmwares téléchargés pour la marque 360, nous n'avons pas analysé cette dernière.

Ensuite, nous utilisons l'approche décrite en section 5.3.2.1 avec Binwalk en version 2.2.0 et avons extrait 5 896 (64.75%) firmwares dont la répartition par marque est visible dans le tableau 5.3. Ce nombre réduit s'explique par :

— le fait que certains firmwares sont chiffrés.

— Binwalk n'arrive toujours pas à identifier le format de donnée du firmware f, et donc échoue à extraire f.

— Binwalk dépasse les 25 minutes allouées à l'opération d'extraction ce qui peut être dû à des firmwares trop volumineux, ou à la présence de données chiffrées

Sur ces 8 301 (68.90%) produits, nous sélectionnons uniquement ceux associés à un firmware publié entre 2009 et 2019 car peu de produits sont disponibles avant 2009 (83 : 0.6%), et après 2019, (292 : 2.4%). Le faible nombre de produits publiés en 2020 s'explique par le fait que nous avons téléchargé les firmwares au mois de mai 2020. De plus, nous ignorons également les 991 (8.2%) produits associés à un firmware dont la date de publication est inconnue. Ainsi, seuls 6 935 (57.5%) produits correspondant à 4 730 (51.9%) firmwares sont analysés. La répartition finale des produits et firmwares analysés est présentée dans le tableau 5.4.

Marque	**Nb. produits analysés**	**Nb. firmwares analysés**
360	0	0
Asus	336 (82.76%)	336 (82.76%)
Belkin	0 (0%)	0 (0%)
D-Link	758 (47.29%)	675 (41.90%)
Edimax	137 (52.69%)	126 (57.27%)
Linksys	97 (59.88%)	83 (64.34%)
NETGEAR	1565 (43.34%)	1563 (47.32%)
Reolink	20 (46.51%)	20 (47.62%)
Trendnet	833 (49.64%)	644 (54.43%)
Tp-Link	657 (44.88%)	589 (43.40%)
Ubiquiti	2532 (91.57%)	694 (86.64%)
Total	6935 (57.57%)	4730 (51.9%)

Tableau 5.4 – Répartition par marque du nombre de produits et firmwares analysés, c'est-à-dire distribués entre 2009 et 2019

rang	**Ensemble des produits**	**Produits analysés**
1	routeur (`appareil réseau`, 29.84%)	routeur (`appareil réseau`, 35.43%)
2	commutateur (`appareil réseau`, 18.66%)	point d'accès sans fil (`appareil réseau`, 15.14%)
3	point d'accès sans fil (`appareil réseau`, 11.98%)	hub (`appareil réseau`, 13.22%)
4	caméra (`objet IdO`, 8.77%)	commutateur (`appareil réseau`, 9.56%)
5	hub (`appareil réseau`, 8.13%)	caméra (`objet IdO`, 6.46%)

Tableau 5.5 – Répartition des catégories les plus représentées dans l'ensemble des produits présents dans la base de données et ceux analysés

Enfin, l'ensemble des 12 047 produits a ensuite été catégorisé à l'aide de l'approche décrite dans la section 5.3.1.2. Dans le tableau 5.5, nous comparons la répartition des catégories et sous-catégories entre l'ensemble des produits et les 6 935 produits analysés. Étant donné le grand nombre de produits catégorisé, il est difficile de vérifier manuellement si les 12 047 ou 6 935 sont correctement catégorisés. Nous avons sélectionné aléatoirement 30 produits puis vérifié manuellement les catégories et sous-catégories inférées et remarquons que 93% des catégories (`appareil réseau` ou `objet IdO`) et 90% des sous-catégories (routeur, hub, caméra, etc.) sont correctement assignées. Globalement, sur les 6 935 produits, 86.08% d'entre eux sont des `appareils réseau`, 12.62% des `objets IdO` et 1.30% `autres`, c'est-à-dire indéfinis.

5.5.2 Mesure du niveau d'exposition des produits

Le niveau d'exposition d'un produit correspond au nombre de binaires présents dans ce dernier, et capable de déployer un des services listés dans le tableau 5.1 de la section 5.2.3. C'est

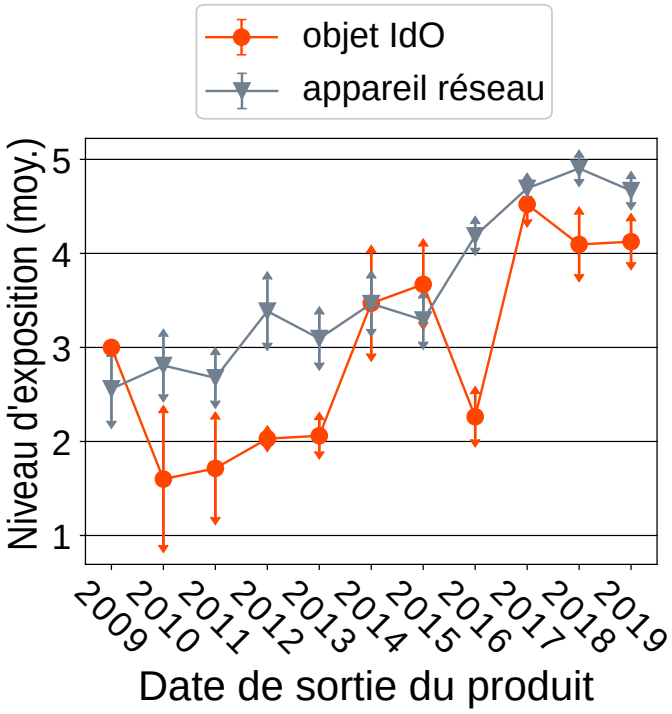


FIGURE 5.7 – Niveau d'exposition des produits appartenant à la catégorie `appareil réseau` ou `objet IdO`. L'intervalle de confiance est configuré à 95%

pourquoi nous pouvons supposer que, plus ce nombre est important, plus le nombre de services déployés par un produit est grand.

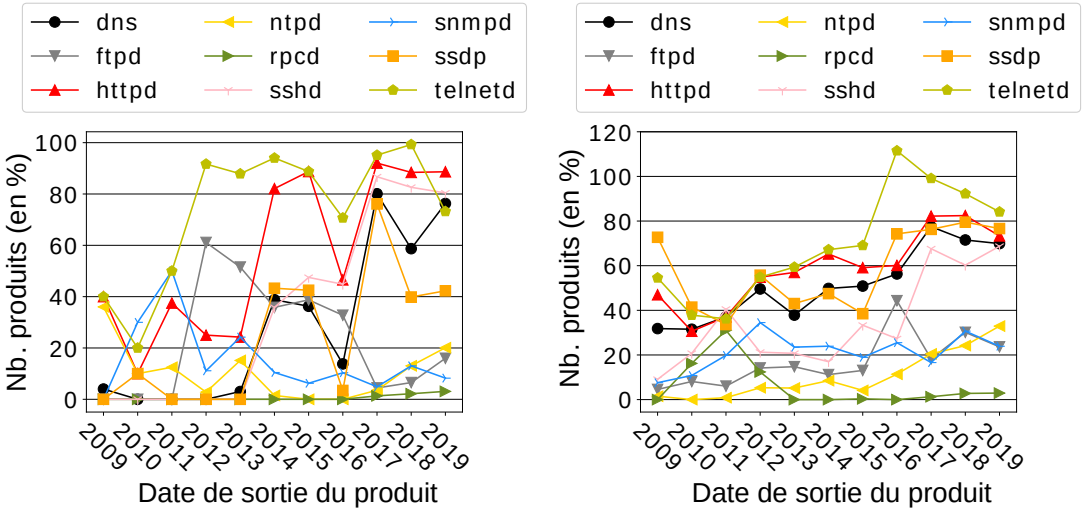
Dans la figure 5.7, nous remarquons qu'entre 2009 et 2019, les deux catégories de produits étudiées voient leurs nombres de services croître et ainsi, deviennent plus exposés aux attaquants. D'un côté, les `objets IdO` voient leur niveau d'exposition augmenter de 3 en 2009 à environ 4 dès 2017 néanmoins, une baisse à 2.26 est observée en 2016. Cette baisse du niveau d'exposition s'explique par une plus grande proportion de la sous-catégorie `caméra` (87.93%) en 2016, alors que globalement cette sous-catégorie ne représente, en moyenne, que 51.20% des `objets IdO`. Ainsi, cela nous permet de conclure que les caméras connectées ne semblent pas suivre la tendance générale des objets IdO consitant à exposer plus de services.

D'un autre côté, les produits catégorisés comme `appareil réseau` voient également leur niveau d'exposition pratiquement doubler jusqu'à atteindre 4.66 en 2019. Ces derniers exposent généralement plus de services que les `objets IdO`.

Sur la période étudiée, les services les plus déployés dans un produit sont TELNET (77.59%), HTTP (66.96%), DNS (63.52%), SSDP/UPnP (54.48%), SSH (53.64%), FTP (22.74%), SNMP (18.83%), NTP (17.71%), RPC (2.42%).

En d'autres termes, 77.59% des produits analysés sont capables de déployer le service TELNET en mode serveur à l'aide d'un binaire comme `telnetd` ou `utelnetd`. On constate que, malgré la sécurité de TELNET, notamment exploité par de nombreux botnets [91, 13, 26], celui-ci est présent dans plus de trois quarts des objets. Nous comparons dans la figure 5.8, la répartition des services disponibles dans les produits catégorisés comme étant des `appareils réseau` ou des `objets IdO`.

Dans la figure 5.8(b) nous observons que le nombre de binaires capables de déployer TELNET



(a) Services dans les `objets IdO`

(b) Services dans les `appareils réseau`

FIGURE 5.8 – Distribution des services suivant la catégorie des produits sur la période 2009-2019

dépasse notamment les 100% en 2016 pour les produits catégorisés comme `appareil réseau`. En effet, certains produits, notamment ceux distribués par NETGEAR, possèdent plusieurs binaires comme `utelnetd` et `telnetd` en même temps. Une observation similaire est notée pour le service SSDP/UPnP pour les marques ASUS ou D-Link. L'utilisation de plusieurs binaires pour déployer un même service est questionnable d'un point de vue sécurité et surtout vis-à-vis de la conception du firmware, étant donné qu'il est plutôt recommandé de fournir uniquement les binaires et librairies partagées permettant de déployer les services prévus. D'ailleurs, d'après l'OWASP, il serait possible de considérer cette observation comme étant un manquement à la gestion du produit ou *Lack of Device Management* en anglais, ce qui correspond à la huitième plus importante vulnérabilité généralement observée dans l'Internet des Objets.

Nous remarquons également dans les figures 5.8(a) et 5.8(b) une augmentation de l'utilisation de Secure Shell (SSH), notamment à partir de 2016. En effet, pour les objets catégorisés comme `objets IdO`, le pourcentage de produits capables de déployer SSH est passé d'environ 30% en 2014 à plus de 80% dès 2017 alors que pour les objets type `appareil réseau`, ce nombre est passé de 10.16% en 2009 à environ 70.31% en 2019.

Cette augmentation de l'utilisation de SSH semble indiquer que les fabricants d'objets connectés ont pris en compte la problématique de la sécurité lors de la conception de leurs firmwares et donc du choix des services déployés dans ces derniers. Cependant, comme décrit ci-avant, TELNET reste tout de même massivement déployé dans les objets.

C'est pourquoi, les récents botnets ciblant les objets de l'Internet des Objets comme Mirai [13, 26, 91] ou Psyb0t [52], utilisent les services SSH ou TELNET pour se connecter à un objet distant. Cependant, ces derniers n'exploitent pas de vulnérabilités associées aux binaires déployant SSH ou TELNET mais effectuent plutôt des attaques par *brute-force* pour trouver l'identifiant et le mot de passe. Néanmoins, nous détaillons dans la section 5.5.4, les vulnérabilités présentes dans les binaires déployant SSH.

Utilisé également pour interagir avec un objet connecté, l'interface web ou service HTTP est de plus en plus déployé dans les objets connectés, dépassant depuis 2017 les 80% des produits

52. Voir `https://en.wikipedia.org/wiki/Psyb0t`, dernier accès le 22/02/2022

catégorisés comme `objets IdO`, et les 60% pour les produits type `appareils réseau`. Cependant, comme décrit dans la section 5.2.3, les attaques ciblant les interfaces web sont souvent liées au binaire déployant ces dernières. C'est pourquoi nous analysons dans la section 5.5.3, les différents binaires et vulnérabilités liés au service HTTP.

5.5.3 Évaluation des serveurs HTTP

À partir de notre base de composition de firmwares, nous observons que 4644 produits (66.96%) possèdent un binaire capable de déployer un serveur HTTP.

Afin de calculer les métriques décrites dans la section 5.4, nous utilisons les expressions régulières listées dans la tableau 5.6 pour extraire les versions des binaires permettant le déploiement d'un serveur HTTP. Seuls les binaires présents dans le tableau 5.6 ont été trouvés dans au moins un produit.

Binaire	**Expression(s) régulière(s)**
`nginx`	`nginx/(\d\.\d{1,2}\.\d{1,2})` et `nginx version: .*`
`Boa`	`Boa/(\d\.\d{1,2}\.\d{1,2}(rc\d{1,2})?)`
`lighttpd`	`lighttpd/(1\.\d\.\d{1,2})`
`Hiawatha`	`Hiawatha v(\d{1,2}\.\d{1,2}(\.\d{1})?)`
`Mongoose`	`version (\d{1,2}\.\d{1,2})`
`thttpd`	`thttpd/(\d{1}\.\d{1,2}[a-zA-Z]?)`
`jhttpd`	`jjhttpd v(\d\.?)+`
`mini_httpd`	`mini_httpd/(\d{1}\.\d{1,2}[a-zA-Z]*)`
`uHTTPd`	`uhttpd/.*`
`shttpd`	`^1\.\d{1,2}$`

Tableau 5.6 – Expressions régulières utilisées pour extraire la version des binaires associés à HTTP

Nous représentons dans la figure 5.9 la distribution des binaires déployant un serveur HTTP dans les produits analysés, le terme *proprietaire* fait référence aux binaires nommés `httpd` dont toutes les tentatives d'extraction de versions ont échoué. D'ailleurs, nous remarquons que le pourcentage de binaires dits *proprietaire* est passé de 36.52 % avant 2015 à seulement 15.61% en 2019. Nous expliquons cette tendance par la forte croissance du marché des objets connectés, notamment les appareils réseau comme les routeurs ou étendeurs wifi. Pour proposer plus rapidement de nouveaux produits sur le marché et réduire les coûts de développement, les fabricants semblent avoir privilégié l'usage de logiciels open source, comme `lighttpd` ou `mini_httpd`, afin de bénéficier des mises à jour proposées par la communauté de ces projets.

De plus, les binaires *proprietaires* souffrant également de nombreuses vulnérabilités, par exemple les CVE-{2018-16119, 2017-14250, 2016-1555} permettant à un attaquant d'effectuer des injections de commandes, l'utilisation de logiciels open source permet de bénéficier de correctifs de sécurité proposés par la communauté et donc, éviter les surcoûts de développement pour les fabricants.

Parmi les projets open source les plus utilisés, nous avons notamment `lighttpd` avec 2 559 (55.10%) produits, `uHTTPd` avec 585 (12.60%) produits puis `mini_httpd` avec 440 (9.47%) produits. Compte tenu de sa surreprésentation, le reste de cette section se focalise sur l'analyse de `lighttpd` où notre approche a réussi à extraire les versions de ces binaires dans 2 544 (99.41%) produits. De plus, ce service web est paramétrable à l'aide de modules permettant d'ajouter des

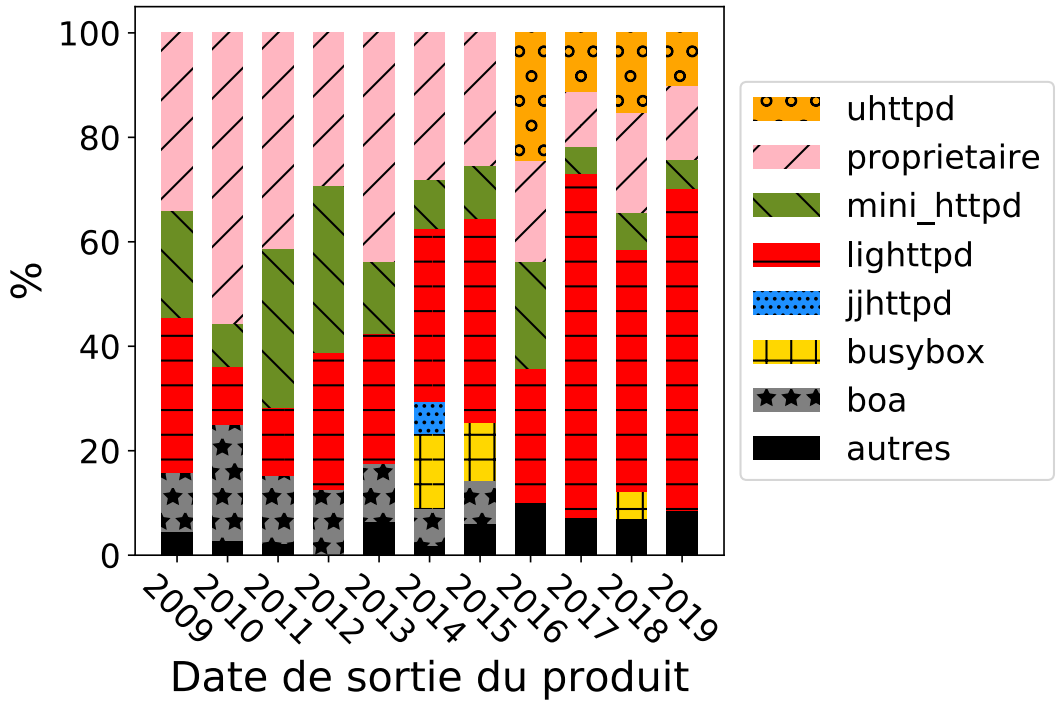


FIGURE 5.9 – Distribution des binaires utilisés pour déployer un serveur HTTP des produits sur la période 2009-2019 (le binaire *autres* regroupe les binaires ayant un pourcentage inférieur à 5%)

fonctionnalités, comme le module `mod_mysql_vhost` pour gérer les interactions avec une base de données.

Dans la figure 5.10, nous avons tout d'abord mesuré le niveau d'obsolescence des binaires de `lighttpd` comme expliqué dans la section 5.4.2. Ainsi, les versions des binaires trouvées dans les produits ont été comparées aux versions connues, et stables, de `lighttpd` extraites depuis le site officiel de `lighttpd`[53]. Nous remarquons que 20 versions différentes de `lighttpd` ont été trouvés dans les produits, parmi lesquelles les trois versions les plus représentées sont v1.4.39 avec 1723 produits (69.08%), v1.4.35 avec 174 produits (6.80%) et v1.4.35 avec 131 produits (5.12%). La version 1.4.39 étant publiée le 2 janvier 2016, cela pourrait expliquer la forte chute des valeurs de δ_{dver} et $\delta_{\#version}$ observée en 2017 dans la figure 5.10, indiquant que les fabricants ont procédé à des mises à jour vers cette version. En effet, en 2017, sur les 1018 produits ayant `lighttpd`, 883 (86.73%) ont effectivement la version 1.4.39.

Sur la période 2009-2019, la valeur de $\delta_{\#version}$ croît d'environ 4 à environ 13, ce qui peut suggérer qu'en réalité les fabricants ne suivent pas nécessaire les dernières mises à jour de `lighttpd`. Or, `lighttpd` étant open source, de nombreuses mises à jour sont proposées au cours d'une année avec, par exemple, 6 versions stables pendant l'année 2016 ou bien, en moyenne, une version tous les 110 jours. Ce qui peut également expliquer pourquoi les valeurs de $\delta_{\#version}$ oscille en moyenne à environ 8 versions. L'augmentation des valeurs de $\delta_{\#version}$ et δ_{dver} après 2017 s'explique par l'utilisation massive de la version 1.4.39 de *lighttpd* pour déployer un serveur HTTP dans les produits.

53. Disponible sur `https://www.lighttpd.net/`, dernier accès le 25/02/2022

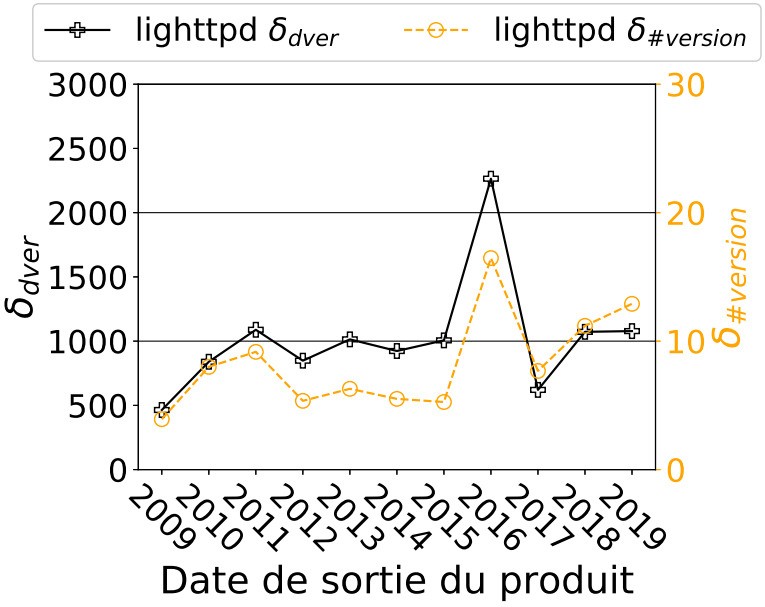


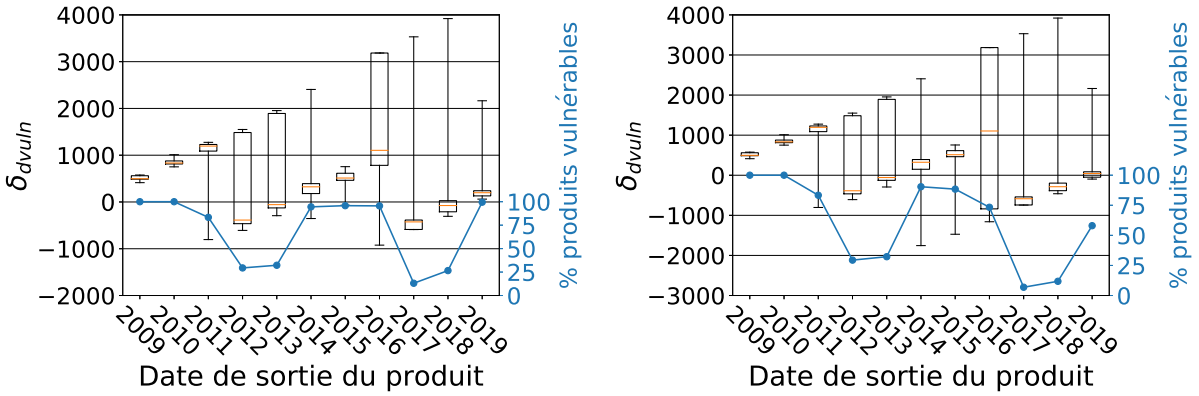
FIGURE 5.10 – Niveau d'obsolescence des binaires `lighttpd`

Une autre raison de mettre à jour un binaire peut être pour corriger une vulnérabilité. C'est pourquoi, nous présentons dans la figure 5.11 les valeurs de notre troisième métrique, décrite en section 5.4.3. Cette dernière, intitulée délai post-vulnérabilité, comptabilise le nombre de jours séparant les dates de publication du firmware et de la première vulnérabilité associée à un binaire trouvé dans ce même firmware.

Nous remarquons premièrement, qu'avec ou sans modules, 17 (100%) des produits distribués avant 2011 ont une version de `lighttpd` inférieure à la v1.4.20 et ainsi, souffrent de la vulnérabilité CVE-2008-4359 permettant à un attaquant d'effectuer des accès illégitimes aux ressources du serveur web. Ainsi, les baisses de la valeur de δ^d_{dvuln} ainsi que du pourcentage de produits vulnérables semblent indiquer que les fabricants ont tenu compte de cette vulnérabilité entre 2011 et 2013 même si 19 (35.18%) produits, sur cette même période, ont toujours une version inférieure à 1.4.20. Cette prise en compte est également remarquable dans la figure 5.10 où entre 2011 et 2012, la valeur de $\delta_{\#version}$ chute de 9 à 5.

La seconde vulnérabilité, CVE-2013-4559, liée aux versions inférieures à 1.4.33 de `lighttd` permet à un attaquant d'effectuer une élévation de privilège. Cette dernière étant publiée le 20 novembre 2013 est remarquable par une δ_{dvuln} positive dans les figures 5.11(a) et 5.11(b) en 2014 où 66 (90.41%) produits souffrent donc de cette vulnérabilité. De plus, nous remarquons qu'entre 2014 et 2019, 391 (15.81%) produits contiennent encore une version de `lighttpd` vulnérable.

Les vulnérabilités suivantes sont liées aux modules pouvant être associés à `lighttpd`, un module étant une fonctionnalité ajoutée au serveur HTTP, comme interagir avec une base de données. Parmi ces dernières, les CVE-2014-2323 et CVE-2015-3200 sont publiées en 2014 et 2015 respectivement. La première cible les versions antérieures à 1.4.35 et permet à un attaquant d'effectuer des injections de commandes via le module gérant les interactions avec une base de données, alors que la seconde vulnérabilité est liée au module d'authentification et cible les versions antérieures à 1.4.36 en permettant un accès illégitime aux logs du produit. Ainsi, il aura fallu entre 2 et 3 ans aux fabricants pour mettre à jour les versions de `lighttpd` vers une version sûre comme la version 1.4.39. Cette observation est visible par les baisses en 2017 des valeurs de



(a) Avec modules

(b) Sans modules

FIGURE 5.11 – Mesure du délai post-vulnérabilité sur les binaires `lighttpd` avec ou sans prise en compte des CVE associées à des modules

δ_{dver} dans la figure 5.10, et de δ_{dvuln} dans les figures 5.11(a) et 5.11(b).

Après 2018, on remarque à l'aide des figures 5.11(a) et 5.11(b) que les modules associés à `lighttpd` rendent ce dernier plus vulnérable, par exemple la CVE-2018-19052 [54] illustrant une attaque par traversée de répertoires. Cependant, la vulnérabilité CVE-2019-11072 publiée initialement le 10 avril 2019 est actuellement contestée. Cette dernière, ciblant toutes les versions inférieures à 1.4.54, permettrait à un attaquant de causer un crash du serveur HTTP via des requêtes HTTP malformées. Néanmoins, notre période d'analyse se terminant en 2019, nous apercevons tout de même la hausse du δ_{dvuln} dans les figures 5.11(a) et 5.11(b). De plus, en 2019 seuls 3 (0.01%) produits ont une version sûre.

De manière générale, nous observons que l'utilisation de `lighttpd` comme serveur HTTP est temporairement sûre. En effet, suivant la configuration du serveur HTTP, notamment avec l'ajout de modules, les risques de sécurité peuvent être accrus d'où l'importance de fournir uniquement les modules nécessaires au bon fonctionnement du service web. Cependant, même sans configuration spécifique, nous montrons dans la figure 5.11(b) qu'il est tout de même nécessaire de régulièrement mettre à jour les versions de `lighttpd` incluses dans les produits. D'ailleurs, du fait que `lighttpd` soit open source, les valeurs de $\delta_{\#version}$ dans la figure 5.10 confirme que de nombreuses versions de `lighttpd` sont souvent proposées par les développeurs de ce projet ainsi, les fabricants pourraient régulièrement effectuer des mises à jour lors de la publication de nouveaux firmwares.

5.5.4 Évaluation des serveurs SSH

Entre 2009 et 2019, 3720 (53.64%) produits possèdent un binaire capable de déployer un serveur SSH. Nous avons ensuite appliqué notre approche pour extraire les versions de ces binaires et avons identifié les binaires et versions dans 3647 (98.04%) produits. Cependant, seuls `dropbear` (3286 : 90.10%) et `OpenSSH` (363 : 9.95%) ont été trouvés. Nous remarquons qu'il existe plus de binaires que de produits ce qui, après une inspection manuelle, est dû à deux produits ayant les deux binaires dans leurs firmwares. Les expressions régulières utilisées sont listées dans le tableau 5.7.

54. Depuis le 18/01/2022, une seconde analyse de cette CVE est en cours

Binaire	Expression régulière
`dropbear`	`v?((20\d2|0)\.((\d2)(-(0\.)?\d1,2)?))$`
`OpenSSH`	`(OpenSSH_(\d\.)+(\d)(p\d)?)$`

Tableau 5.7 – Expressions régulières utilisées pour extraire la version des binaires associés à SSH

À partir des versions extraites, nous calculons l'obsolencence de ces deux binaires et présentons les résultats dans la figure 5.12. Similairement à la section précédente, nous utilisons les versions stables des binaires `dropbear` et `OpenSSH` pour les mesures du niveau d'obsolescence. Pour ce faire, les versions stables de `dropbear` ont été extraites à partir de son site officiel[55] alors que les versions stables de `OpenSSH` ont été obtenues à partir du miroir de téléchargement français[56].

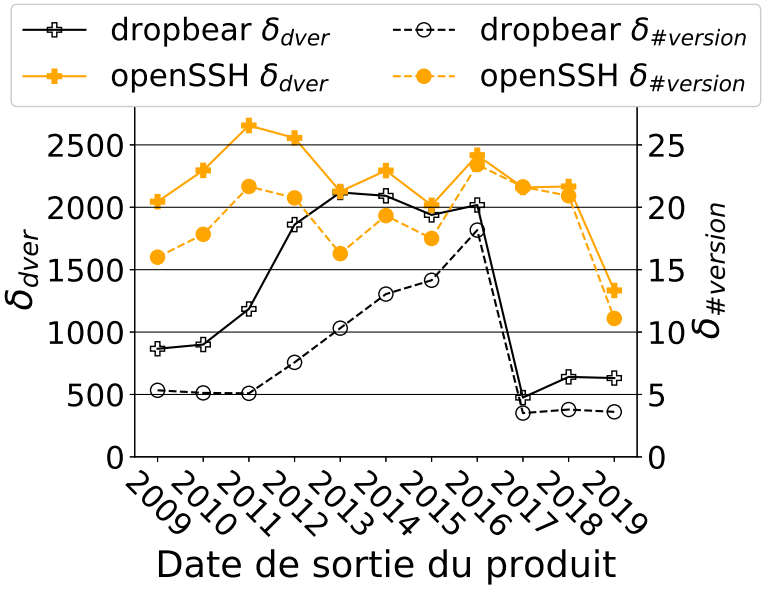


FIGURE 5.12 – Niveau d'obsolescence des binaires `dropbear` et `OpenSSH`

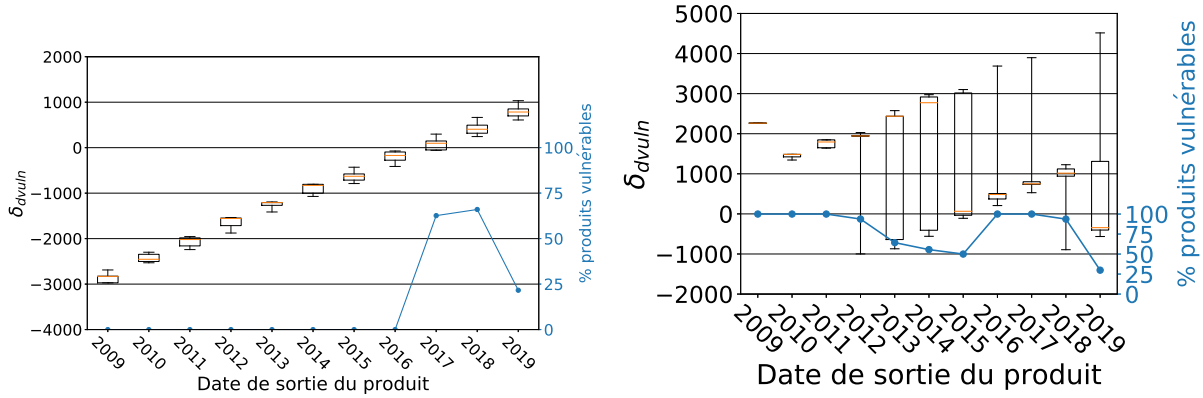
À partir de la figure 5.12, nous observons globalement qu'entre les années 2016 et 2017, une vague de mise à jour est effectuée sur les binaires `dropbear` et `OpenSSH`.

En effet, les 161 (4.41%) produits déployant un binaire `OpenSSH` jusqu'en 2016 ont, en moyenne, une valeur de δ_{dver} d'environ 2 300 jours, ou 6 ans, alors que la valeur de $\delta_{\#version}$ est d'environ 19 versions indiquant donc que de nombreuses nouvelles versions d'`OpenSSH` étaient en réalité disponibles en téléchargement. Cependant, de 2017 à 2019, les moyennes des valeurs de δ_{dver} et $\delta_{\#version}$ baissent à environ 1 886 jours, ou 5 ans, et 17.88 versions. C'est seulement en 2019 que les valeurs les plus faibles de δ_{dver} et $\delta_{\#version}$ sont observées avec 1 333.42 jours et 11.09 versions. Néanmoins, une version de `OpenSSH` est, en moyenne, publiée tous les 109 jours ainsi, on remarque que, même en 2019, les fabricants ont certes mis à jour les binaires `OpenSSH` mais avec des versions antérieures à 2017 comme les versions OpenSSH_7.4p1 (2016) ou OpenSSH_6.7p1 (2014).

55. Disponible sur `https://matt.ucc.asn.au/dropbear/releases/`, dernier accès le 28/02/2022
56. Disponible sur `https://ftp.fr.openbsd.org/pub/OpenBSD/OpenSSH/portable/`, dernier accès le 28/02/2022

De manière similaire, les valeurs de δ_{dver} et $\delta_{\#version}$ de `dropbear` ont augmenté jusqu'à atteindre environ 2018 jours, ou 5 ans, et 18 versions en 2016. Contrairement à `OpenSSH`, une baisse considérable de ces deux valeurs est perceptible entre 2016 et 2017. En effet, à partir de 2017, les valeurs de δ_{dver} et $\delta_{\#version}$ sont en moyenne d'environ 582 jours, ou un an et demi, et de 3.63 versions respectivement. Or, les développeurs de `dropbear` proposent une version en moyenne tous les 99 jours. Il semble donc qu'à partir de 2017, les fabricants déploient une version de `dropbear` datant de l'année antérieure. Cette supposition est notamment vérifiée en 2019 où 406 produits (48.50%) proposent les versions 2018.76 (390 : 96.06%) ou 2019.78 (16 : 3.94%) distribuées respectivement en 2018 et 2019.

Une première explication plausible aux mises à jour de `dropbear` et `OpenSSH` observées entre 2016 et 2017, serait la présence de nombreux botnets [13, 26, 91] actifs durant cette période obligeant les fabricants à améliorer la sécurité de leurs produits.



(a) `dropbear` (b) `OpenSSH`

FIGURE 5.13 – Mesure du délai post-vulnérabilité sur les binaires `dropbear` et `OpenSSH`

La seconde explication serait due aux vulnérabilités présentes dans ces binaires. C'est pourquoi, nous mesurons le délai post-vulnérabilité dans la figure 5.13. À partir de la figure 5.13(a), nous déduisons par les valeurs positives de δ_{dvuln} entre 2009 et 2016 que les versions de `dropbear` présentes dans les 432 (11.84%) produits disponibles sur le marché durant cette même période étaient sûres.

Cependant, l'apparition des premières vulnérabilités se fait sentir à partir de 2017 avec notamment les vulnérabilités CVE-2016-7406 et CVE-2017-9078 publiées respectivement le 3 mars 2017 et le 5 mai 2017. Ces deux vulnérabilités permettent à un attaquant d'effectuer des injections de commandes. Les versions de `dropbear` antérieures à 2016.74 souffrent de la CVE-2016-7406 et/ou CVE-2017-9078, et les versions strictement inférieures à 2017.75 sont ciblées par la CVE-2017-9078. Compte tenu de la date de publication et des versions ciblées, il est normal de voir que 777 (62.61%) des produits distribués en 2017 soient vulnérables. D'un autre côté, les mises à jour observées entre 2016 et 2017, mises en évidence par la baisse des valeurs de δ_{dver} et $\delta_{\#version}$ dans la figure 5.12, ont visiblement été prévues pour corriger la CVE-2016-7406 étant donné que 999 (80.50%) produits avaient la version 2016.74 de `dropbear` et donc n'étaient pas sujets à cette vulnérabilité. La seconde vulnérabilité CVE-2017-9078 publiée seulement 2 mois plus tard aura eu raison de la sécurité des produits distribués en 2017. L'année suivante 514 (65.98%) produits restent vulnérables à au moins une de ces deux CVE. Ce n'est qu'en 2019 que certains fabricants déploient de nouvelles versions exemptes de vulnérabilités, comme les versions 2018.76 ou 2019.78. C'est pourquoi, seuls 181 (21.68%) produits restent vulnérables aux vulnérabilités

présentées ci-avant.

Contrairement à `dropbear` où la première vulnérabilité n'apparaît qu'en 2017, les 55 (15.15%) produits utilisant `OpenSSH` de 2009 à 2011 souffrent d'au moins une vulnérabilité. En effet, les versions d'`OpenSSH` déployées avant 2012, comme OpenSSH_3.7.1 ou OpenSSH_3.8.1p1, datent de plus de 6 ans et par conséquent, souffrent de deux vulnérabilités : CVE-2003-0695 et CVE-2006-5794. La première est publiée le 6 octobre 2003 et décrit une attaque par déni de service ou une injection de commandes à cause d'une mauvaise gestion de la mémoire effectuée par le binaire si sa version est inférieure ou égale à OpenSSH_3.7.1. La seconde vulnérabilité, publiée le 8 novembre 2006, cible les versions d'`OpenSSH` inférieures ou égales à OpenSSH_4.4, et permet de contourner le processus d'identification pour se connecter.

Les valeurs de δ_{dvuln} dans la figure 5.13(b) illustrent deux vagues de mises à jour, la première entre 2012 et 2015, puis la seconde à partir de 2018. La première vague, également visible dans la figure 5.12, fait baisser le pourcentage de produits vulnérables de 93.75% en 2012 à 50% en 2015. Sur les 20 (5.5%) produits `OpenSSH` distribués au cours de 2015, 10 (2.75%) embarquaient la version sécurisée OpenSSH_6.0p1. Or, la CVE-2015-5600 est publiée la même année et décrit une attaque pouvant causer un déni de service pour les versions d'`OpenSSH` inférieures à OpenSSH_7.0.

Les 48 (13.22%) produits déployant `OpenSSH` en 2016 ont encore une version vulnérable à cette CVE. C'est seulement deux ans plus tard, à partir de 2018, que la deuxième vague de mises à jour introduit une nouvelle fois des versions d'`OpenSSH` sécurisées. En effet, de 2018 à 2019, le nombre de produits vulnérables chute de 74 (93.67%) produits à 23 (29.87%) produits. Nous remarquons que les valeurs de δ_{dvuln} en 2019 illustrent que certains produits sont toujours vulnérables à de très anciennes vulnérabilités (CVE-2006-5794 ou CVE-2015-5600) malgré 54 (70.13%) produits avec une version sûre.

D'après notre analyse, les fabricants utilisant `dropbear` dans leurs produits considèrent les risques de sécurité liés à ce binaire dans leurs produits. Malgré des valeurs élevées de δ_{dver} et $\delta_{\#version}$ jusqu'en 2016, ces dernières se justifient par la non présence de vulnérabilité. Dès les premières vulnérabilités en 2017, les fabricants ont, au fur et à mesure, effectué des mises à jour jusqu'à atteindre les 78.32% de produits avec des versions de `dropbear` sûres en 2019. Les 363 (9.95%) produits utilisant `OpenSSH` comme serveur SSH souffrent dès 2009 d'anciennes vulnérabilités (CVE-2003-0695 ou CVE-2006-5794), visibles dans la figure 5.13(b) par les valeurs de δ_{dvuln} supérieures à 2000. Cela reflête donc que certains fabricants ne prennent que très peu en compte les vulnérabilités associées à `OpenSSH` et continuent à distribuer leur produit avec des versions non sûres. Enfin, et similairement à `dropbear`, la dernière vague de mises à jour est également effectuée en 2019, majoritairement par un fabricant, où la majorité des produits (54 : 70.13%) déploie une version sécurisée d'`OpenSSH` même si, d'après la figure 5.12, les versions distribuées en 2019 ont une date de publication antérieure à 2016.

5.6 Synthèse

Dans ce chapitre, l'objectif était d'étudier la sécurité d'objets IdO à partir de la composition de leur firmware, c'est-à-dire les binaires permettant de déployer un service, sans pour autant détecter de nouvelles vulnérabilités dans les firmwares. En effet, nous nous focalisons plutôt sur les tendances de développement de firmwares employées par les fabricants d'objets IdO.

Pour ce faire, notre contribution comprend plusieurs étapes. La première est la construction d'un ensemble de données de firmwares d'objets IdO. La seconde étape extrait les fichiers et binaires présents dans les firmwares téléchargés, puis est suivie par l'étape effectuant l'analyse de

ces données qui, est confrontée à de nombreux challenges comme les diverses architectures processeur utilisées dans l'Internet des Objets, ou l'absence de composants matériels (`/dev/nvram`, `/dev/mem`) empêchant l'exécution de certains binaires. Notre contribution répond à ces problèmes en proposant d'une part une analyse statique où le contenu textuel d'un binaire ou fichier est analysé, et d'autre part, une analyse dynamique exécutant partiellement un binaire.

À partir des informations obtenues par notre approche, nous avons présenté les trois métriques permettant d'établir des tendances vis-à-vis du développement des firmwares d'objets IdO. La première métrique mesure le nombre de services que peut déployer un objet IdO ainsi, plus sa valeur est élevée plus un attaquant a de possibilités pour compromettre cet objet. Nous mesurons également le niveau d'obsolescence des binaires, c'est-à-dire nous vérifions si ces derniers sont régulièrement mis à jour avant d'être distribués dans des firmwares. Cependant, un binaire non à jour n'implique pas forcément que ce dernier est vulnérable. C'est pourquoi, notre troisième métrique vérifie si un binaire présent dans un firmware souffre d'une vulnérabilité connue au moment où le firmware est mis à disposition sur le site du fabricant. Dans notre étude, nous nous sommes basés sur la base de données de vulnérabilités CVE.

Nous avons expérimenté notre approche sur 4 730 firmwares associés à 6 935 objets IdO distribués entre 2009 et 2019 par de grands fabricants comme D-Link, NETGEAR ou ubiquiti. Durant cette période, nous avons noté que les objets IdO étudiés ont un nombre croissant de services dont certains sont connus pour être utilisés par des attaquants comme TELNET, SSH et HTTP. C'est pourquoi nous avons par la suite, focalisé notre analyse sur les binaires associés à HTTP et SSH. Pour ces deux services, nous avons remarqué une utilisation massive de binaires open source avec par exemple `lighttpd` ou `mini_lighttpd` pour le serveur HTTP, ou `dropbear` pour le serveur SSH. D'ailleurs, nous expliquons cette utilisation de binaires open source par le fait que les fabricants d'objets IdO peuvent bénéficier des nombreuses mises à jour, dont des correctifs de sécurité, proposées par la communauté, et ainsi réduire leur coût de développement et donc proposer plus rapidement des objets IdO sur le marché.

Malgré l'utilisation de binaires open source, nos mesures du niveau d'obsolescence sur les binaires `lighttpd`, `dropbear` et `OpenSSH` montrent que ces derniers ne sont pas régulièrement mis à jour et même, dans certains produits, les versions déployées dataient de plus de 6 ans. À titre de comparaison, 1) un utilisateur de smartphone prend moins d'une semaine pour appliquer une mise à jour disponible sur l'une de ses applications mobile [125], 2) alors qu'un ingénieur logiciel ou sécurité applique les mises à jour disponibles dans un délai de 18 à 24 jours [118]. De plus, les valeurs du délai post-vulnérabilité, ont montré malheureusement que certains objets IdO étaient distribués avec une ou plusieurs vulnérabilités connues, comme les objets utilisant `OpenSSH` de 2009 à 2011. Néanmoins, à partir de 2016 notamment, les fabricants ont entrepris de nombreuses mises à jour de leurs binaires vers des versions plus récentes, et souvent sécurisées. Il est intéressant de noter qu'entre les années 2016 et 2017, de nombreux botnets comme Mirai [91, 13, 26], IoTroop [57], etc. étaient actifs et ciblaient les spécifiquement les objets IdO. Ainsi, même si ces derniers ont pu motiver les fabricants à mettre à jour certains binaires, les mises à jour pour corriger les problèmes de sécurité sont bien souvent effectuées une à deux années suivant la publication d'une vulnérabilité, par exemple des versions vulnérables de `dropbear` sont toujours présentes dans des firmwares distribués deux ans après la publication des vulnérabilités.

57. Voir `https://research.checkpoint.com/2017/iotroop-botnet-full-investigation/`, dernier accès le 01/03/2022

Chapitre 6

Identification active d'un objet IdO

Sommaire

6.1 Introduction

Le nombre d'objets de l'Internet des Objets (IdO) connectés à l'Internet est en forte croissance, et estimé à plus de 43 milliards d'ici 2023 [58]. Cette croissance s'effectue au détriment de la sécurité. Similairement au chapitre précédent, nous nous focalisons sur les systèmes embarqués incluant donc les objets IdO comme des prises et des caméras connectées, ou bien des équipements réseau comme des routeurs, passerelles, etc. Dans ce chapitre, nous utilisons les expressions objets IdO et systèmes embarqués indistinctement.

À partir de 2014, de nombreux botnets [91, 13, 107], comme Bashlite [59], Mirai ou BrickerBot [60], ont focalisé leurs attaques sur les objets IdO connectés à l'Internet. Avant toute attaque, une phase de reconnaissance doit être effectuée. Des outils spécifiques existent pour cela, comme ceux présentés dans la section 1.2.2 du chapitre 1. Cette phase de reconnaissance permet aux

58. Disponible sur `https://www.mckinsey.com/industries/private-equity-and-principal-investors/our-insights/growing-opportunities-in-the-internet-of-things`, dernier accès le 12/04/2022
59. Voir `https://digital.nhs.uk/cyber-alerts/2018/cc-2557`, dernier accès le 12/04/2022
60. Voir `https://digital.nhs.uk/cyber-alerts/2017/cc-1322`, dernier accès le 12/04/2022

attaquants d'obtenir des informations sur les objets connectés, et surtout de repérer les objets à attaquer.

Obtenir des informations précises sur un objet connecté est souvent difficile. C'est pourquoi, les botnets listés utilisent généralement des attaques de type *brute-force*, notamment sur les identifiants et mots de passe de connexion. Cela génère de nombreuses interactions avec l'objet ciblé, ce qui réduit la furtivité de l'attaque. Un attaquant doit, dans ces conditions, minimiser ses interactions avec l'objet ciblé tout en maximisant ses chances d'exploiter une ou plusieurs vulnérabilités spécifiques.

Dans ce chapitre, nous nous positionnons du point de vue d'un attaquant et proposons une méthode de fingerprinting active [188] ayant comme objectif de déduire des propriétés sur un objet IdO à partir d'un scan de port TCP/UDP initial. Notre méthode utilise des algorithmes de classification *Random Forest* entraînés à partir des données issues de firmwares d'objets IdO. Excepté le scan de ports, **les déductions sont effectuées sans interagir avec l'objet ciblé**, et identifient certaines propriétés, comme le nom et la version du binaire déployant un service web ou bien la marque de l'objet ciblé.

Le chapitre est découpé en trois sections. Dans la section 6.1, nous motivons notre choix d'utiliser les données de firmwares d'objets IdO afin d'établir une méthode de fingerprinting. Dans la section 6.2, nous définissons des propriétés à inférer, puis nous présentons notre méthode de fingerprinting. Enfin, dans la section 6.3, nous présentons les performances de notre méthode, puis étudions ses limites.

6.2 De l'analyse de firmware au fingerprinting actif

Dans cette section, nous motivons notre choix d'implémenter une méthode de fingerprinting active à partir des données issues d'une analyse de firmware. Dans la section 6.2.1, nous étudions les avantages à utiliser les données d'une analyse de firmwares, puis présentons dans la section 6.2.2 les objectifs de notre méthode de fingerprinting.

6.2.1 Motivation

Un attaquant peut utiliser l'une des méthodes de fingerprinting actives présentées dans la section 1.2.2 du chapitre 1 de l'état de l'art.En plus de pouvoir être détecté par des systèmes de détection d'intrusion (IDS) ou bloqué [61] par des règles de pare-feu, ces méthodes ne retournent pas impérativement ou complètement ces deux informations.

Étendre les méthodes existantes à de nouveaux couples <binaire, version_binaire> pose un problème de passage à l'échelle. Nous avons vu dans le chapitre 5, que de nombreux couples <binaire, version_binaire> étaient utilisés dans l'IdO pour déployer les services, comme HTTP ou SSH.

Nous avons également montré dans le chapitre 5 que l'analyse de firmwares d'objets IdO permettait d'obtenir la liste des couples <binaire, version_binaire> ainsi que les identifiants et mots de passe des utilisateurs. Nous détaillons dans la section 6.2.2, comment un attaquant pourrait utiliser des données extraites d'une analyse de firmwares d'objets IdO afin d'élaborer une méthode de fingerprinting active.

61. Voir `https://wiki.ipfire.org/configuration/firewall/blockshodan`, dernier accès le 24/03/2022

6.2.2 Objectifs et hypothèse

Afin d'élaborer une méthode de fingerprinting active à partir des données extraites par une analyse de firmwares, il est nécessaire de poser une hypothèse sur le fonctionnement des objets IdO par rapport au contenu de leurs firmwares. Nous supposons que les binaires présents dans un firmware, et capables de déployer un service, sont exécutés par l'objet IdO associé au firmware en utilisant les ports par défaut. Par exemple, un objet IdO dont le firmware contient les binaires `lighttpd` et `telnetd` déploie un serveur HTTP et TELNET sur les ports 80 et 23. L'intêret de cette hypothèse est donc de supposer les résultats obtenus par un scan de ports TCP/UDP. Dans le reste de ce chapitre, nous utiliserons l'expression services actifs pour faire référence aux binaires présents dans un firmware et capables de déployer un service. En complément des services actifs, l'analyse de firmware permet également de connaître les noms d'utilisateurs (identifiants) et mots de passe définis par défaut dans chaque objet IdO.

Notre méthode de fingerprinting a pour objectifs de déduire les propriétés suivantes :

O1 la marque de l'objet. Un attaquant pourrait déduire de cette information les identifiants et mots de passe propres à certaines marques, comme l'identifiant `ubnt` et mot de passe `ubnt` utilisé par la marque Ubiquiti.

O2 Le nom du binaire déployant un des services actifs de l'objet. À partir de cette valeur, un attaquant pourra privilégier les attaques ciblant spécifiquement ce binaire, et donc augmenter les chances de réussite de ses attaques.

O3 La version du binaire déployant un des services actifs de l'objet. Cette valeur est inférée seulement après avoir trouvé le nom du binaire. Par conséquent, un attaquant, à partir d'un couple <binaire, version_binaire>, pourra sélectionner la vulnérabilité connue la plus appropriée aux objectifs de son attaque.

O4 Les identifiants utilisables sur l'objet, c'est-à-dire les noms d'utilisateurs possédant un compte. Contrairement aux attaques par *brute-force* sur les couples <utilisateur, mot de passe>, inférer la liste des noms d'utilisateurs disponibles sur un objet connecté permet à l'attaquant de réduire le nombre de tentatives et ainsi, gagner en furtivité.

O5 Les mots de passe associés aux utilisateurs de l'objet. Ainsi, en complément de **O4**, l'attaquant aura à sa disposition une liste de couples <utilisateur, mot de passe> de taille inférieure à celle utilisée par des botnets, comme [91, 13, 107]. Par conséquent, les tentatives de connexion vers l'objet ciblé seront moins nombreuses et donc également plus furtives.

Notre méthode de fingerprinting met en lumière la véracité des risques de sécurité présentés dans le dernier classement proposé par l'*Open Web Application Security Project (OWASP)* en 2018 [62]. Par exemple, inférer *facilement* les identifiants et mots de passe (**O4** et **O5**) correspond au premier risque de ce classement, nommé *Weak, Guessable, or Hardcoded Passwords*. Déduire le nom et la version d'un binaire (**O2** et **O3**) peut refléter que ces derniers ne sont pas suffisamment mis à jour, et correspond au cinquième risque du classement intitulé *Use of Insecure or Outdated Components*.

62. Disponible sur `https://owasp.org/www-pdf-archive/OWASP-IoT-Top-10-2018-final.pdf`, dernier accès le 25/03/2022

6.3 Identification des propriétés d'un objet IdO à partir d'informations partielles

Cette section présente le cas d'utilisation de notre méthode au cours de la phase de reconnaissance, puis décrit la construction de notre base de données des compositions de firmwares, et son utilisation par des algorithmes d'apprentissage automatique pour la prédiction de propriétés associées à un objet IdO connecté.

6.3.1 Schéma d'attaque étudié

Avant d'effectuer une attaque, un attaquant doit premièrement trouver une ou plusieurs cibles potentielles. À l'aide d'outils de scans de ports, un attaquant peut obtenir des informations sur une machine connectée à travers l'Internet. Ces informations sont souvent incomplètes car elles se limitent généralement à la liste des services actifs avec, dans certains cas, le nom et la version des binaires déployant ces services. Dans le reste de ce chapitre, nous utilisons l'expression information partielle pour faire référence à l'ensemble des informations qu'un attaquant est capable d'inférer d'une machine connectée. De plus, afin de maximiser sa furtivité, un attaquant se doit de minimiser ses interactions avec la machine ciblée. L'information partielle ainsi collectée contient initialement la liste des services actifs, par exemple les ports 21 et 23 sont ouverts donc la machine déploie un serveur FTP et TELNET. Cette dernière ne suffit pas à choisir une attaque spécifique à un binaire ou à la marque de la machine cible.

C'est pourquoi, il est important pour un attaquant de dériver d'autres propriétés sur la machine ciblée afin de sélectionner les attaques ayant le plus de chance de succès pour compromettre le maximum de machines ciblées, ou une machine spécifique. Pour ce faire, nous considérons que l'attaquant effectue préalablement une analyse à large échelle de firmwares d'objets IdO et dispose d'une large base de données contenant, entre autres, la marque, les couples <binaire, version_binaire> ainsi que les identifiants et mots de passe présents dans les firmwares.

Compte tenu des objectifs et de l'hypothèse présentés dans la section 6.2.2, l'attaquant peut ensuite utiliser la liste des services actifs et les données précédemment listées pour entraîner des algorithmes de classification afin d'inférer des propriétés de plus en plus précises sur l'objet ciblé. Ceci permet ensuite de réduire progressivement l'ensemble des attaques nécessaires. Contrairement aux méthodes de fingerprinting actives traditionnelles, cette approche est réalisée hors ligne par l'attaquant, c'est-à-dire sans interagir de manière continue avec l'objet ciblé.

Dans les sections suivantes, nous détaillons les étapes effectuées par notre approche, et illustrées dans la figure 6.1. La section 6.3.2 détaille l'étape (1) consistant à construire une base de données des compositions de firmwares d'objets IdO. Enfin, la section 6.3.3 présente les algorithmes de classification utilisés pour prédire des propriétés d'un objet IdO, soit les étapes (2), (3) et (4).

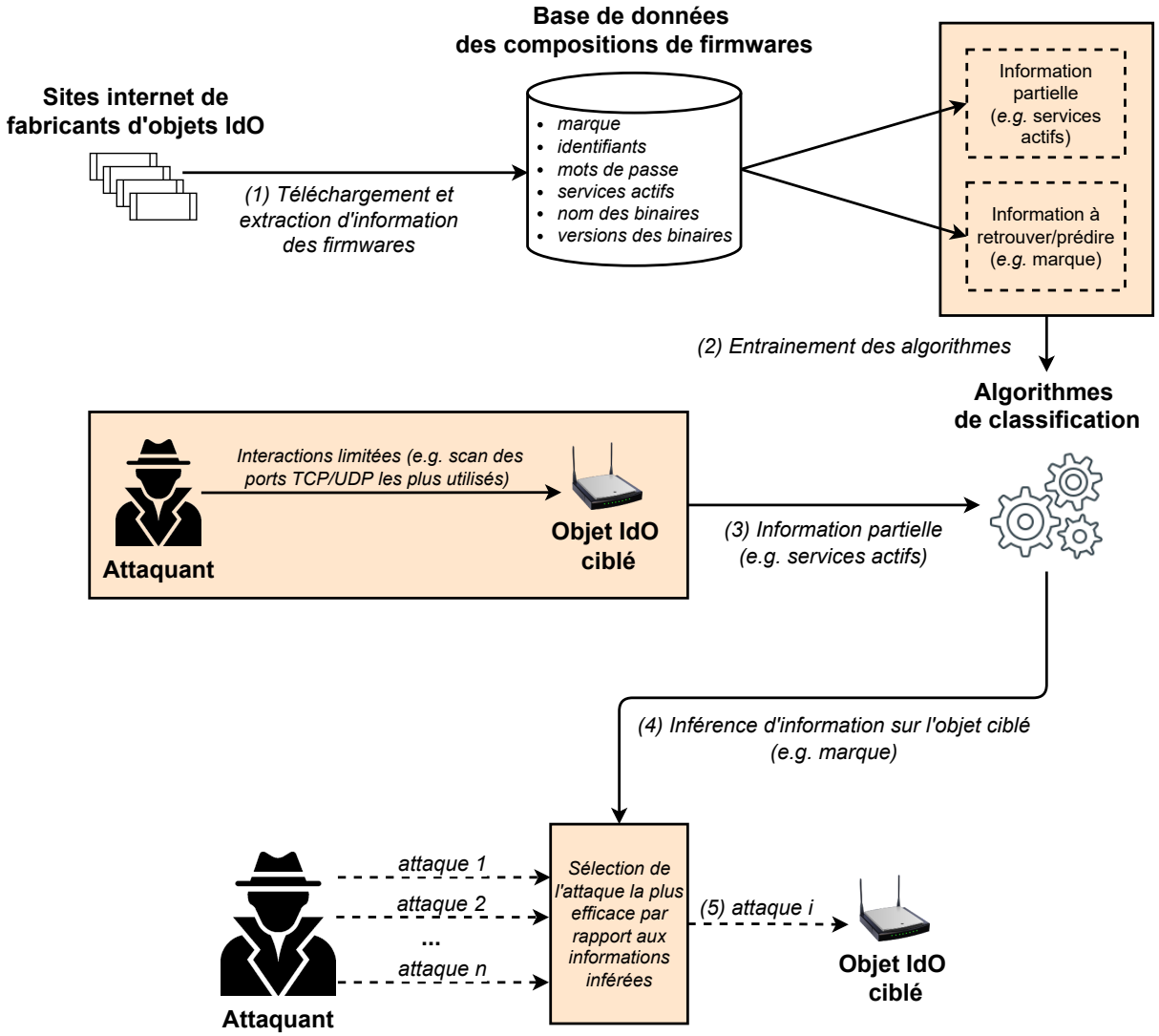


FIGURE 6.1 – Inférence d'informations associées à un objet IdO connecté à l'aide d'algorithmes de classification entraînés par les données obtenues par une analyse de firmwares.

6.3.2 Construction du jeu de données

L'étape (1) de la figure 6.1 consiste à télécharger des firmwares d'objets IdO depuis l'Internet puis d'extraire de ces derniers des informations, comme les binaires présents et leur versions. Ceci permet de construire une base de données des compositions des firmwares à l'aide de la méthode présentée dans le chapitre 5.

Parmi les données présentes dans la base de données des compositions des firmwares, la marque associée à chacun de ces firmwares est directement obtenue lors de l'étape de téléchargement. La marque associée à un firmware d est notée $marque_d$, et sera utilisée pour répondre à l'objectif **O1**.

Lors de la phase de reconnaissance, le scan de ports peut évaluer la présence de centaines de services, ce qui génère autant de requêtes et n'est donc pas furtif. Dans notre cas, nous nous intéressons seulement aux services DNS, FTP, HTTP, NTP, SNMP, SSDP/UPnP, SSH et TELNET car ce sont des services déployés dans l'IdO [36, 45], accessibles depuis l'Internet [182]

et souvent utilisés pour effectuer des attaques. Par exemple, les attaquants peuvent se connecter à un objet distant à l'aide des services TELNET ou SSH, ou exfiltrer des informations associées à un utilisateur d'un serveur FTP. De plus, nous avons détaillé dans la section 5.2.3 du chapitre 5 que les services basés sur le protocole UDP pouvaient également être contournés par un attaquant pour effectuer des attaques par réflexion, voire amplification. Ainsi, le résultat d'un scan de ports TCP/UDP obtenu par un attaquant lors de l'étape (3) de la figure 6.1 peut être représenté sous la forme d'un vecteur binaire, noté S, où la présence de ces huit services est vérifiée. Conformément à l'hypothèse énoncée dans la section 6.2.2, la liste des services déployés par un firmware d, notée S_d, est dérivée des binaires présents dans d.

Il est également nécessaire d'extraire la liste des binaires Sw_d utilisés pour déployer les services dans S_d. Ainsi, le nom sw_{nom} et la version $sw_{version}$ de chaque binaire dans Sw_d est extrait. La liste des binaires associés aux huit services étudiés est présentée dans le tableau 5.1 du chapitre 5. Ces couples $< sw_{nom}, sw_{version} >$ seront notamment utilisés pour répondre aux objectifs **O2** et **O3**.

Nécessaire pour les objectifs **O4** et **O5**, les identifiants et mots de passe présents dans un firmware d sont également extraits par l'analyse de firmwares. Ainsi, le contenu des fichiers dont le nom est similaire à `/etc/passwd*` ou `/etc/shadow*` est extrait. Nous n'utilisons pas les noms exacts `/etc/{passwd,shadow}` car dans certains firmwares, un suffixe `.bak` peut être présent. Enfin, les mots de passe présents dans ces fichiers peuvent être hachés à l'aide de fonctions de hachage[63] ne permettant pas de connaître directement leurs valeurs non chiffrées.

Les deux objectifs précédents évoquent la possibilité pour un attaquant d'utiliser des identifiants de connexion pour se connecter. Il est donc nécessaire d'inspecter le contenu de ces fichiers afin de garder uniquement les utilisateurs actifs, notés u_d. Dans un système UNIX, les utilisateurs verrouillés, ou non utilisables, ont généralement un symbole comme `"*"` ou `"!"` à la place du mot de passe. À titre d'exemple, les utilisateurs `bin` ou `daemon` sont généralement verrouillés. Par conséquent, seuls les noms (identifiants) u_d et mots de passe pwd_{u_d} associés aux utilisateurs non verrouillés sont considérés. Il est à noter que les mots de passe dans pwd_{u_d} peuvent donc avoir des valeurs chiffrées.

Chaque firmware téléchargé d et analysé de façon à insérer dans la base de données de composition de firmwares :

— sa marque $marque_d$,

— ses services actifs S_d,

— les binaires Sw_d déployant S_d,

— le nom sw_{nom} et la version $sw_{version}$ de chacun de ces binaires,

— les identifiants non verrouillés u_d,

— les mots de passe associés à ces identifiants pwd_{u_d}.

Une fois la base de données construite, il est possible d'entraîner des algorithmes de classification développés dans la section suivante.

6.3.3 Algorithmes de classification

Nous proposons une approche à deux niveaux illustrée dans la figure 6.2.

Le premier niveau, ou *module d'inférence de marque*, est notamment utilisé pour répondre à l'objectif **O1**, et donc déduire la marque à laquelle appartient un objet connecté. Parmi les méthodes possibles, notre choix s'est porté sur l'algorithme de classification *Random Forest* compte

63. Voir `https://fr.wikipedia.org/wiki/Fonction_de_hachage`, dernier accès le 11/04/2022

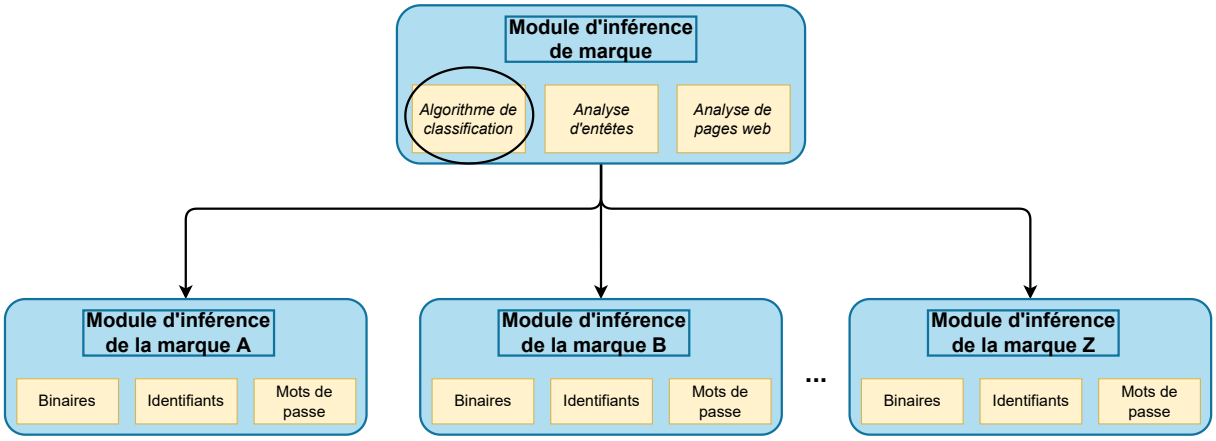


FIGURE 6.2 – Approche à deux niveaux permettant l'inférence de propriétés d'un objet IdO connecté comme sa marque, ses binaires ou identifiants

tenu des explications données dans la section 6.3.1, et de la taille *limitée* de la base de données ne permettant pas d'utiliser efficacement des algorithmes d'apprentissage profond. D'autres méthodes existent pour déduire la marque d'un objet mais souffrent de plusieurs inconvénients. L'analyse d'entêtes nécessite d'inspecter le trafic réseau de milliers d'objets IdO physiques ou émulés, ce qui pose un problème de passage à l'échelle. La marque ou le modèle d'un objet sont généralement directement accessibles depuis les pages web d'un produit ou les bannières de connexion aux serveurs SSH ou TELNET. Cependant, leur extraction peut nécessiter un traitement particulier, et il n'est pas garanti que tous les produits possèdent un serveur HTTP(S), SSH ou TELNET.

Certaines de nos expérimentations requièrent de connaître la marque de l'objet ciblé ainsi, si notre méthode échoue à identifier cette dernière, les méthodes décrites ci-avant peuvent être testées.

Plusieurs marques d'objets IdO peuvent être présentes dans notre base de données. Il est donc nécessaire que notre algorithme de classification, ou classificateur, soit capable de prédire une marque parmi plusieurs disponibles. Il s'agit donc d'une classification multi-classes, ce qui est géré par l'algorithme *Random Forest* dont les performances dans des problèmes similaires sont bonnes [146, 34].

Une fois la marque de l'objet ciblé identifiée, le *module d'inférence* associé à cette marque est utilisé. Ce dernier permet de mettre en lumière les pratiques de développement associées à chaque fabricant, c'est-à-dire les propriétés de leurs objets. Si le classificateur prédit avec une précision de 100% la version de n'importe quel binaire, cela signifie qu'un attaquant sera capable d'exploiter en, potentiellement une seule attaque, l'intégralité des objets d'une même marque.

À l'aide du *module d'inférence* associé à la marque de l'objet ciblé, les informations suivantes sont déduites :

— le nom d'un binaire déployant un des huit services présentés dans la section 6.3.2 et correspondant à l'objectif **O2**,
— la version associée à ce binaire (**O3**),
— les identifiants actifs sur le produit (**O4**),
— les mots de passe associés à ces identifiants (**O5**).

Chacune de ces informations est déduite à l'aide d'un classificateur *Random Forest* mais, contrairement au *module d'inférence de marque*, celui-ci est entraîné uniquement avec les données as-

sociées à la marque du *module d'inférence* durant l'étape (2). De plus, certaines prédictions impliquent de retourner une liste d'éléments ou labels, comme la liste des utilisateurs. Par conséquent, ce type de classification n'est plus multi-classes mais multi-labels. Cette classification est également effectuée à l'aide du classificateur *Random Forest*. Supposons qu'une liste de trois labels $L = \{l_1, l_2, l_3\}$ est à prédire à partir d'un échantillon donné, alors le classificateur principal C va instancier trois *sous* classificateurs binaires *Random Forest* C_1, C_2, C_3 tel que C_i prédit l_i avec $i \in \{1, 2, 3\}$, puis C regroupe et retourne les valeurs prédites par chaque C_i.

6.4 Expérimentation et résultats

Dans cette section, nous détaillons premièrement nos métriques d'évaluation (section 6.4.1), puis détaillons dans la section 6.4.2 de notre base de données de compositions de firmware. Enfin, nous présentons dans la section 6.4.3, les performances de notre approche.

6.4.1 Métriques d'évaluation

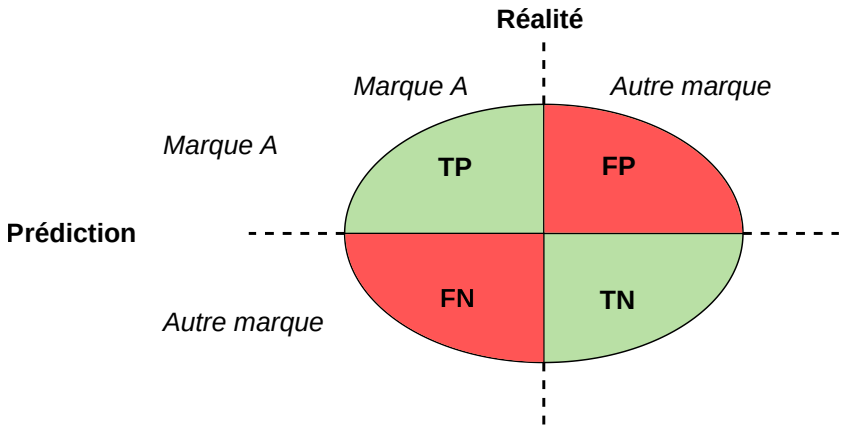


FIGURE 6.3 – Métriques de classification

Les résultats retournés par notre approche sont obtenus par des classificateurs dont les principales métriques sont la précision et le rappel, ou respectivement *precision* et *recall* en anglais. Afin d'expliquer le rôle et la formule pour calculer ces deux métriques, nous utilisons l'exemple illustré dans la figure 6.3 où, à partir des services actifs d'un objet, l'objectif est de prédire si ce dernier appartient à la marque A. Ainsi, une prédiction retournée par le classificateur peut être considérée comme :

TP (Vrai Positif ou *True Positive* en anglais) L'objet est de marque A, et est prédit comme tel.

FP (Faux Positif ou *False Positive* en anglais) L'objet n'est pas de marque A mais est prédit à tort comme appartenant à la marque A.

FN (Faux Négatif ou *False Negative* en anglais) L'objet est de marque A mais est identifié à tort comme n'appartenant pas à cette marque.

TN (Vrai Négative ou *True Negative* en anglais) L'objet n'est pas de marque A, et est prédit à juste titre comme n'appartenant pas à cette marque.

La précision mesure la proportion de prédictions correctes et se calcule à l'aide de l'équation 6.1.

$$\text{précision} = \frac{TP}{TP + FP} \quad (6.1) \qquad \text{rappel} = \frac{TP}{TP + FN} \quad (6.2)$$

D'un autre côté, le rappel est calculé à l'aide de l'équation 6.2 et mesure le ratio de prédictions exactes correctement identifié.

Les formules 6.1 et 6.2 calculent les performances du classificateur pour une classe donnée. Dans l'exemple de la figure 6.3, il s'agit de l'appartenance d'un échantillon à la marque A. Or, comme détaillé dans la section 6.3.3, certains classificateurs sont multi-classes, et donc peuvent associer à un échantillon une classe parmi plusieurs disponibles.

Classe	**Occurrence**	**TP**	**FP**	**TN**	**FN**	**Précision**
C_1	9	9	0	0	0	1.00
C_2	1	0	0	0	1	0.00

Tableau 6.1 – Exemple de résultats d'un classificateur multi-classes

Il est donc nécessaire d'adapter le calcul des deux métriques précédentes [66]. À l'aide des données listées dans le tableau 6.1, nous présentons les deux méthodes de calcul proposées pour tenir compte de ces multiples classes :

(a) la macro-moyenne, ou *macro-average* en anglais, est la moyenne arithmétique des précisions, ou des rappels, de chacune des classes. Elles se calculent par les formules 6.3 et 6.4 où P_i est la précision de la classe i, R_i le rappel de la classe i, et n le nombre de classes disponibles.

$$\text{macro-moyenne précision} = \frac{\sum_{i=1}^{i \leq n} P_i}{n} \quad (6.3) \qquad \text{macro-moyenne rappel} = \frac{\sum_{i=1}^{i \leq n} R_i}{n} \quad (6.4)$$

Ces macro-moyennes permettent donc d'avoir un aperçu global des performances d'un classificateur à prédire correctement plusieurs classes. Cependant, dans le cas d'un ensemble de données déséquilibrées, ou *imbalanced dataset* en anglais, certaines classes peuvent être plus fréquentes que d'autres. Dans ce cas, la valeur de la macro-moyenne peut être biaisée. Par exemple, à partir des données présentées dans le tableau 6.1, la macro-moyenne de ces précisions sera de seulement $\frac{1.0+0.0}{2} = 0.5$, ce qui n'est pas représentatif des performances globales du classificateur étant donné que 90% des valeurs sont correctement prédites.

(b) La micro-moyenne, ou *micro-average* en anglais, n'utilise pas directement les valeurs des précisions et rappels mais se calcule à partir des TN, TP, FN et FP de chacune des classes comme illustrée par les équations 6.5 et 6.6 où n représente le nombre de classes disponibles.

$$\text{micro-moyenne précision} = \frac{\sum_{i=1}^{i \leq n} TP_i}{\sum_{i=1}^{i \leq n} (TP_i + FP_i)} \quad (6.5)$$

$$\text{micro-moyenne rappel} = \frac{\sum_{i=1}^{i \leq n} TP_i}{\sum_{i=1}^{i \leq n} (TP_i + FN_i)} \quad (6.6)$$

À l'aide des données présentes dans le tableau 6.1, la micro-moyenne des précisions sera de $\frac{9+0}{9+0} = 1.0$ soit la valeur optimale, ce qui indique que parmi les valeurs prédites comme appartenant à une marque, le classificateur ne se trompe pas. Cependant, la micro-moyenne des rappels sera de $\frac{9+0}{9+1} = 0.9$ indiquant effectivement que le classificateur *rate* l'élément

de classe C_2. Dans le cas d'un ensemble de données déséquilibrées, la micro-moyenne permet de mieux transcrire les performances globales d'un classificateur indépendamment des prédictions de chaque classe (contrairement à la macro-moyenne).

Les performances de nos classificateurs seront mesurées dans la section 6.4.3 à l'aide de ces deux moyennes sur les précisions et rappels.

Comme abordé dans la section 6.3.3, certaines prédictions sont multi-labels, c'est-à-dire le classificateur va retourner une liste de labels, comme une liste d'identifiants. Du point de vue d'un attaquant, il n'est pas utile de s'assurer que l'ensemble des éléments prédits soit correct, au moins un doit l'être. En effet, lors d'une attaque par *brute-force*, un attaquant essaie de se connecter à l'aide de nombreux couples <utilisateur, mot de passe> mais arrête les tentatives de connexion dès qu'un de ces couples fonctionne. De plus, ce nombre de tentatives doit être suffisamment faible afin 1) d'éviter à l'attaque d'être détectée, et/ou 2) de voir la connexion bloquée.

C'est pourquoi, nous introduisons une métrique permettant justement de mesurer ce nombre de tentatives, ou *trial* en anglais, nécessaire avant de trouver une bonne prédiction dans la liste des éléments retournés. Le calcul de cette métrique dépend du type de classification.

Dans le cas d'une classification multi-classes, supposons que notre classificateur C doit prédire l'une des trois classes suivantes c_1, c_2, c_3 à partir d'un échantillon S. Pour chaque classe c_i, C calcule la probabilité que S appartienne à c_i, notée $P_{c_i}(S)$. À partir des probabilités calculées $P_{c_1}(S), P_{c_2}(S), P_{c_3}(S)$, C sélectionne la classe avec la probabilité la plus élevée. Dans notre cas, les probabilités $P_{c_i}(S)$ sont d'abord triées par ordre décroissant, puis le *trial* correspond au rang de la classe à laquelle appartient réellement l'échantillon S. Par exemple, S appartient à la classe c_3, et notre classificateur C retourne la liste des probabilités triées suivantes : $P_{c_1}(S), P_{c_3}(S), P_{c_2}(S)$, alors le *trial* est de deux car $P_{c_3}(S)$ est en deuxième position dans la liste. Ainsi, $trial = 1$ indique que le classificateur retourne directement la bonne prédiction (classe).

Dans le cas d'une classification multi-labels, le calcul est spécifique à l'implémentation du classificateur multi-labels utilisée qui est, dans notre cas, issue de *scikit-learn*[64], et qui consiste à construire un classificateur pour chacun des éléments à prédire.

Supposons un ensemble de $n \geq 1$ identifiants id_i avec $i \in [1, n]$ à prédire, alors le classificateur multi-labels principal va 1) construire n *sous-classificateurs*, un pour chacun des id_i, puis 2) prédire la présence de chacun des id_i à partir de ces n classificateurs avant 3) d'agréger les résultats.

Or, durant l'étape 2) il est possible de connaître la probabilité d'un *sous-classificateurs* i à prédire le label id_i. Les probabilités retournées par chacun des n *sous-classificateurs* sont triées par ordre décroissant, puis l'indice associé au premier label correctement prédit correspond à notre métrique du nombre de tentatives ou *trial*. Par exemple, $trial = 1$ indique que le classificateur multi-labels prédit directement un label correct alors que $trial = 3$ implique que les deux premiers labels retournés sont incorrects mais pas le troisième ainsi, dans le cas d'une attaque par *brute-force*, équivaut à trois tentatives de connexion dont deux sont infructueuses.

En plus du nombre de tentatives, nous mesurons également si au moins un des labels prédits par le classificateur multi-labels est correct. Dans le cas d'une classification multi-classes, nous vérifions que le classificateur puisse prédire la vraie classe, c'est-à-dire sa probabilité P_{c_i} doit être non nulle.

64. Disponible sur `https://scikit-learn.org/stable/modules/multiclass.html`, dernier accès le 01/04/2022

6.4.2 Base de données de compositions de firmwares

Afin d'entraîner des classificateurs, nous utilisons la base de données des compositions de firmwares décrite dans la section 5.5.1. Celle-ci contient les données extraites de 4 730 firmwares associés à 6 935 objets IdO publiés entre 2009 et 2019. Comme visible dans le tableau 5.4, le nombre de produits par marque est déséquilibré. Les marques Linksys, Reolink et Edimax ne représentant que 4% (254) des produits, nous ne les avons pas maintenus dans la base.

6.4.3 Résultats

Dans cette section, nous présentons les expériences listées dans le tableau 6.2 où est spécifié, pour chacune d'elles, l'information partielle utilisée (•) par le classificateur pour prédire un élément (?). Les résultats détaillés dans le tableau sont obtenus par le classificateur du *module d'inférence de marque*. Pour chacune des expériences détaillées ci-après, une validation croisée par la méthode *k-fold* avec $k = 4$ a été appliquée.

Objectif			1	2	3	2	3	2	3	4		5
Eval id.			1	2	3	4	5	6	7	8	9	10
Section			6.4.3.1	6.4.3.2		6.4.3.3		6.4.3.4		6.4.3.5		6.4.3.6
Marque $marque_d$			?	•	•	•	•	•	•		•	•
Services S_d			•	•	•	•	•	•	•	•	•	•
Liste des binaires sw_d	HTTP	nom		?	•							
		version			?							
	SSH	nom				?	•					
		version					?					
	DNS	nom						?	•			
		version							?			
Identifiants u_d										?	?	
Mots de passe pwd_{u_d}												?
Micro-moyenne précision (%)			76.86	89.56	94.40	98.68	80.29	99.23	73.14	88.93	92.57	90.33
Macro-moyenne précision (%)			76.85	75.31	63.75	95.73	61.89	94.47	44.12	43.77	58.82	36.37
Micro-moyenne rappel (%)			76.86	87.28	88.81	98.68	74.54	98.95	64.23	78.90	93.02	86.32
Macro-moyenne rappel (%)			66.57	68.19	62.97	96.83	51.77	88.99	40.44	34.12	55.50	33.67
Nb. tentatives en moyenne			1.35	1.12	1.13	1.01	1.30	1.01	1.44	1.09	1.03	1.15

Tableau 6.2 – Récapitulatif de nos expérimentations. ? représente l'information à prédire à partir de l'information partielle symbolisée par •

6.4.3.1 Inférence de la marque

Notre première évaluation (Eval. 1) est effectuée à l'aide du classificateur du module d'inférence de marque présenté dans la section 6.3.3, il répond à l'objectif **O1** consistant à inférer la marque $marque_d$ d'un objet IdO d à partir des services actifs sur ce dernier S_d. En d'autres termes, un attaquant effectue un scan de ports TCP/UDP limité aux huit services étudiés, puis utilise ces résultats pour déduire la marque de l'objet ciblé.

Notre classificateur a, respectivement, une macro-moyenne précision et rappel de 76.85% et 66.57%. Enfin, comme illustré dans la figure 6.4, les performances varient entre les marques. En effet, les marques **A** et **E** sous-performent avec une précision et rappel moyen inférieurs à 61%, et donc ne peuvent être clairement identifiées avec notre classificateur. Les marques **B**, **C** et **F** ont une précision moyenne supérieure à 89.6% mais le rappel varie entre 57.8% et 92.6%. Cela reflète un grand nombre de faux négatif, c'est-à-dire notre classificateur identifie à tort la marque de

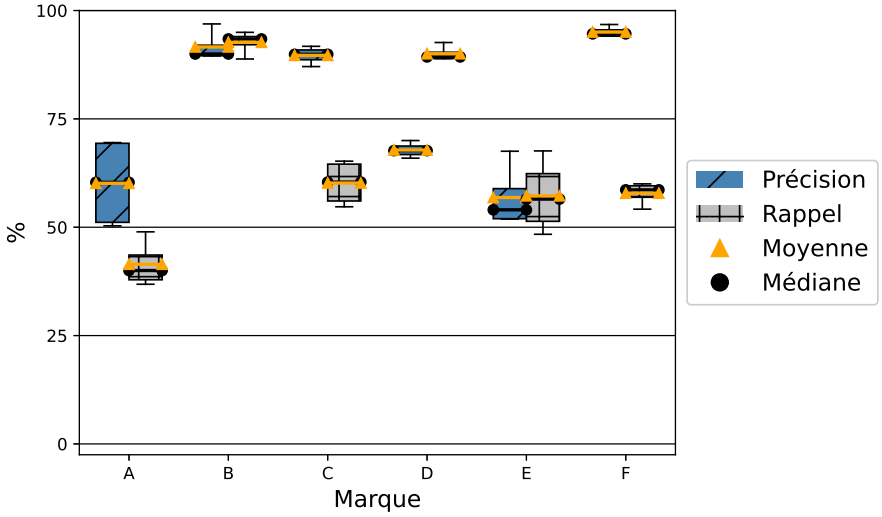


FIGURE 6.4 – Inférence de la marque d'un objet connecté à partir de ses services actifs

certains objets. De plus, comme l'indique le rappel moyen supérieur à 90%, notre classificateur rate peu d'instances des marques **B** et **D**, ce qui laisse penser que les objets déployés par ces marques ont des services actifs particuliers. Malgré qu'un attaquant ne puisse prédire précisément chacune des marques, la valeur moyenne du *trial* (1.35) indique que le classificateur retourne la bonne marque dans les deux premières prédictions, et donc un attaquant peut tout de même réduire l'ensemble des marques de six à deux.

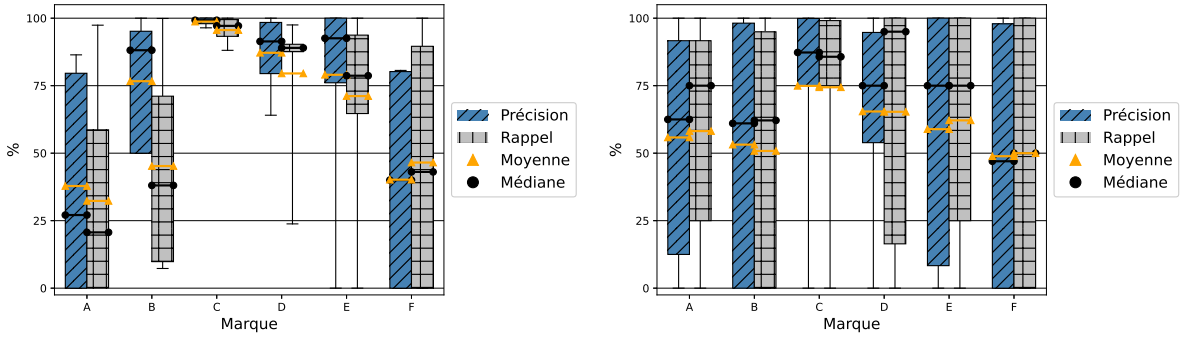
Nous remarquons ainsi par les performances de notre classificateur, que l'ensemble des services actifs S_d, soit huit services, ne semble pas suffire pour prédire correctement chaque marque. Une première piste d'amélioration serait d'intégrer d'autres services, comme `dec-notes` (port 3333) ou `ripd` (port 2602), à notre liste de services actifs. De plus, comme discuté dans la section 6.3.3, d'autres méthodes existent pour inférer la marque d'un produit, comme les bannières SSH ou TELNET.

Pour les expérimentations suivantes, nous considérons la marque comme étant connue, et pouvons ainsi utiliser les classificateurs spécifiques à chaque marque. Les expérimentations détaillées ci-après répondent aux objectifs **O2** et **O3**, c'est-à-dire inférer 1) le nom, puis 2) la version d'un binaire déployant un service tel que HTTP (6.4.3.2), SSH (6.4.3.3) ou DNS (6.4.3.4).

6.4.3.2 Identification du serveur HTTP

À partir de la marque $marque_d$ et des services actifs S_d d'un objet connecté d, notre expérimentation (Eval.2) prédit le nom du binaire déployant le service HTTP, et obtient des valeurs des micro-moyennes précision et rappel élevés (86.89% et 87.28% respectivement). Les macro-moyennes précision et rappel, elles, ont des valeurs plus faibles (75.31% et 68.19% respectivement). En effet, les binaires sous-représentés, comme `shttpd` ou `hiawatha`, sont souvent faussement classifiés alors que les binaires majoritairement déployés tels que `lighttpd` ou `mini_httpd`, sont eux correctement identifiés. D'après la figure 6.5(a), nous observons que les noms des binaires HTTP associés aux marques **D**, **E** et particulièrement **C** sont correctement identifiés. **B** utilise majoritairement `lighttpd`, c'est pourquoi son classificateur associé identifie correctement ce binaire avec une précision et un rappel supérieurs à 95% contrairement aux binaires minoritaires, comme `nginx` ou `uhttpd`, dont le rappel est de 6% et 69% respectivement. La macro-moyenne

précision des marques **A** et **F** est d'environ 40%, nous expliquons cela par l'utilisation de multiples binaires HTTP dans des objets IdO déployant des services identiques, d'où la difficulté de notre classificateur à identifier ces binaires correctement.



(a) Nom du binaire HTTP

(b) Version du binaire HTTP

FIGURE 6.5 – Inférence du nom et de la version des binaires HTTP

Enfin, le nombre moyen de tentatives pour trouver le nom du binaire HTTP est de 1.16, réduisant ainsi le nombre d'attaques nécessaires pour compromettre ou identifier ce binaire à deux interactions au maximum.

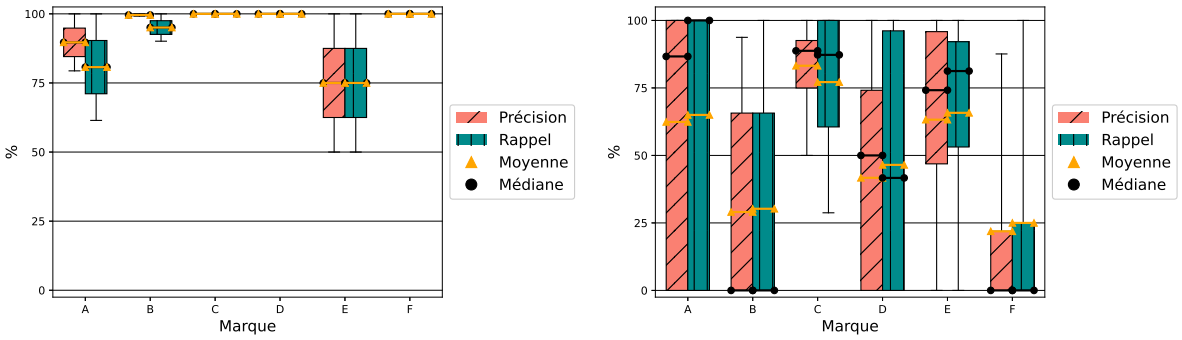
Dans la seconde inférence (Eval. 3), nous considérons le nom du binaire HTTP comme étant connu afin d'inférer la version de ce binaire, et observons des valeurs des micro-moyennes précision et rappel également élevées (94.40% et 88.81% respectivement). Cependant, les macro-moyennes précision et rappel atteignent environ 63% indiquant que parmi les nombreuses versions des binaires HTTP disponibles, les versions les plus représentées sont souvent correctement prédites alors que les versions minoritaires sont omises. D'ailleurs, les valeurs des précisions et rappels obtenus par les classificateurs de chaque marque et illustrées dans la figure 6.5(b), varient énormément. Par exemple, le classificateur spécifique à la marque **F** a des précisions et rappels oscillant entre 0 et 1.

Ces larges variations peuvent s'expliquer par les mises à jour effectuées par chacun des fabricants entre 2009 et 2019. C'est pourquoi, les macro-moyennes précision et rappel sont faibles. Malgré ces mises à jour, les valeurs élevées des micro-moyennes indiquent que notre approche permet d'identifier une large proportion des versions des binaires HTTP. D'ailleurs, en moyenne, le classificateur de chaque marque est capable de retourner la version correcte dans 99.44% des cas, et en moins de deux tentatives (1.08).

6.4.3.3 Identification du serveur SSH

Comme discuté dans la section 5.5.4, le binaire `dropbear` (90.08%) est largement utilisé par rapport à `OpenSSH` (9.92%) pour déployer un serveur SSH. C'est pourquoi les prédictions du nom de ce binaire à partir des services actifs S_d et de la $marque_d$ (Eval. 4) sont généralement correctes, comme le confirment les précisions et rappels visibles dans la figure 6.6(a). De plus, la valeur de la micro-moyenne précision de 98.68% implique que la connaissance de $marque_d$ et S_d est bien corrélée au binaire déployant SSH.

Les performances des classificateurs associés aux marques **A** et **E** sont plus faibles. Après une inspection manuelle des résultats, nous expliquons ces performances par 1) la distribution équivalente des deux binaires dans les objets de la marque **A**, et 2) la présence de deux objets ayant `OpenSSH` pour la marque **E**.



(a) Nom du binaire SSH

(b) Version du binaire SSH

FIGURE 6.6 – Inférence du nom et de la version des binaires SSH

Similairement à l'expérience 6.4.3.2, nous supposons le nom du binaire SSH comme étant connu avant d'inférer la version de ce dernier (Eval .5). Comme illustré par la figure 6.6(b), les précisions et rappels retournés par le classificateur de toutes les marques, hormis **C**, sont inférieures à 65%. D'un autre côté, les micro-moyennes précision et rappel listées dans le tableau 6.2 sont de 80.29% et 74.54% respectivement. Cela illustre donc que la majorité des objets IdO utilisent massivement certaines versions, comme `dropbear` version `2016.74` ou `OpenSSH` version `6.6.1p1`, qui sont, elles, souvent correctement prédites, ce qui n'est pas le cas des versions plus *rares* telles que `OpenSSH` version `6.7p1` ou `dropbear` version `0.45`. En ce qui concerne les performances de la marque **C**, nous observons que seules huit versions déployées ont des précisions toutes supérieures à 50% ainsi, il semblerait que la version déployée dépend du type de l'objet.

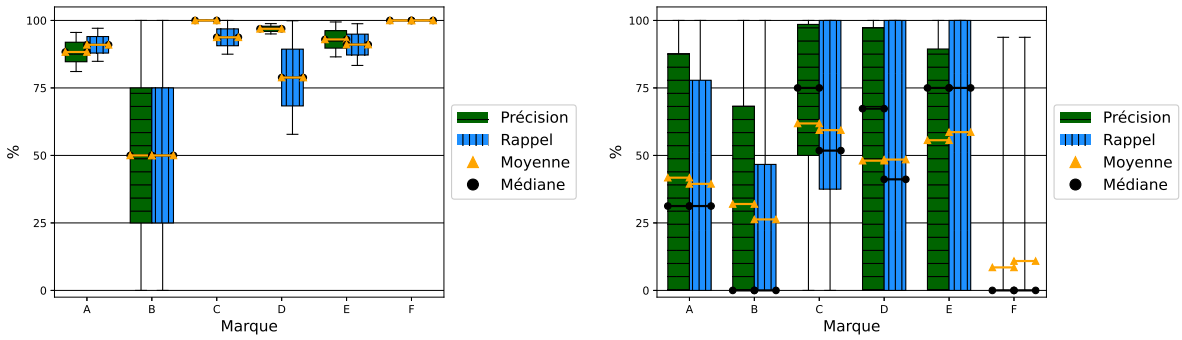
À l'aide des classificateurs par marque, nous observons que ces derniers sont capables de retourner la vraie version du binaire SSH dans plus de 98.37% des cas. Le nombre de tentatives moyen varie de 1.01 pour **A** à 1.37 pour **B**, avec un écart-type d'au plus 0.66 ce qui montre que chaque classificateur semble retourner directement une version souvent majoritaire. Par conséquent, en deux tentatives d'exploit si des vulnérabilités existent, un attaquant serait capable de compromettre le binaire SSH.

6.4.3.4 Identification du serveur DNS

Similairement à 6.4.3.3, peu de binaires sont utilisés pour déployer un serveur DNS dont `dnsmasq` (96.16%), `BusyBox dnsd` (3.77%), `nsd` (0.05%) et `maradns` (0.02%). Ainsi, un attaquant peut naïvement *prédire* le nom du binaire déployant DNS avec une précision de 96.16%. Néanmoins, nous mesurons tout de même les performances des classificateurs à prédire le nom du binaire DNS à partir des services actifs S_d et de la marque $marque_d$ d'un objet d (Eval 6.). Comme visible dans la figure 6.7(a), les précisions et rappels varient pour la marque **B** entre 0.00 et 1.00. Nous expliquons cela par la présence de deux binaires `BusyBox dnsd` et `dnsmasq`, le premier est déployé dans seulement 5 produits et n'est pas prédit correctement par le classificateur, alors que le second binaire est présent dans 2253 objets et est prédit comme tel.

De plus, les micro-moyennes précision et rappel présentées dans le tableau 6.2 sont supérieures à environ 99% , ce qui nous permet de conclure qu'utiliser $marque_d$ et S_d pour prédire le nom du binaire DNS permet d'améliorer les performances par rapport à la prédiction naïve de `dnsmasq`.

Similairement aux problèmes de sécurité du service HTTP discutés dans la section 5.5.3 du chapitre 5, les attaques ciblant les binaires DNS, notamment `dnsmasq`, sont souvent liées aux paramètres de configuration utilisés. Par exemple les vulnérabilités CVE-2017-14495 CVE-2020-



(a) Nom du binaire DNS

(b) Version du binaire DNS

FIGURE 6.7 – Inférence du nom et de la version des binaires DNS

Marque	A	B	C	D	E	F
***trials* moy.**	1.20	1.44	1.09	1.44	1.18	1.94
***trials* écart-type**	0.58	0.75	0.29	0.87	0.55	1.46

Tableau 6.3 – Nombre de tentatives, ou *trials*, pour identifier la version du binaire DNS

25687 peuvent causer des dépassements de tampons, ou *buffer overflow* en anglais, si certaines options de configuration, comme DNSSEC, sont activées et exploitées par des attaquants. Cependant, pour confirmer ces vulnérabilités, une analyse complémentaire des fichiers de configuration serait nécessaire. Néanmoins, d'autres vulnérabilités non dépendantes aux options de configuration existent, comme les CVE-2017-14492 et CVE-2020-25685 permettant à un attaquant de causer un dépassement de tampons ou un empoisonnement du cache DNS (*DNS cache poisoning* en anglais). Ainsi, l'identification de la version du binaire DNS par un attaquant reste critique.

La prédiction de la version du binaire DNS se fait à l'aide de la marque $marque_d$, des services actifs S_d et du nom du binaire DNS associé à un objet d (Eval. 7). Les précisions et rappels retournés par le classificateur de chaque marque sont visibles dans la figure 6.7(b). Ces dernières fluctuent sur de larges intervalles, ce que nous expliquons par le nombre de versions par marque variant de 9 (**C**) à 20 (**D**). Les marques **B** et **F** ont les pires résultats avec des médianes sur les précisions et rappels égaux à 0, car la plupart des 14 (**B**) et 17 (**F**) versions disponibles ont une précision et rappel égaux à 0. En effet, contrairement aux autres services, nous observons que certaines versions sont déployées uniquement dans 1 objet. Malgré les nombreuses versions disponibles, les micro-moyennes précision et rappel, présentées dans le tableau 6.2, sont égales à 73.14% et 64.23% respectivement dû à certaines versions majoritairement utilisées comme `dnsmasq 2.66` et `dnsmasq 2.47`. Malgré une macro-moyenne précision inférieure à 62%, les marques **A**, **C**, **D** et **E** ont toutes des micro-moyennes précision et rappel supérieures à 75%, confirmant l'usage d'une ou plusieurs versions majoritaires dans certains types d'objets. En moyenne, nous remarquons également que le classificateur propre à chaque marque est capable de trouver la vraie version du binaire DNS dans au moins 95% des cas.

C'est pourquoi, nous listons dans le tableau 6.3, les nombre de tentatives moyens ainsi que l'écart-type de ces derniers obtenus par le classificateur de chaque marque. Les écart-types des marques **B**, **D** et **F** sont élevés confirmant donc la difficulté pour chaque classificateur à correctement identifier une version peu représentée.

6.4.3.5 Inférence des identifiants

Afin de compromettre un objet connecté déployant un serveur SSH ou TELNET, un attaquant peut essayer de se connecter à ce dernier à l'aide des noms d'utilisateurs (identifiants) et mots de passe par défaut de l'objet. En effet, des milliers d'objets IdO connectés à l'Internet utilisent toujours des identifiants et mots de passe par défaut [49, 91, 13]. De plus, naïvement supposer que l'identifiant `root` est utilisable dans tous les objets IdO connectés n'est pas valide. En effet, seuls 2239 (47.67%) des objets analysés ont cet identifiant.

Dans cette section, nous inférons les identifiants présents u_d dans un objet connecté d. Premièrement, nous déduisons ces derniers à partir des services actifs S_d de l'objet (Eval. 8), puis dans une seconde expérimentation (Eval. 9), nous ajoutons en connaissance la marque de l'objet $marque_d$. Dans les deux cas, le classificateur retourne un identifiant valide dans plus de 99.86% des cas, avec un *trial* égal à 1.03 lorsque $marque_d$ est connue, 1.09 sinon.

Malgré le grand nombre d'utilisateurs valides possible (30), nous remarquons que les micro-moyennes précision et rappel sont supérieures à 88% et 79% dans les deux expérimentations (Eval. 8 et 9). Cela montre qu'un faible nombre d'identifiants sont souvent présents dans les objets IdO connectés, et sont correctement prédits par nos classificateurs. Par exemple, l'identifiant `admin`, présent dans 506 objets, est prédit avec une précision supérieure à 84% dans l'Eval. 8 et 90% dans l'Eval. 9. Alors que les identifiants, propres à la marque **A**, comme `ZX4q9Q9JUpwTZuo7` (1 objet) ou `gkJ9232xXyruTRmY` (3 objets) ne sont pas prédits dans les deux expérimentations.

Contrairement à une méthode *brute-force*, notre approche est capable d'identifier rapidement, c'est-à-dire en moins de deux tentatives, un des identifiants actifs d'un objet connecté. Cependant, il est bon de noter que nous ne tenons pas compte dans ces deux expérimentations du contenu des services actifs de l'objet. En effet, l'intêret de prédire les identifiants d'un objet n'exposant pas de services, ou des services ne permettant pas l'accès à distance, est limité.

6.4.3.6 Inférence des mots de passe

Dans la section précédente, nous avons montré que notre méthode était capable d'inférer un nom d'utilisateur (identifiant) valide en, au plus, deux tentatives. Or, un processus de connexion requiert un couple <utilisateur, mot de passe>. C'est pourquoi dans cette section, notre objectif est de prédire les mots de passe pw_{ud} associés aux utilisateurs présents dans un objet d, notés u_d à partir des services actifs S_d et de la marque $marque_d$ de ce dernier (Eval. 10). Du point de vue d'un attaquant, il n'est pas utile de prédire l'intégralité des mots de passe de l'objet, un seul suffit, d'où l'introduction de notre métrique du nombre de tentatives.

D'ailleurs, comme évoqué dans la section 6.3.2, ces pw_{ud} peuvent être chiffrés. Ainsi, même en prédisant l'intégralité des pw_{ud} un attaquant devra déchiffrer ces derniers, ce qui, suivant la complexité du mot de passe et la fonction de hachage utilisée, peut nécessiter un temps de calcul non négligeable [103]. Cette opération s'effectue discrètement par l'attaquant sans interagir avec l'objet, c'est-à-dire hors-ligne.

Malgré les 83 différents mots de passe à prédire, les résultats obtenus par le classificateur principal présentés dans le tableau 6.2 confirment que, comme pour les identifiants, un sous-ensemble de mots de passe est majoritairement utilisé.

Dans le tableau 6.4, nous présentons les résultats associés aux classificateurs de chaque marque, notamment le nombre de tentatives, pour inférer le mot de passe 1) de l'utilisateur `root`, et 2) de n'importe quel utilisateur présent dans les u_d, noté $all(u_d)$. Pour la première expérience, nous conservons les instances où l'utilisateur `root` est bloqué afin de souligner que ce dernier n'est pas systématiquement présent dans les objets IdO ni dans toutes les marques.

Marque	Utilisateur(s)	Nb. Mot de passe	Mot de passe trouvé	Moy. *trial*	Écart-type *trial*
A	`root`	13	73.12	1.19	0.52
B	`root`	22	55.06	1.05	0.21
C	`root`	13	88.12	1.17	0.39
D	`root`	10	86.85	1.08	0.31
E	`root`	12	75.32	1.23	0.66
F	`root`	N/A	N/A	N/A.	N/A.
A	$all(u_d)$	21	97.67	1.28	0.81
B	$all(u_d)$	25	99.19	1.03	0.19
C	$all(u_d)$	15	98.89	1.24	0.60
D	$all(u_d)$	16	99.73	1.10	0.48
E	$all(u_d)$	24	99.27	1.27	0.79
F	$all(u_d)$	3	97.05	1.00	0.00

Tableau 6.4 – Nombre de tentatives, ou *trial* en anglais, pour inférer un des mots de passe associés à n'importe quel utilisateur présent dans u_d, noté $all(u_d)$, et `root`

En effet, le classificateur de la marque **B** est capable de prédire un mot de passe dans seulement 55.06% des cas, alors que le classificateur de la marque **F** n'infère aucun mot de passe pour l'utilisateur `root` d'où les valeurs N/A. De même, en comparant les performances entre les inférences de l'utilisateur `root` et $all(u_d)$, nous remarquons que les marques **A** et **E** semblent, elles aussi, utiliser d'autres utilisateurs. Cela confirme donc l'observation de la section 6.4.3.3 que d'autres utilisateurs, comme `admin` ou `Admin`, sont privilégiés par certaines marques pour les tâches d'administration.

Vis-à-vis des mots de passe associés aux $all(u_d)$, nous observons par les résultats listés dans le tableau 6.4 que, peu importe la marque, notre approche est capable de trouver au moins un des mots de passe dans plus de 97% des cas et ce, avec au plus deux tentatives. D'ailleurs, le classificateur de la marque **F** retourne directement le bon mot de passe alors que, pour la marque **E**, nous observons par la valeur de l'écart-type (0.79) que le classificateur associé a plus de difficulté à prédire un mot de passe correct parmi les 24 mots de passe possibles.

6.4.4 Discussion

Compte tenu des résultats exposés dans le tableau 6.2, nous avons vu que ces derniers dépendent souvent de la marque des objets. De plus, nos classificateurs ont souvent des difficultés à prédire correctement toutes les classes. Comme notre méthode de fingerprinting active est censée être utilisée par un attaquant, ce dernier va concentrer ses attaques sur les objets dont les propriétés sont facilement inférables, c'est-à-dire les inférences avec les meilleures performances.

En effet, l'une des premières phases d'une attaque à large échelle est la phase de reconnaissance qui, dans notre cas, consiste à obtenir la liste des services actifs à l'aide d'un scan de ports TCP/UDP. À partir de ces milliers, voire millions, de listes, notre approche ne requiert pas de retourner des résultats exacts sur chacune de ces listes. C'est pourquoi, un attaquant va plutôt chercher à obtenir les propriétés les plus probables associées à chacun des objets ciblés. C'est pourquoi, par exemple, des botnets comme Mirai [91, 13] effectuent des attaques par *brute-force* sur les couples <utilisateur, mot de passe> les plus répandus dans l'IdO.

Afin de garantir une certaine furtivité, chaque attaque doit être effectuée en minimisant le nombre d'interactions avec l'objet ciblé. C'est pourquoi, nous avons introduit dans la section 6.4.1, la métrique du nombre de tentatives afin de connaître le nombre d'interactions nécessaire à un attaquant qui, d'après nos résultats, est globalement inférieur ou égal à deux. C'est

d'ailleurs, l'une des différences entre notre approche et les méthodes de classification standards où la prédiction directement retournée doit être correcte.

Enfin, notre méthode utilise les services actifs sur un objet connecté comme principale information pour répondre ensuite aux objectifs présentés dans la section 6.2.2. Il est ainsi relativement facile pour un fabricant de contrer cette dernière en déployant, entre autres, des services fictifs sur les ports analysés ou de déployer certains services sur des numéros de port non standards, sur ses objets.

6.5 Synthèse

Dans ce chapitre, nous avons introduit une méthode de fingerprinting active permettant de déduire des informations, comme le nom des binaires ou des utilisateurs, associées à des objets IdO connectés à l'Internet. Notre méthode, théorique, suppose un scénario où un attaquant effectue premièrement un scan de ports TCP/UDP, puis, à partir des résultats, ce dernier va chercher à inférer des informations sur l'objet, par exemple sa marque ou le nom et la version du binaire déployant le service HTTP ou SSH. Contrairement aux méthodes existantes, comme l'analyse de bannières, notre méthode utilise initialement le résultat d'un scan de ports pour déduire des connaissances sur un objet connecté, ce qui ne requiert pas d'interagir d'avantage avec ses services déployés.

Pour ce faire, nous avons utilisé des algorithmes de classification *Random Forest* entraînés à partir des données de firmwares d'objets IdO publiés entre 2009 et 2019. Parmi les données utilisées, nous avons, entre autres, les noms et versions des binaires déployant des services comme HTTP, SSH ou DNS, ainsi que les identifiants et mots de passe présents dans les fichiers de configuration.

Dans la première évaluation, nous nous sommes intéressés au scénario où un attaquant identifie la marque d'un objet connecté à partir des résultats d'un scan de ports TCP/UDP associé à huit services, ou services actifs. En moyenne, notre approche permet d'identifier la marque d'un objet avec une précision et un rappel de 76.85% et 66.57% respectivement, et trois des six marques testées sont identifiées avec une précision supérieure à 89.6%. Néanmoins, la valeur moyenne du rappel illustre la présence de faux négatifs. Ainsi, intégrer d'autres services aux résultats du scan de ports pourrait améliorer les performances de notre classificateur.

Une fois la marque identifiée, nous avons inféré les noms, puis versions, des binaires associés aux services HTTP, SSH et DNS déployés par un objet. Pour ce faire, nous avons utilisé un classificateur par marque. Dans un premier temps, nous avons inféré le nom de ces binaires à partir des services actifs de l'objet. Bien que différents binaires soient utilisés pour déployer le service HTTP, notre approche est capable de retourner dans plus de 99% des cas le nom du binaire, et ce en deux tentatives au maximum. Les services SSH et DNS sont généralement déployés à l'aide d'un binaire souvent majoritairement présent dans les objets, comme `dropbear` et `dnsmasq` respectivement. Néanmoins, notre approche s'est montrée plus efficace qu'une méthode naïve consistant à retourner le nom du binaire majoritaire. Le nom du binaire ne suffit généralement pas à exploiter une vulnérabilité particulière. C'est pourquoi, nous avons ensuite cherché à prédire la version des binaires utilisés. Indépendamment des services, nos prédictions ont des macro-moyennes précisions et rappels faibles, ce qui illustre que de nombreuses versions différentes sont disponibles, et que nos classificateurs peinent à identifier la version correcte directement. Néanmoins, et en moyenne, notre approche permet à un attaquant d'identifier la vraie version d'un binaire HTTP, SSH ou DNS en deux tentatives. Par conséquent, l'utilisation de notre approche permet d'identifier rapidement et efficacement les propriétés associées aux binaires

déployant ces trois services, et ce en minimisant les interactions avec l'objet ciblé.

Les objets IdO proposant des interfaces de connexions, via TELNET ou SSH, sont souvent sujets à des attaques par *brute-force* [91, 13, 107]. Ces dernières génèrent énormément de trafic réseau et sont donc peu furtives. C'est pourquoi, nous avons mesuré les performances de notre approche sur l'inférence des noms d'utilisateurs et les mots de passe d'un objet IdO. Cette dernière s'est montrée capable de prédire le nom d'au moins un utilisateur dans plus de 99% des cas, et quasi instantanément (en moyenne 1.03 tentatives). Selon la marque, entre 3 et 25 possibilités existent. Malgré cela, notre approche arrive, en moyenne, à identifier au moins un mot de passe valide dans plus de 97% des prédictions, et en 2 tentatives maximum.

L'utilisation de notre méthode de fingerprinting active s'est montrée efficace pour la déduction des propriétés d'un objet connecté. Cette dernière permet de réduire les interactions avec les objets ciblés, et ainsi augmenter la furtivité d'une attaque. L'identification des noms et des versions des binaires permet également à un attaquant de sélectionner, ou implémenter, des vulnérabilités spécifiques. Cependant, nous avons supposé que la marque de l'objet ciblé était correctement prédite alors que les performances de notre approche à inférer cette dernière sont mitigées. C'est pourquoi nous proposons d'inclure d'autres services lors du supposé scan de ports.

Quatrième partie

Étude des cyberattaques ciblant les objets IdO

Chapitre 7

Implémentation d'un pot de miel à forte interaction pour objets IdO

Sommaire

7.1 Introduction

Les pots de miel spécifiques aux objets IdO ont été présentés dans le chapitre 3 de l'état de l'art. Ces derniers étant des systèmes conçus spécifiquement pour recevoir des attaques, et ainsi étudier 1) les vulnérabilités exploitées ou 2) les actions effectuées par les attaquants.

Pour observer des attaques complexes, ou venant d'attaquants humains, un pot de miel doit simuler de manière réaliste le fonctionnement d'un objet IdO tout en laissant l'attaquant effectuer

les actions de son choix, l'objectif étant de faire croire à ce dernier qu'il se trouve sur un objet IdO physique, ou réel, et parfaitement fonctionnel. C'est pourquoi, un pot de miel à forte interaction correspond le mieux à ce besoin mais est compliqué à implémenter dans le contexte de l'IdO.

En effet, l'hétérogénéité dans les fonctionnalités et services présents dans les objets IdO est difficile à simuler correctement. D'un côté, des objets IdO réels peuvent être connectés à l'Internet et instrumentés pour observer des attaques, mais cette approche présente un problème de passage à l'échelle et nécessite d'automatiser la réinitialisation de ces objets physiques en cas de compromission. D'un autre côté, l'utilisation d'objets IdO émulés permet de rapidement déployer des pots de miel, facilement maintenables, mais impose des ajustements pour rendre ces objets IdO émulés 1) similaires à leurs équivalents physiques, et 2) capables d'observer discrètement les actions effectuées par les attaquants depuis ces derniers.

Notre contribution [190] présente un pot de miel à forte interaction simulant un objet IdO. Pour ce faire, nous avons utilisé et amélioré la méthode introduite par FIRMADYNE [36] permettant d'émuler un objet IdO à partir de son firmware. Cependant, cette méthode requiert de modifier le contenu du firmware, en ajoutant notamment des fichiers, donc un attaquant avisé peut détecter un objet IdO émulé, et déployé en pot de miel, en cherchant les fichiers ajoutés par FIRMADYNE, ou sa version améliorée FirmAE [90]. Les améliorations proposées prennent en compte le besoin de furtivité inhérent à un pot de miel, et sont implémentées de sorte à être facilement évolutives. Contrairement au pot de miel existant Honware [169], notre contribution intègre des mécanismes internes afin d'augmenter la furtivité du pot de miel, comme en falsifiant le contenu de fichiers de configuration ou en cachant les fichiers nécessaires au bon déroulement de l'émulation. De plus, d'après [169], il est difficile pour un attaquant de distinguer un objet émulé de son équivalent physique à partir de son trafic réseau.

Ce chapitre est découpé en quatre sections. Dans la section 7.2, nous justifions notre choix d'un déploiement de pots de miel à partir d'objets IdO émulés. Dans la section 7.3, nous présentons les améliorations apportées au processus d'émulation, puis détaillons dans la section 7.4 les modifications implémentées afin de rendre l'objet émulé furtif mais aussi capable d'intercepter les activités des attaquants. Enfin, nous présentons dans la section 7.5, les performances de notre contribution à émuler des objets IdO, ainsi que les attaques reçues sur nos pots de miel.

7.2 Déploiement de pots de miel à partir de firmware d'objets IdO

Dans cette section, nous détaillons les deux objectifs de notre approche, le premier étant l'amélioration du processus d'émulation d'un objet, et le second l'implémentation, dans un objet émulé, de fonctionnalités inhérentes à un pot de miel. Enfin, nous présentons l'infrastructure logicielle employée ainsi que le noyau et module Linux utilisé pour répondre à ces objectifs.

7.2.1 Objectifs

Notre pot de miel repose sur la méthode d'émulation d'objets IdO [36] présentée dans l'annexe C. Cette dernière modifie le firmware de l'objet IdO à émuler. Ces modifications doivent être cachées aux attaquants sans pour autant réduire les performances d'émulation. En complément de la partie émulation, le pot de miel doit intercepter les actions effectuées par les attaquants le plus furtivement possible, ce qui n'est pas implémenté dans [36]. Ainsi, nous avons étendu cette méthode en une nouvelle infrastructure logicielle, ou *framework* en anglais, autour des deux objectifs suivants :

— L'amélioration des performances d'émulation. Les objets IdO étant hétérogènes en termes de fonctionnalités et de composants matériels, notre méthode doit être capable de corriger des erreurs liées au processus d'émulation, comme des problèmes de connectivité réseau ou des plantages système. Des corrections ont d'ailleurs été proposées dans [169, 90], et notre méthode adapte certaines d'entre elles. Toutes les mesures implémentées par notre *framework* sont détaillées dans la section 7.3.

— La définition et l'implémentation de fonctionnalité d'un pot de miel, comme l'interception de manière furtive des activités effectuées par l'attaquant ou le camouflage de la partie émulation. L'ensemble des fonctionnalités implémentées dans notre pot de miel sont détaillées dans la section 7.4.

Ces deux objectifs sont fortement couplés. En effet, la partie émulation modifie le firmware de l'objet IdO à émuler, ce qui réduit la furtivité du pot de miel. Pour compenser cela, des mesures doivent être implémentées dans la partie pot de miel.

7.2.2 Infrastructure logicielle

Nous utilisons le *framework* illustré dans la figure 7.1 afin d'isoler le pot de miel de la machine hôte, c'est-à-dire la machine depuis laquelle le pot de miel est déployé.

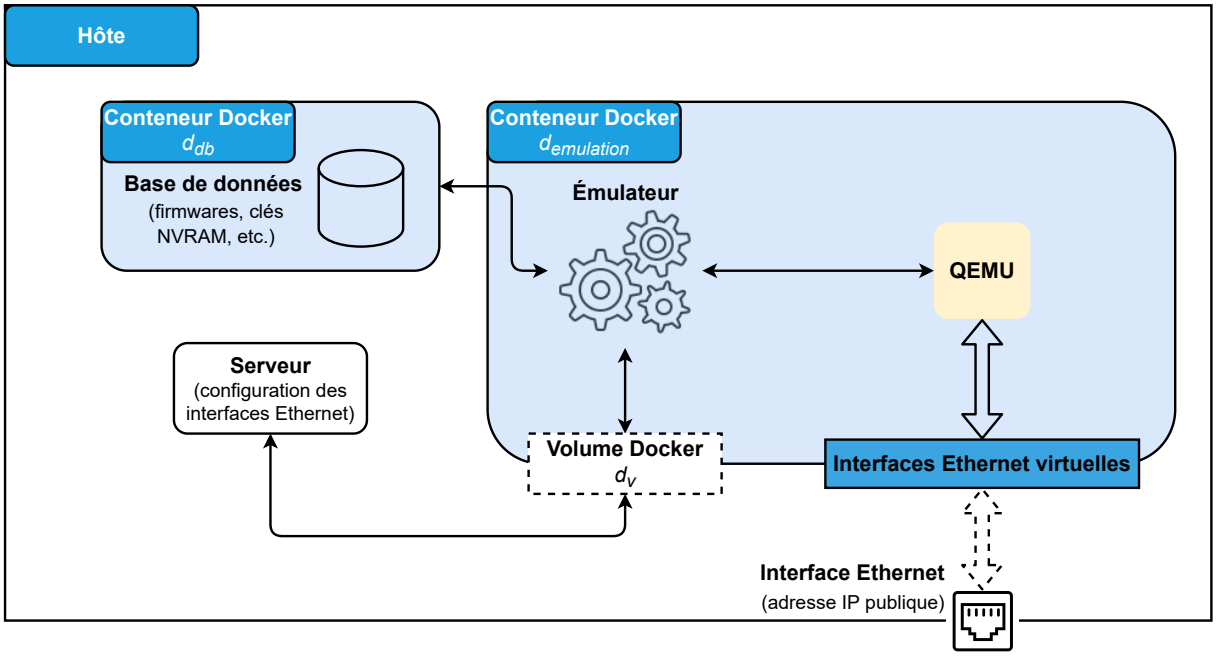


FIGURE 7.1 – Infrastructure logicielle utilisée pour émuler un objet IdO, et le déployer en pot de miel

Depuis la machine hôte, deux conteneurs Docker sont utilisés : d_{db} et $d_{emulation}$. Le premier, d_{db}, déploie une base de données stockant les informations sur les firmwares à émuler (firmwares, clés NVRAM, etc.), ainsi que les données à falsifier dans le pot de miel (contenu des fichiers `/proc/cpuinfo` ou `/proc/cmdline`). Le second conteneur, $d_{emulation}$, gère le processus d'émulation et le déploiement du pot de miel à l'aide des données stockées dans d_{db}. Durant le processus d'émulation, le firmware de l'objet à émuler est exécuté itérativement par QEMU afin de permettre à notre émulateur de corriger les erreurs d'émulation à l'aide des mesures détaillées dans

la section 7.3. Une fois les erreurs d'émulation corrigées, le déploiement en pot de miel requiert de configurer les interfaces réseau entre la machine hôte et l'objet émulé via QEMU.

Pour ce faire, notre émulateur interagit à l'aide d'un socket UNIX localisé dans le volume Docker d_v avec le serveur exécuté depuis la machine hôte. Ce dernier va créer les interfaces ethernet virtuelles entre $d_{emulation}$ et la machine hôte, puis configurer ces interfaces et les règles de pare-feu, afin de rendre accessible le pot de miel depuis l'Internet.

De plus, d_v permet également à la machine hôte et à $d_{emulation}$ d'avoir un répertoire commun dans lequel les informations sur l'émulation et le pot de miel sont stockées. Par exemple, les historiques d'émulation et les captures du trafic réseau du pot de miel.

De la même manière que [36, 169, 90], QEMU est utilisé pour émuler un objet IdO à partir de son firmware car ce dernier permet 1) d'émuler des architectures processus différentes de la machine hôte (par exemple MIPS, ARM), et 2) une configuration aisée des interfaces réseau (NIC) à allouer sur l'objet émulé. Cette dernière fonctionnalité permet notamment d'assigner un nombre quelconque d'interfaces réseau et d'enregistrer automatiquement leur trafic réseau vers des fichiers PCAP.

Linux étant le système d'exploitation le plus répandu dans l'IdO [36, 173], nous émulons uniquement des objets IdO compatibles avec ce dernier. De plus, nous utilisons un noyau Linux en version 2.6.36 compilé pour les architectures processeur MIPS 32 bits *little-endian* et *big-endian*, car il s'agit, respectivement, de la version et des architectures les plus déployées dans l'IdO [36, 90, 173].

Similairement à [36, 90, 169], nous implémentons également un module Linux (*Linux Kernel Module (LKM)* en anglais) intégré au noyau durant sa compilation. Le rôle de ce module est 1) d'améliorer le processus d'émulation, et 2) d'assurer la furtivité du pot de miel tout en interceptant les activités des attaquants.

7.2.3 Module et noyau Linux

Notre module Linux est capable d'intercepter jusqu'à 20 appels système à l'aide des outils présentés dans la section C.1.1 de l'annexe C. Les appels système interceptés sont décrits et listés dans le tableau C.1. Parmi ces appels système, 6 sont nécessaires au pot de miel pour remplir ses fonctionnalités, 11 appels système génèrent des messages de débogage, et sont utilisés par la partie émulation pour détecter, entre autres, les fichiers manquants, ou la configuration des interfaces réseau attendu par l'objet émulé. 3 appels système sont nécessaires pour le bon déroulement de l'émulation, et intègrent des fonctionnalités inhérantes au pot de miel. Par exemple, l'appel système `do_vfs_ioctl` permet de détecter les interfaces réseau manquantes, et créée également de fausses interfaces réseau sans-fil, comme détaillé dans la section C.2.8.

Par conséquent, notre approche utilise deux noyaux Linux. Le premier est compilé pour intercepter l'intégralité des 20 appels système, et est utilisé durant le processus d'émulation discuté dans la section 7.3. Le second est compilé avec les 11 appels système propres à la partie pot de miel, et est utilisé pour le déploiement du pot de miel sur l'Internet dont les fonctionnalités sont détaillées dans la section 7.4.

Enfin, notre processus d'émulation ainsi que la partie pot de miel doivent communiquer avec notre module. Pour ce faire, notre module implémente un fichier f dans le répertoire `/proc` de façon à pouvoir recevoir des instructions, et les interpréter. Par exemple, créer des fichiers spéciaux, ou cacher des fichiers aux attaquants. Une fois toutes les instructions envoyées au module, le fichier f peut être supprimé à l'aide d'une instruction spécifique : `echo "0|" > /proc/repair`. Cette dernière est impérativement effectuée lorsque l'objet émulé est déployé en pot de miel afin de maintenir un bon niveau de furtivité.

7.3 Amélioration du processus d'émulation

Dans cette section, nous présentons les améliorations apportées au processus d'émulation initialement présenté par [36]. Dans la section 7.3.1, nous discutons de notre processus de création d'une image QEMU, puis expliquons dans la section 7.3.2 comment notre approche sélectionne le script d'initialisation nécessaire au démarrage de l'objet émulé. Afin de corriger les erreurs d'émulation, nous présentons dans la section 7.3.3, une approche itérative permettant 1) de détecter ces erreurs, et 2) distribuer ces dernières à des modules effectuant des correctifs sur l'image QEMU. Chacun de ces modules est implémenté pour corriger un type d'erreur particulier, comme la gestion des fichiers manquants présentée dans la section 7.3.6, ou les problèmes de connectivité réseau discuté dans la section 7.3.7.

7.3.1 Construction d'une image QEMU à partir d'un firmware

Comme illustré dans la figure 7.1, nous utilisons l'émulateur QEMU pour émuler un système complet. Pour ce faire, ce dernier a besoin : d'un noyau Linux, et d'une image QEMU contenant un système de fichiers racine extrait du firmware de l'objet IdO à émuler. La construction d'une image QEMU est détaillée dans la section C.1.2.

Dans cette image QEMU Q, un dossier d est créé et contient les fichiers nécessaires au bon déroulement de l'émulation. Similairement à [169, 90], notre approche utilise un deuxième script shell afin de configurer les interfaces réseau, modifier les règles de pare-feu ou démarrer des binaires depuis l'objet émulé. Dans notre approche, ce script est copié dans d pour faciliter son camouflage. En effet, le nom du dossier d est 1) généré aléatoirement, et 2) caché via la méthode décrite dans la section C.2.4, afin d'empêcher un attaquant de détecter la présence de ce dossier, et donc le pot de miel.

Afin de construire une image QEMU stable, c'est-à-dire avec le moins d'erreurs d'émulation possibles, nous émulons à plusieurs reprises Q pour corriger itérativement les erreurs d'émulation, comme décrit dans la section 7.3.3.

7.3.2 Sélection du script d'initialisation

Préalablement à l'émulation d'une image QEMU, il est impératif de sélectionner le script d'initialisation à utiliser pour démarrer l'émulation de cette image. En effet, pour émuler une image QEMU Q à l'aide de l'émulateur du même nom et d'un noyau Linux, il est nécessaire de spécifier à l'émulateur le binaire présent dans Q à exécuter en premier par le noyau Linux, c'est-à-dire le script d'initialisation.

Comme détaillé dans la section C.1.3, un script shell, noté *preinit*, est également exécuté avant le script d'initialisation, et doit remplir les objectifs suivants :

1. vérifier l'état du système de fichiers,
2. communiquer à notre module les instructions à effectuer,
3. préconfigurer les interfaces réseau,
4. démarrer le script d'initialisation.

De cette manière, notre approche permet de spécifier au noyau, n'importe quel script d'initialisation. Le choix de ce dernier est effectué à l'aide d'une liste ordonnée, notée L, de scripts d'initialisation présents dans les objets IdO (par exemple `/init`, `/init.d/rcS` ou `/sbin/acos_init`), de telle sorte que le premier script présent dans L et trouvé dans Q sera le script d'initilisation utilisé pour démarrer l'objet émulé. L'ordre des scripts dans la liste L a été développé empiriquement.

7.3.3 Amélioration itérative de l'image QEMU

Afin d'éviter les erreurs associées aux fichiers spéciaux dans `/dev` et `/proc`, les travaux précédents [36, 90] créent ces fichiers automatiquement dans l'image QEMU Q peu importe la marque ou le type de l'objet IdO émulé. Par exemple, 44 fichiers spéciaux standards[65] sont créés dans `/dev`. Cependant, utiliser cette stratégie trahirait le pot de miel. C'est pourquoi, notre approche privilégie la création des fichiers nécessaires à l'objet que l'on souhaite émuler afin de proposer un système de fichiers le plus similaire possible à celui de son équivalent physique.

Pour ce faire, l'image QEMU Q est émulée itérativement de façon à corriger les erreurs d'émulation trouvées à chaque itération comme illustré dans la figure 7.2. Une itération consiste à émuler Q pendant un laps de temps incrémental.

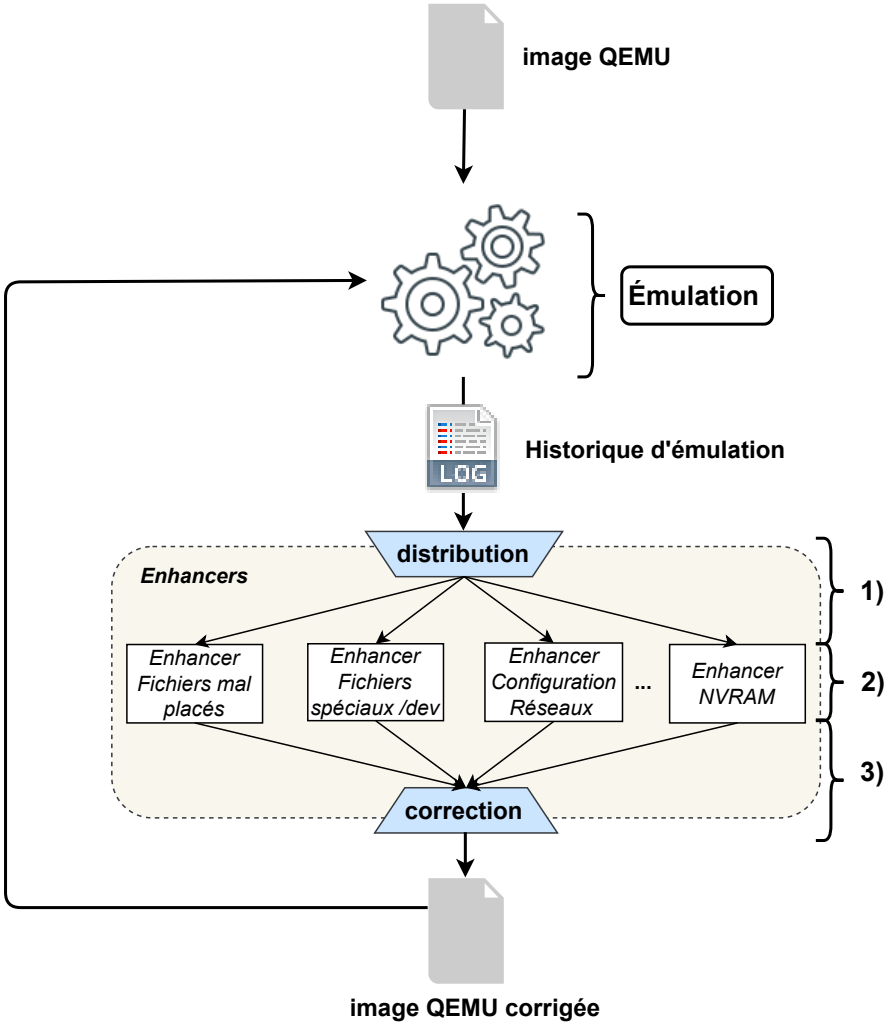


FIGURE 7.2 – Correction des erreurs d'émulation à partir des logs d'émulation associés à une image QEMU

Pour rendre notre approche évolutive, et facilement modifiable, chaque type d'erreur est corrigé par un *Enhancer* spécifique. Ce dernier est en réalité une classe `python` implémentée afin

65. Disponible sur `https://www.kernel.org/doc/Documentation/admin-guide/devices.txt`, dernier accès le 20/04/2022

qu'elle puisse recevoir le log d'une émulation, et appliquer des correctifs sur Q. Il est possible de corriger un nouveau type d'erreur en implémentant un *Enhancer*, et activer ce dernier dans le processus d'émulation.

De manière générale, le processus de correction illustré dans la figure 7.2 fonctionne de la manière suivante après chaque itération :

1) **Distribution de l'historique d'émulation** : les appels système interceptés génèrent des messages de débogages durant l'émulation de l'image Q. L'intégralité de ces messages ainsi que les messages générés par les binaires exécutés dans Q sont transmis à chacun des *Enhancer* activés.
2) **Filtrage des logs et génération des correctifs** : chaque *Enhancer* sélectionne les messages associés au type d'erreur qu'il corrige à l'aide d'expressions régulières. Ensuite, suivant les messages extraits, chacun des *Enhancer* génère les corrections appropriées.
3) **Application des correctifs** : séquentiellement, chaque *Enhancer* applique ses correctifs sur Q.

Une fois tous les correctifs appliqués, Q peut être de nouveau émulé.

Par défaut, notre approche implémente les *Enhancers* suivants :

— *Enhancer Compilation de la librairie partagée* : une librairie partagée est nécessaire pour 1) le bon déroulement de l'émulation (simulation de la NVRAM), et 2) au pot de miel (masquer certaines informations, déchiffrer des paquets réseau). Or, certaines de ces fonctionnalités dépendent du contenu de Q. C'est pourquoi cet *Enhancer* compile la librairie partagée adéquatement.

— *Enhancer NVRAM* : simule le fonctionnement de la NVRAM.

— *Enhancer Fichiers mal placés* : détecte les fichiers manquants, et tente de les créer à partir des autres fichiers présents dans Q ou le firmware original.

— *Enhancer Fichiers spéciaux /dev* : crée les fichiers spéciaux dans `/dev` nécessaires à Q pour fonctionner.

— *Enhancer Fichiers spéciaux /proc* : crée les fichiers spéciaux dans `/proc` nécessaires à 1) Q pour fonctionner, ou 2) au pot de miel pour falsifier certaines informations matérielles.

— *Enhancer Configuration Réseaux* : s'assure de la connectivité de l'objet émulé à la machine hôte, et à l'Internet.

— *Enhancer Exécution de tâches* : permet à un utilisateur de spécifier des tâches à effectuer depuis l'objet émulé. Par exemple, exécuter certains binaires ou modifier le contenu de fichiers, comme `/etc/shadow` ou `/etc/passwd`.

Nous décrivons chacun de ces *Enhancers* dans les sections suivantes.

7.3.4 Compilation dynamique de la librairie partagée

Initialement employée pour uniquement simuler le fonctionnement de la NVRAM, notre librairie partagée, programmée en langage C, intègre d'autres fonctionnalités dépendantes de l'objet émulé, c'est-à-dire du contenu de l'image QEMU Q. C'est pourquoi, au lieu d'insérer la même librairie partagée dans toutes les images QEMU construites, ce premier *Enhancer* compile cette dernière suivant le contenu de l'image QEMU Q. Ce choix nous permet de :

— **Simuler fidèlement la NVRAM** : la NVRAM est généralement utilisé par un objet afin de retrouver et stocker certains paramètres de configuration. Ainsi, en simulant la NVRAM, il est possible d'assigner des valeurs par défaut à certains paramètres (par exemple l'adresse IP ou le nombre d'interfaces réseau), et donc mieux contrôler l'émulation de l'objet.

— **Adapter les fonctionnalités du pot de miel** : certaines fonctionnalités du pot de miel ont des prérequis relatifs aux librairies partagées déjà présentes dans Q, c'est-à-dire les librairies présentes dans des répertoires comme `/lib`, `/usr/lib` et `/usr/local/lib`.

Cette librairie doit être chargée par tous les programmes exécutés par l'objet émulé au cours de son existence afin d'intercepter les appels de fonction utilisant la NVRAM, comme `nvram_get` ou `nvram_set`, et donc retourner les valeurs par défaut assignées. Enfin, les fonctionnalités associées au pot de miel doivent aussi intercepter les appels de certaines fonctions pour 1) cacher l'existence de cette librairie partagée aux attaquants, et 2) implémenter les fonctionnalités décrites dans la section 7.4.

La variable d'environnement `LD_PRELOAD` est employée pour forcer le chargement de cette librairie partagée. Pour ce faire, à l'aide de la méthode décrite dans la section 7.2.3, une instruction est transmise à notre module Linux pour assigner automatiquement notre librairie partagée à la variable d'environnement `LD_PRELOAD` à chacun des nouveaux processus démarrés par l'objet émulé. Contrairement aux approches existantes [36, 169, 90], cette approche permet de donner un nom quelconque à la librairie partagée tout en s'assurant qu'elle sera insérée correctement à la variable d'environnement `LD_PRELOAD` de tous les futurs processus.

7.3.5 Simulation de la mémoire RAM non volatile

La simulation de la mémoire RAM non volatile, ou *Non-volatile random-access memory (NVRAM)*, est effectuée à l'aide d'une librairie partagée dont le fonctionnement est décrit dans la section C.1.4.

Dans notre approche, les couples <clé, valeur> utilisés pour simuler la NVRAM de l'objet émulé sont stockés dans le répertoire d détaillé dans la section 7.3.1. Nous motivons ce choix par la nécessité de cacher à l'attaquant la simulation de la NVRAM.

Enfin, les couples <clé, valeur> ne sont plus codés en dur dans le code de la librairie partagée mais sont simplement listés dans un fichier texte. Ce fichier est ensuite lu par l'*Enhancer* s'occupant de la compilation de la librairie partagée, et les 26 couples présents par défaut sont ajoutés dans le code de la librairie partagée à l'aide de macros en C. L'utilisation d'un fichier texte permet à un utilisateur non averti de modifier facilement le contenu de la NVRAM, et donc le comportement de l'objet émulé.

À la première itération du processus d'amélioration itérative de l'image QEMU Q, cet *Enhancer* inspecte les fichiers présents dans Q, et cherche ceux pouvant contenir des couples <clé, valeur>. Seuls les fichiers dont le nom contient un mot comme `nvram`, `default` ou `config`, sont inspectés, et chaque clé trouvée dans ces fichiers entraîne la création d'un fichier dans d avec comme donnée, la valeur de la clé.

Lors des itérations suivantes, l'objet peut demander la valeur d'une clé inconnue c. Si tel est le cas, notre librairie partagée retourne, par défaut, la valeur 0 afin d'éviter un plantage système. Si, au cours de l'émulation, la valeur x est affectée à c alors, cet *Enhancer*, en inspectant les logs d'émulation, détectera ce nouveau couple $< c,\ x >$ et modifiera Q afin de créer un fichier c dans d avec comme valeur x. Lors de l'itération suivante, la valeur x sera retournée au lieu de 0.

7.3.6 Gestion des fichiers manquants

Au cours de l'émulation, de nombreux fichiers sont ouverts par l'objet émulé dont certains sont nécessaires à l'objet pour son bon fonctionnement. Dans les approches existantes [36, 90], 77 fichiers spéciaux *standards* sont systématiquement créés dans `/dev`, par exemple `/dev/console` ou `/dev/ttyS0`. De plus, des fichiers spéciaux propres à l'IdO, comme `/proc/gpio` ou `/dev/nvram`,

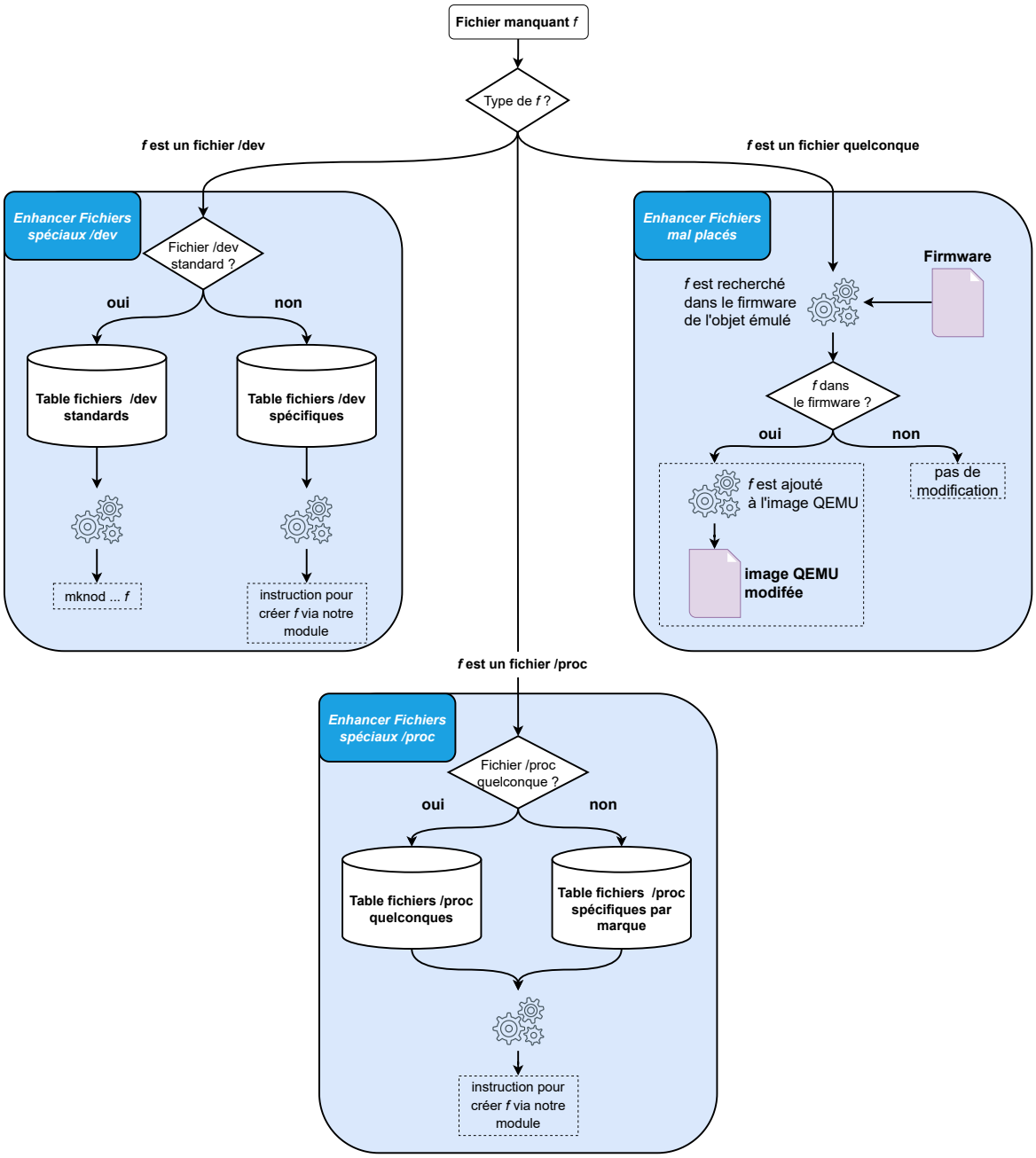


FIGURE 7.3 – Exemple de corrections appliquées lorsqu'un fichier f est manquant

sont également créés automatiquement par leur module afin d'améliorer le processus d'émulation. Cette approche a les désavantages suivants :

— Ajouter de nouveaux fichiers spéciaux propres à certains objets IdO requiert de modifier et recompiler le noyau Linux. Cela représente un problème d'évolutivité.

— Certains fichiers spéciaux sont associés à un fabricant ou type d'objet IdO. Par exemple, le fichier `/dev/sc_led` avec des objets de la marque NETGEAR. Suivant l'objet émulé, certains de ces fichiers sont inutiles, et peuvent notamment être utilisés par un attaquant pour détecter le pot de miel.

Les trois *Enhancers* présentés dans cette section ont pour objectif de modifier l'image QEMU Q de l'objet émulé afin d'y ajouter le minimum de fichiers standards ou spéciaux nécessaires au bon fonctionnement de son émulation, et également falsifier le contenu de certains fichiers de configuration (par exemple `/proc/cpuinfo` ou `/proc/cmdline`) pour la partie pot de miel.

Pour assurer la création des fichiers spéciaux dans `/dev` et `/proc`, nous utilisons quatre tables dans la base de données déployée dans le conteneur Docker d_{db}. Ces dernières peuvent être décrites comme suit :

(1) La première table mémorise les arguments nécessaires à la création des fichiers spéciaux *standards* `/dev`, c'est-à-dire le numéro mineur et majeur, le type et le nom du fichier. Cette table est construite à partir des données de la documentation du noyau Linux [66].

(2) Une table mémorise les données à retourner sur les fichiers spéciaux `/dev` propres à certaines marques.

(3) Une table mémorise les données à falsifier sur les fichiers `/proc` propres à certaines marques.

(4) La dernière table stocke les données à falsifier sur les fichiers `/proc` quelconques.

Les trois dernières tables doivent être complétées manuellement à partir de données provenant, entres autres, d'analyses dynamiques comme [36, 90].

Durant l'émulation de l'objet, les fichiers manquants sont détectés par notre module qui va générer des messages de débogages spécifiques. Ces derniers seront transmis, à chaque fin d'itération, à trois *Enhancers* dont l'objectif est de créer ces fichiers manquants. Le fonctionnement de chacun de ces *Enhancers* est illustré dans la figure 7.3.

Les instructions, et commandes `mknod`, générées par l'*Enhancer Fichiers spéciaux /dev* et l'*Enhancer Fichiers spéciaux /proc* sont ajoutées au script shell *preinit* décrit dans la section 7.3.2. Chacun des fichiers spéciaux `/dev` ou `/proc` créé par notre module se voit affecter une structure `file_operation` contenant les fonctions `read`, `write` mais aussi `ioctl`. Cette dernière est configurable uniquement pour les fichiers spéciaux `/dev` où il est possible de spécifier dans l'instruction transmise, les conditionnelles à utiliser et les valeurs à retourner. Contrairement aux méthodes existantes où l'ajout de fichiers spécifiques dans `/dev` ou `/proc` implique de modifier le module, puis de recompiler le noyau Linux, notre méthode permet ainsi de définir des comportements spécifiques à la fonction `ioctl` sans recompiler le noyau Linux, et donc s'assurer que les appels vers cette fonction retournent des valeurs prédéfinies et propres à chaque fichier `/dev` créé dynamiquement par notre module.

7.3.7 Configuration des interfaces réseau

Comme discuté dans la section 7.2.2, le pot de miel est exécuté depuis un conteneur Docker $d_{emulation}$ afin de rendre ce dernier accessible depuis l'Internet et de l'isoler du reste du système, c'est-à-dire de la machine hôte h. Cette dernière et $d_{emulation}$ diposent d'un volume Docker d_v commun dans lequel un socket UNIX est utilisé par un serveur exécuté depuis h et l'émulateur dans $d_{emulation}$ afin d'échanger des informations, notamment la configuration des interfaces réseau. Dans cette section, nous détaillons les mesures prises par notre approche pour configurer les interfaces réseau dans l'objet émulé o, ainsi qu'entre $d_{emulation}$ et h. Plus précisément, deux niveaux de connectivité sont à configurer : 1) $o \leftrightarrow d_{emulation}$ et 2) $d_{emulation} \leftrightarrow h$. Ces derniers sont présentés dans les deux sections suivantes.

66. Disponible sur `https://www.kernel.org/doc/Documentation/admin-guide/devices.txt`, dernier accès le 20/04/2022

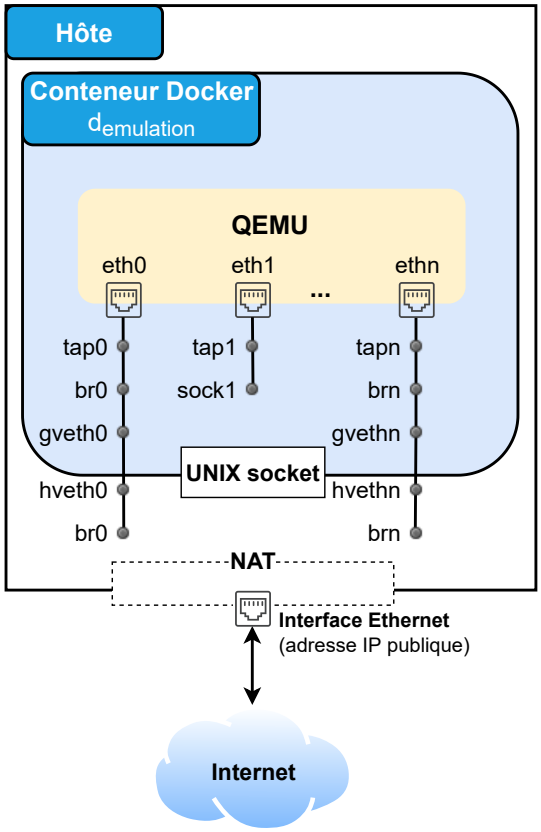


FIGURE 7.4 – Interfaces réseau créées et connectées entre l'objet émulé par QEMU et la machine hôte

7.3.7.1 Connectivité entre l'objet émulé et le conteneur Docker ($o \leftrightarrow d_{emulation}$)

Il n'est pas possible de connaître à l'avance les interfaces réseau utilisés par l'objet émulé o. Ce dernier peut utiliser des interfaces réseau sans-fil, comme `wlan0`. Contrairement aux approches existantes [36, 90], notre méthode configure l'intégralité des interfaces réseau demandées par o. Ce choix est justifié par la partie pot de miel. En effet, par souci de furtivité, o doit avoir ses interfaces réseau configurées aussi fidèlement que possible par rapport à son équivalent physique. Par exemple, un routeur déployant une seule interface réseau `eth0` est suspect. C'est pourquoi ces dernières doivent être 1) présentes, et 2) configurées.

Afin de détecter les interfaces réseau requises par o au cours de son émulation, nous interceptons plusieurs appels système listés et décrits ci-après :

- `ip_rt_ioctl`
- `fib_magic`

} Ajoute, supprime ou modifie une route

- `inet_bind` : retrouve les informations sur les services ouverts
- `register_vlan_dev`
- `register_vlan_device`

} Affectation de VLAN id à une interface

- `br_add_if` :
- `br_dev_ioctl` :

} Création et gestion de pont réseau, ou *bridge* en anglais

— `__inet_insert_ifa` : affectation d'adresse IP

Ces appels système sont interceptés par notre module, et génèrent, à chaque appel, un message contenant les arguments utilisés. L'ensemble de ces messages est ensuite extrait de l'historique d'émulation par l'*Enhancer Configuration Réseaux* pour vérifier si la configuration des interfaces réseau permet ou non une connectivité réseau. Pour ce faire, ce dernier effectue les étapes 2 à 4 illustrées dans la figure 7.5.

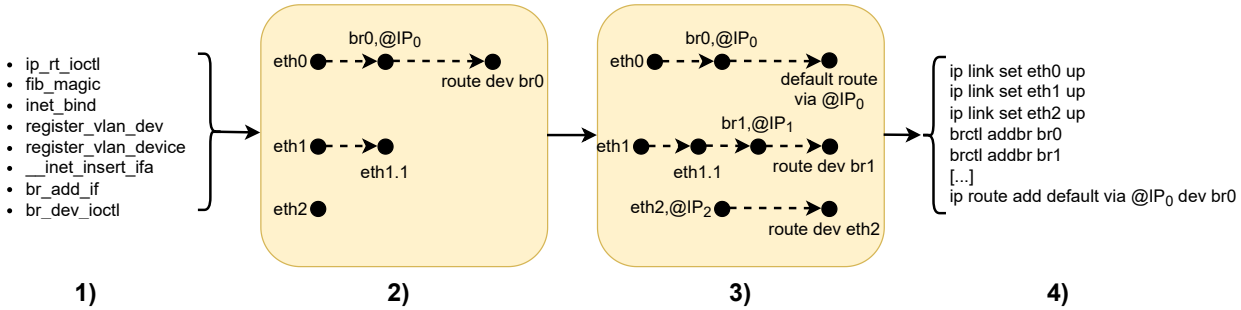


FIGURE 7.5 – Vue globale de l'algorithme implémenté par l'*Enhancer Configuration Réseaux* pour configurer les interfaces réseau d'un objet émulé

Durant l'étape 2), l'*Enhancer* construit un graphe à partir des informations extraites du log d'émulation. Chaque nœud correspond à une interface réseau, un pont (*bridge* en anglais) ou une route, puis les liens suivants sont formés :

— interface ethernet → interface ethernet.{VLAN id}

— interface ethernet.{,VLAN id} → *bridge*

— interface ethernet.{,VLAN id} ou *bridge* → route

À l'image du graphe illustré à l'étape 2) de la figure 7.5, ce dernier peut être incomplet, c'est-à-dire ne pas garantir une connectivité réseau. C'est pourquoi l'étape 3) consiste à exécuter l'algorithme 3 afin de s'assurer qu'une interface réseau possède une adresse IP ainsi qu'une route valide. La première conditionnelle (ligne 5) assigne obligatoirement à une interface ethernet possédant un VLAN un *bridge*, ce qui améliore la connectivité de l'objet émulé selon nos tests. Ensuite, une seconde conditionnelle (ligne 15) garantit que chaque interface ethernet est associée à une route avec une adresse IP valide. L'adresse IP est générée à partir d'une liste prédéfinie de 14 classes d'adresses, comme `10.0.0.0/8` ou `192.168.0.0/16`.

Algorithme 3 Création des nœuds manquants dans le graphe des interfaces réseau

```
1: fonction CompletionGraphe(G, ethifs)
2:     // G ← graphe orienté de la configuration réseau
3:     // ethifs ← liste des interfaces Ethernet
4:     pour ethif dans ethifs faire
5:         si ethif a un VLAN id alors
6:             /* On recherche un nœud bridge sans lien.
7:               Si aucun bridge n'existe alors un bridge est créé */
8:             bridge ← chercher_bridge(G)
9:             ajouter_lien(ethif, bridge, G)
10:        fin si
11:    fin pour
12:    /* L'algorithme suit les liens à partir d'ethif
13:      jusqu'à trouver un nœud sans lien */
14:    dernier_noeud ← parcourir(G,ethif)
15:    si dernier_noeud n'est pas un nœud de type route alors
16:        ip, masque ← generer_ip()
17:        noeud_route ← creer_noeud_route(G, ip, masque)
18:        ajouter_lien(ethif, noeud_route, G)
19:    fin si
20:    // Retourne une version mise à jour de G
21:    retourne G
22: fin fonction
```

Enfin, le graphe orienté obtenu à l'étape 3) est parcouru et les commandes `ip` et `brctl` associées sont générées. Similairement aux approches [169, 90], ces commandes sont insérées dans le script shell `fix.sh` discuté dans la section 7.3.1. Cependant, notre approche exécute ce script minute après le démarrage de l'objet, et non instantanément. Le script est invoqué par notre module, et ne reste pas vivant par souci de furtivité. Le délai d'une minute est nécessaire afin de s'assurer que l'objet émulé ne modifie plus la configuration de ses interfaces réseau au moment où ce script est exécuté.

Une fois les interfaces réseau requises par l'objet émulé o connues, chacune d'elles doit être associée à une interface réseau de type `tap` ou `socket` dans QEMU, par exemple `tap0` ou `socket1` dans la figure 7.4. Les interfaces réseau configurées dans o sont associées à un `tap`, alors que les interfaces non configurées sont associées à un `socket`. En effet, dans certains cas, o configure des interfaces réseau non contiguës (par exemple `eth0` et `eth2`). Nous supposons que les interfaces manquantes existent et sont allouées via un `socket` dans QEMU. L'interface réseau associée à la route par défaut dans o sera celle sélectionnée pour être connectée à l'Internet. Les interfaces réseau sans-fil requises par o sont créées à l'aide de la méthode décrite dans la section C.2.8.

Seules les interfaces `tap` entraînent la création de *bridges* dans $d_{emulation}$ afin de pouvoir les connecter à la machine hôte h. Cette étape est détaillée dans la section suivante, et s'effectue à l'aide du serveur localisé dans h qui reçoit la configuration réseau à implémenter via le socket UNIX présent dans le volume Docker d_v.

7.3.7.2 Connectivité entre le conteneur Docker et la machine hôte $d_{emulation} \leftrightarrow h$

Lors de la création du conteneur Docker $d_{emulation}$, ce dernier est associé à un espace de noms réseau (*network namespace (NS)*). Par défaut, une interface réseau `eth0` est présente dans NS.

Cette dernière est en réalité une paire d'interfaces ethernet virtuelles, ou *virtual ethernet (veth)* en anglais, dont une extrémité (`eth0`) est ajoutée à NS, et l'autre à la machine hôte h.

Les interfaces réseau créées dans la section précédente ne sont pas visibles, ni accessibles, depuis h. Dans le cas des `socket`, ces derniers ne nécessitent pas de connectivité et donc, n'entraînent pas de modifications dans h. Les *bridges* présents dans $d_{emulation}$ ont eux besoin d'être accessibles.

Pour ce faire, le serveur dans h reçoit la configuration réseau attendue par l'objet émulé o via le socket UNIX présent dans le volume Docker d_v, puis pour chaque *bridge* b_i créé dans $d_{emulation}$, effectue les actions suivantes :

1. créée une paire d'interfaces Ethernet virtuelles $< veth_a,\ veth_b >$ et un *bridge* bh_i dans h,
2. associe $veth_b$ au NS et attache $veth_b$ au *bridge* b_i,
3. attache $veth_a$ au *bridge* bh_i,
4. attribue une adresse IP à bh_i.

Enfin, le déploiement en pot de miel requiert les trois fonctionnalités suivantes :

— *NAT* : le trafic réseau d'une interface réseau exposée à l'Internet et redirigé vers un *bridge* bh_j à l'aide de règles `iptables prerouting` et `postrouting`.

— *limitation du débit* : le pot de miel peut être infecté et servir de relai à des attaques de type déni de service. Des règles `iptables` sont implémentées afin de limiter le débit du trafic réseau sortant du pot de miel et ainsi limiter l'impact des attaques propagées par le pot de miel.

— *identification et préventions des attaques* : afin de protéger les autres machines connectées à l'Internet des attaques propagées par notre pot de miel, le système de prévention d'intrusion (IPS) Suricata [67] est configuré avec les règles *emerging threat*.

7.3.8 Modification du comportement de l'objet émulé

Durant l'émulation de l'objet, il est possible que des binaires échouent à démarrer, notamment à cause d'interfaces réseau non connectées. De plus, un objet peut exécuter des règles de pare-feu limitant ainsi les interactions vers certains services. C'est pourquoi exécuter des binaires depuis l'objet émulé permet d'améliorer les performances d'émulation [90]. Afin d'étudier des cyberattaques spécifiques, il peut être nécessaire de modifier le contenu de l'objet émulé. Par exemple, les mots de passe présents dans le fichier `/etc/shadow` peuvent être changés pour éviter les attaques par *brute-force* sur les identifiants et mots de passe.

Par conséquent, notre *Enhancer Exécution de tâches* peut :

— forcer l'exécution de binaires,

— modifier le contenu de fichiers de configuration, comme `/etc/passwd` ou `/etc/shadow`,

— remplacer des fichiers : binaires ou fichiers de configuration.

Pour ce faire, les modifications à effectuer sont spécifiées dans un fichier JSON lu et interprété par l'*Enhancer* qui, 1) modifiera l'image QEMU de l'objet émulé, ou 2) insérera les instructions spécifiques dans le script shell `fix.sh`. L'image QEMU et le script shell sont présentés dans la section 7.3.1. Dans le cas où les chemins d'accès des fichiers ne sont pas connus, ces derniers peuvent être désignés à l'aide d'expressions régulières, par exemple `*bin/iptables` pour retrouver le binaire capable d'éxecuter des commandes `iptables`.

67. Disponible sur `https://suricata.io/`, dernier accès le 22/04/2022

7.4 Fonctionnalités du pot de miel

Notre pot de miel doit enregistrer et exfiltrer les activités des attaquants le plus furtivement possible. Pour ce faire, nous avons implémenté les fonctionnalités listées dans le tableau 7.1. Les objectifs de ces fonctionnalités sont triples :

— **Analyser les cyberattaques** : la fonctionnalité 1 intercepte les activités effectuées par les attaquants depuis le pot de miel. Celle-ci permet d'étudier les différentes commandes saisies par un attaquant au cours d'une attaque, mais aussi d'identifier les touches, ou raccourcis clavier, utilisés. Ces derniers pourront ensuite être analysés pour identifier si l'attaquant est un humain ou un robot. Par exemple, un humain peut saisir la touche `backspace` pour corriger une faute de frappe. Durant l'attaque, il est possible qu'un *malware* soit téléchargé, puis exécuté. Notre approche détecte ces téléchargements en inspectant les opérations d'écriture effectuées sur le pot de miel. Enfin, les commandes saisies et les *malwares* téléchargés sont exfiltrés par le pot de miel à l'aide de paquets réseau. Cette opération est implémentée par la fonctionnalité 2.

— **Cacher la méthode d'émulation** : notre méthode d'émulation utilise une librairie partagée et un répertoire spécifique où sont stockées, entre autres, les clés NVRAM et cette librairie. Ainsi, ces deux éléments doivent être cachés aux attaquants. La fonctionnalité 3 implémente les mécanismes pour cacher la librairie partagée, alors que la fonctionnalité 4 se charge de dissimuler le répertoire. Enfin, les fonctions définies dans notre module Linux sont visibles depuis la liste des symboles (`kallsyms`), et peuvent donc être détectées par un attaquant. C'est pourquoi la fonctionnalité 5 permet de cacher ces fonctions si besoin.

— **Dissimuler les composants matériels ou logiciels émulés** : empiriquement, nous avons observé que les services web d'objets émulés pouvaient contenir des informations erronées ou incomplètes. Par exemple, des adresses MAC affectées à `00:00:00:00:00`, ou un schéma de l'état du réseau affichant un problème de connexion à l'Internet peuvent être visibles sur l'interface graphique du serveur HTTP(S). Un attaquant pourrait utiliser ces informations pour déduire la présence du pot de miel. La fonctionnalité 6 restreint, si possible, l'accès au serveur HTTP(S) à la page d'identification. Une fois connecté au pot de miel, les attaquants peuvent inspecter les composants matériels ou logiciels, comme le processeur ou la liste des modules. Par exemple, obtenir l'architecture processeur du pot de miel en lisant le contenu du fichier `/proc/cpuinfo`. Cependant, le contenu de certains fichiers dans `/proc` peuvent suggérer que l'objet est émulé. La fonctionnalité 7 permet d'assigner des données à ces fichiers. Malgré qu'un objet IdO dispose généralement d'une ou plusieurs interfaces réseau, il n'est pas possible d'en instancier via QEMU. Or, un objet IdO sans interface sans-fil est suspect. La fonctionnalité 8 assure la création de ces fausses interfaces, non fonctionnelles, avec des statistiques (par exemple nombre d'octets envoyés ou reçus) générées aléatoirement.

Numéro	Fonctionnalité	Motivation	Section
1	Interception des actions des attaquants	Trace les activités effectuées par les attaquants depuis n'importe quel terminal du pot de miel. Plus précisément, notre solution est capable d'intercepter chaque touche, ou raccourci clavier, saisi par un attaquant. Cette approche permet d'identifier le type d'un attaquant (humain ou robot). Les téléchargements de *malwares* sont identifiés en analysant les opérations d'écriture effectuées sur le pot de miel sans tenir compte des binaires (`wget`, `curl` etc.) utilisés pour effectuer le téléchargement.	C.2.1
2	Exfiltration des actions des attaquants	Les commandes saisies et les *malwares* téléchargés depuis le pot de miel peuvent être supprimés par les attaquants. C'est pourquoi notre solution exfiltre discrètement ces deux éléments via des paquets réseau.	C.2.2
3	Dissimulation de la librairie partagée	La librairie partagée est essentielle pour la simulation de la NVRAM, et la restriction de l'accès au service web. Cette dernière est insérée dans la variable d'environnement `LD_PRELOAD`, et est donc visible par les attaquants. Le contenu de la variable `LD_PRELOAD` est altéré afin de retirer notre librairie partagée.	C.2.3
4	Dissimulation de répertoires	Les fichiers nécessaires au bon déroulement de l'émulation sont stockés dans un répertoire spécifique qui ne doit pas être détecté par les attaquants. Notre approche cache ce répertoire.	C.2.4
5	Dissimulation de symboles	Les fonctions et variables définies dans notre module peuvent être visibles depuis la liste des symboles `kallsyms`. Notre approche permet de retirer n'importe quelle fonction ou variable de cette liste.	C.2.5
6	Restriction de l'accès au service web	Les informations visibles sur le serveur HTTP(S) peuvent être incomplètes ou erronées. Ces dernières pourraient donc être utilisées par un attaquant pour détecter le pot de miel. Notre approche restreint, si possible, l'accès au serveur HTTP(S) à la page d'identification, et pour ce faire, peut déchiffrer les paquets réseau destinés au serveur HTTP(S).	C.2.6
7	Falsification des données de fichiers de configuration	Les composants matériels ou logiciels (par exemple le processeur ou la liste des modules) peuvent être inspectés par les attaquants à l'aide des fichiers dans `/proc`. Ces fichiers contiennent généralement des données indiquant que l'objet est émulé ainsi, notre solution modifie ces dernières avec des données personnalisables	C.2.7
8	Création de fausses interfaces réseau sans-fil	Un objet IdO dispose généralement d'une ou plusieurs interfaces réseau sans-fil. Ces dernières ne sont pas configurables via QEMU, ainsi notre approche est capable d'en instancier des fausses.	C.2.8

Tableau 7.1 – Récapitulatif des fonctionnalités implémentées dans la partie pot de miel. Les détails techniques associés à chacune de ces fonctionnalités sont présentés dans la section référencée

7.5 Expérimentations et résultats

Dans cette section, nous mesurons les performances de notre approche à émuler des objets IdO à partir de leur firmware, puis présentons les attaques reçues sur le pot de miel.

7.5.1 Environnement expérimental

Nous sélectionnons 1 000 firmwares d'architectures processeur MIPS *little-endian* et *big-endian* de la base de données de firmwares détaillée dans la section 5.5.1 du chapitre 5. La distribution par marque de ces firmwares est présentée dans le tableau 7.2.

Ces 1 000 firmwares ont été émulés sur deux serveurs S_1 et S_2. Le premier fonctionne sur Ubuntu 18.04 LTS avec un processeur Xeon E5-2620v3 2.40GHz et 128Go de RAM, alors que le second fonctionne sur Ubuntu 16.04 LTS avec un processeur Xeon E5-2420v2 2.20GHz et 64Go de RAM. Les firmwares ont été répartis dans S_1 et S_2 afin d'obtenir les résultats d'émulation plus rapidement, et nous n'avons pas noté de différences significatives entre S_1 et S_2 vis-à-vis du temps nécessaire pour émuler un objet IdO. Dans les deux cas, un firmware d'objet IdO peut être déployé en tant que pot de miel en environ 5 minutes (incluant la phase itérative de corrections), ce qui illustre l'avantage de l'émulation pour déployer rapidement un pot de miel.

La construction d'une image QEMU stable à partir d'un firmware, c'est-à-dire l'amélioration itérative présentée dans la section 7.3.3, se fait en trois itérations. Ce nombre a été décidé empiriquement après plusieurs tests sur un faible nombre de firmwares. Les évaluations détaillées ci-après sont effectuées sur les images QEMU stables.

Marque	**Nb. firmwares**
Netgear	279 (27.9%)
Ubiquiti	200 (20.0%)
Tp-Link	131 (13.1%)
Trendnet	131 (13.1%)
D-Link	129 (12.9%)
Asus	96 (9.6%)
Linksys	18 (1.8%)
Edimax	16 (1.6%)
Total	1 000

Tableau 7.2 – Distribution des firmwares par marque

Nous comparons les performances de notre approche à émuler un objet IdO à partir de son firmware avec FirmAE [90], ce dernier étant une version améliorée de FIRMADYNE [36]. Nous ne pouvons pas comparer notre approche avec Honware [169] car son code source n'est pas disponible.

FirmAE étant un outil d'émulation, ce dernier intègre par défaut des fonctionnalités comme le démarrage de services (HTTP : `lighttpd`, `uhttpd`) ou la suppression de règles de pare-feu afin d'accéder aux services de l'objet émulé. Via la méthode présentée dans la section 7.3.8, nous pouvons aussi forcer l'exécution de certains programmes et également supprimer les règles de pare-feu. Ainsi, chacun des 1 000 firmwares sera émulé selon trois modes différents :

— *natif* : le firmware de l'objet IdO est émulé sans forcer le démarrage de binaires.

— *actif* : le firmware de l'objet IdO est émulé et le binaire `sleep` est exécuté afin de forcer l'objet émulé à exécuter un processus. Cette instruction est insérée en parallèle de l'exécution

du script d'initialisation de l'objet émulé.

— *complet* : combine le mode `actif` avec la suppression des règles de pare-feu de l'objet émulé. L'exécution de ces règles de pare-feu s'effectue à l'aide de l'*Enhancer* détaillé dans la section 7.3.8.

Le mode `complet` est le mode le plus similaire à FirmAE, alors que le mode *natif* est le mode à privilégier pour le déploiement en pot de miel. En effet, le binaire `sleep` exécuté par les modes *actif* et *complet* sera visible dans la liste des processus de l'objet émulé, et est donc suspect pour un attaquant.

7.5.2 Émulation d'objets IdO

Dans cette section, nous présentons les performances de notre méthode d'émulation. Dans un premier temps, nous mesurons la connectivité des objets émulés, c'est-à-dire s'ils sont accessibles depuis le réseau. Enfin, nous discutons des services déployés sur les objets émulés.

7.5.2.1 Connectivité des objets émulés

Comme détaillé dans la section 7.2.2, l'objet émulé d est exécuté depuis un conteneur Docker $d_{emulation}$, lui-même exécuté depuis la machine hôte h. Nous vérifions la connectivité de d 100 secondes après son démarrage, et en lui envoyant depuis h 25 paquets *ICMP echo request* toutes les 0.5 secondes. Le pourcentage d'objets émulés accessibles depuis h est présenté dans la figure 7.6.

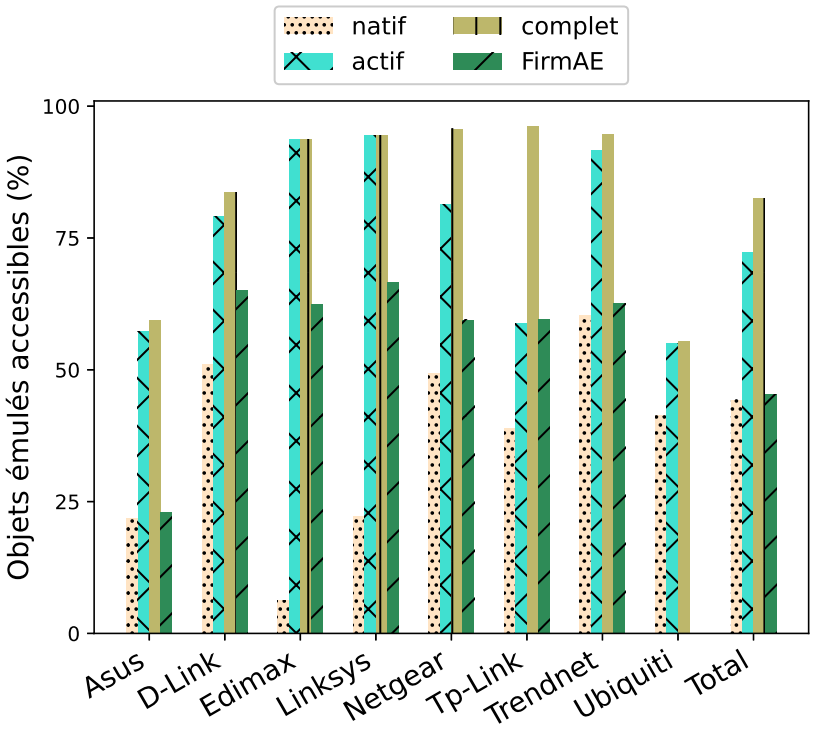


FIGURE 7.6 – Pourcentage des objets émulés accessibles par marque

Pour toutes les marques, nous observons que le pourcentage d'objets accessibles augmente entre le mode *natif* et *complet*. La différence est d'ailleurs flagrante entre les modes *natif* et

actif pour les marques Edimax et Linksys qui voient ce pourcentage augmenter de 6% (1 objet) à 94% (15 objets), et 22% (4 objets) à 94% (17 objets) respectivement. Nous expliquons cette amélioration par l'exécution du binaire `sleep` qui permet d'éviter une erreur de type panique noyau, ou *kernel panic*, et ainsi permettre à notre script shell `fix.sh`, décrit dans la section 7.2.3, de configurer les interfaces réseau.

Comme mentionné dans FirmAE [90], la marque Tp-Link utilise des règles de pare-feu, c'est pourquoi son pourcentage d'objets accessibles augmente entre les modes *actifs* et *complet*, plus précisément de 77 objets (59%) à 126 objets (96.18%).

Seule la moitié des objets appartenant aux marques Ubiquiti et Asus sont correctement émulés et accessibles. Ce phénomène est encore plus marqué pour FirmAE où aucun objet de marque Ubiquiti n'est accessible, et seulement 22 objets (23%) de marque Asus. Nous avons manuellement inspecté les résultats d'émulation de ces deux marques, et avons remarqué que 1) les objets de marque ASUS utilisaient des clés NVRAM dans un encodage non supporté par notre approche [68], alors que 2) les objets de marque Ubiquiti échouent à charger le module `ubnt_platform` ou à terminer leur script d'initialisation, ce qui résulte en une panique du noyau avec l'erreur *attempted to kill init.*

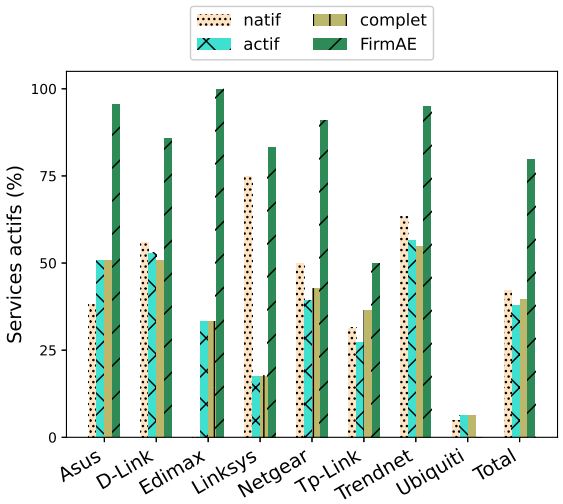
De manière générale, notre approche est capable d'émuler et de rendre accessible sur le réseau, 443 firmwares en mode *natif* donc 44.30% des firmwares peuvent être déployés en tant que pot de miel. D'un autre côté, entre 723 et 825 firmwares ont été émulés et accessibles via les modes *actif* et *complet*, soit respectivement 269 et 371 firmwares de plus que FirmAE.

7.5.2.2 Nombre de services déployés

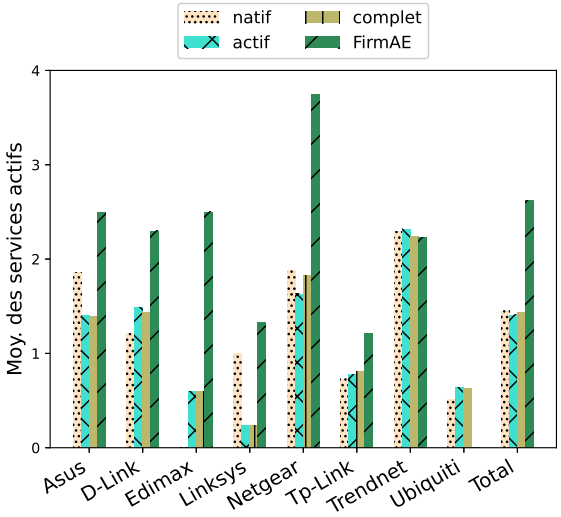
Une fois la connectivité de l'objet mesuré, nous utilisons Nmap [121] pour détecter les services déployés par chaque objet émulé parmi les 1024 ports TCP les plus fréquents recherchés par Nmap.

La figure 7.7(a) illustre le pourcentage d'objets émulés déployant au moins un service. Compte tenu des résultats de leur connectivité, au maximum 7 objets (6.31%) appartenant à la marque Ubiquiti déploient au moins un service actif avec le mode *complet*. Globalement, et peu importe le mode utilisé, environ 40% des objets accessibles déploient au moins un service actif. 381 (83.92%) objets émulés et accessibles par FirmAE [90] déploient au moins un service. Nous observons dans la figure 7.7(b), qu'en moyenne, la différence en nombre de services actifs entre les objets émulés par notre approche et FirmAE est de un. Cette différence peut s'expliquer par le fait que FirmAE force l'exécution de serveur web dans les objets émulés. Malgré ce biais, la figure 7.7(c) montre que les objets correctement émulés par notre approche peuvent déployer plus de services.

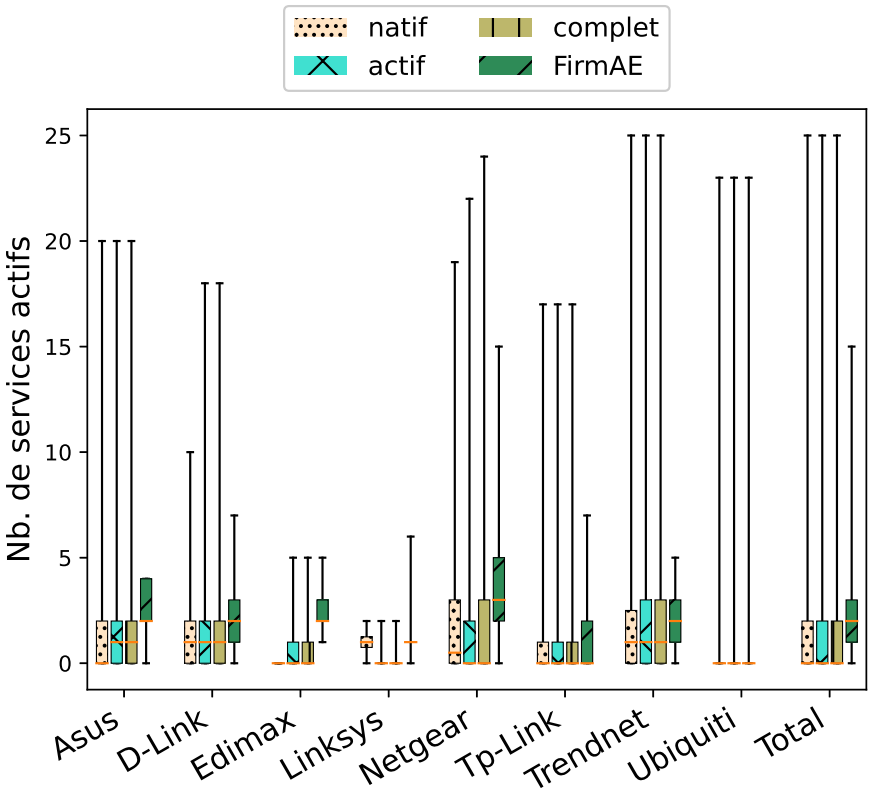
68. lorsque l'option `verbose` est active



(a) Pourcentage d'objets émulés ayant au moins un service actif



(b) Nombre moyen de services actifs dans les objets émulés



(c) Nombre de services actifs dans les objets émulés

FIGURE 7.7 – Services actifs dans les objets émulés

Les performances moindres de notre approche dépendent de la manière de configurer les interfaces réseau de l'objet émulé. En effet, celles-ci sont configurées 60 secondes après le démarrage de l'objet et non au cours du script d'initialisation comme FirmAE. Il est probable que les bi-

naires déployant des services échouent à démarrer lorsqu'aucune interface réseau n'est configurée. Par conséquent, forcer le démarrage de binaires, comme introduit dans [90], est la solution la plus simple pour pallier à ce type d'erreur. Ceci illustre l'importance de notre *Enhancer* de la section 7.3.8 implémentant cette fonctionnalité. Cependant, le choix des binaires à démarrer est propre à chaque utilisateur, c'est pourquoi nous n'avons pas forcé le démarrage de binaires.

7.5.3 Déploiement du pot de miel

Dans les deux expérimentations suivantes, nous présentons les attaques reçues sur nos pots de miel. Plusieurs centaines d'objets IdO ont été émulés et séquentiellement déployés comme pot de miel depuis un serveur appartenant à notre laboratoire. Nous analysons plus précisément les attaques liées aux services : HTTP, TELNET et SSH. Pour le premier, nous étudions les attaques par traversée de répertoires (ou *path traversal* en anglais). Ces dernières sont détectées à l'aide d'expressions régulière, comme `../..*` ou `*/etc/*`, sur les données HTTP reçues par le pot de miel. Par rapport aux services TELNET et SSH, nous analysons les commandes tapées par les attaquants et les virus informatiques téléchargés depuis le pot de miel. Chaque *malware* téléchargé est ensuite envoyé pour analyse auprès de VirusTotal[69] (VT) afin d'identifier son type, et également sa date de publication. Les commandes saisies sont analysées afin d'identifier si l'attaquant est un humain ou un robot. Pour ce faire, nous cherchons dans ces dernières la présence de raccourcis clavier ou de touches individuelles, comme `enter` ou `backspace`.

Notre première expérimentation, entre mai et août 2021, avait pour objectif de configurer et tester les déploiements du pot de miel à l'aide de 500 firmwares sélectionnés aléatoirement. Chacun d'eux a été déployé pour une durée de 3 heures. Ces déploiements ont permis de collecter environ 153 Mo de paquets réseau. 25 308 attaquants uniques ont interagi avec nos pots de miel. Cependant, la majorité d'entre eux scannait les services du pot de miel, et seuls 2 643 (10.44%) des attaquants ont envoyé plus de 10 paquets. Cependant, 6 d'entre eux ont présenté des commandes dans le pot de miel contenant des signes d'activités humaines. Comme énoncé précédemment, cette première expérimentation nous a permis de tester le pot de miel, et corriger certaines erreurs affectant notamment la connectivité du pot de miel au réseau. De plus, les firmwares sélectionnés aléatoirement n'exposaient pas nécessairement des services tels que TELNET ou SSH pour se connecter au pot de miel.

C'est pourquoi dans notre seconde expérimentation, nous avons sélectionné 138 firmwares déployant au moins un des trois services suivants : TELNET, SSH, HTTP(S). Similairement à l'expérience précédente, ces firmwares ont été émulés et déployés séquentiellement pendant 3 heures entre juin 2021 et avril 2022. Durant ce laps de temps, notre pot de miel n'est pas détecté comme tel par l'outil `honeyscore`[70] proposé par Shodan.

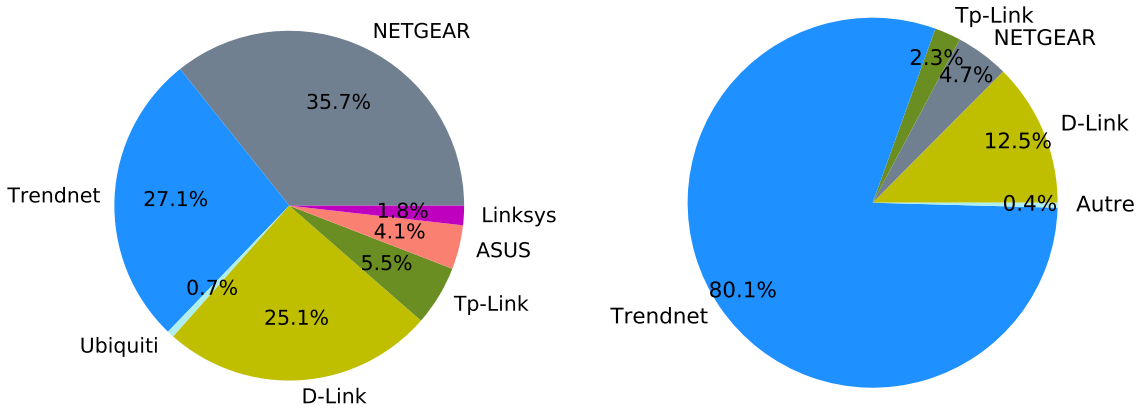
La distribution des objets émulés par marque est visible dans la figure 7.8(a). Sur ce laps de temps, 26.4 Go de paquets réseau ont été collectés. Environ 500 000 adresses IP, dont 172 818 (34.54%) distinctes, ont envoyé au moins un paquet au pot de miel. Seules 19 812 (3.9%) des adresses IP ont envoyé au moins 10 paquets réseau.

Comme présenté dans le tableau 7.3, plus de huit millions (68.98%) de paquets réseau étaient à destination des services TELNET et SSH. Alors qu'environ 16% des objets déploient un serveur HTTP(S), moins de 1% des paquets réseau sont destinés à ce service. Les attaques reçues semblent donc privilégier les services donnant un accès à l'objet. De plus, TELNET et SSH sont connus pour être facilement exploitables à l'aide d'attaques *brute-force*, comme démontré par le botnet Mirai [91, 13].

69. Disponible sur `https://www.virustotal.com/`, dernier accès le 29/04/2022
70. Disponible sur `https://honeyscore.shodan.io/`, dernier accès le 30/04/2022

Service	Port	Protocole	Nb. Paquets
TELNET	23	tcp	5 706 386 (48.66%)
SSH	22	tcp	2 383 433 (20.32%)
HTTP	80	tcp	103 142 (0.88%)
HTTPS	443	tcp	13 123 (0.11%)
NetBIOS	445	tcp	11 668 (0.10%)
DNS	53	udp	7 210 (0.06%)
HTTP	81	tcp	4 083 (0.03%)
NTP	123	udp	2 197 (0.02%)
NetBIOS	139	tcp	1 278 (0.01%)
DNS	53	tcp	1 169 (0.01%)

Tableau 7.3 – Classement des 10 services les plus utilisés par les attaquants



(a) Marques déployées en pot de miel (en %) (b) Attaques reçues (en %)

FIGURE 7.8 – Distribution des marques des objets déployés et leurs pourcentages d'attaques reçues

En complément des services les plus utilisés par les attaquants, nous présentons dans la figure 7.8(b), la proportion des attaques reçues selon la marque de l'objet émulé. Nous observons que les objets émulés de marque Trendnet et D-Link reçoivent plus de 92% des attaques alors qu'ils ne représentent qu'environ 50% des objets émulés. D'un autre côté, les 219 (35.7%) objets de marque NETGEAR ont reçu environ 550 000 (4.7%) interactions. Alors que nos mesures dans la section 7.5.2.2 montrent que NETGEAR, Trendnet et D-Link déploient, en moyenne, le même nombre de services. D'un autre côté, cette différence du nombre d'attaques pourrait s'expliquer par le fait que les objets de marque Trendnet et D-Link soit plus facilement exploitables, c'est-à-dire les identifiants et mots de passe par défaut sont fréquents et utilisés par les méthodes *brute-force*. Le classement des 10 modèles les plus attaqués est présenté dans le tableau 7.4.

Nous avons ensuite localisé l'origine géographique des attaquants, et remarqué que le tiers d'entre eux sont originaires des États-Unis, puis de Chine (9.7%), et d'Allemagne (5.4%). L'origine d'un attaquant est identifiée à l'aide de son adresse IP et de la base de données des localisations d'adresses IP : `GeoIP2`[71]. Nous présentons l'origine par pays des attaquants dans la

71. Disponible sur `https://github.com/maxmind/GeoIP2-python`, dernier accès le 30/04/2022

Rang	**Marque**	**Modèle**	**Type**	**Service(s)**
1	Trendnet	TI-G160WS	commutateur	SSH, HTTP(S), TELNET, SNTP
2	Trendnet	TEW-455APBO_v2	point d'accès sans-fil	TELNET
3	Trendnet	TEW-676APBO	point d'accès sans-fil	TELNET
4	Trendnet	TEW-714TRU	routeur	TELNET
5	Trendnet	TI-PG541i	commutateur	SSH, HTTP, TELNET, SNTP
6	D-Link	DIR-505L	routeur	HTTP, FTP, DNS, UDHCP, LLMNR, TELNET, Avahi, NetBIOS, Microsoft SQL Server
7	Trendnet	TEW-823DRU	routeur	SSH, DNS, DHCP
8	Trendnet	TEW-738APBO	point d'accès sans-fil	TELNET, HTTP
9	Tp-Link	RE450	point d'accès sans-fil	SSH, HTTP, TDDP
10	D-Link	DAP-2330	point d'accès sans-fil	TELNET

Tableau 7.4 – Classement des 10 objets émulés les plus attaqués

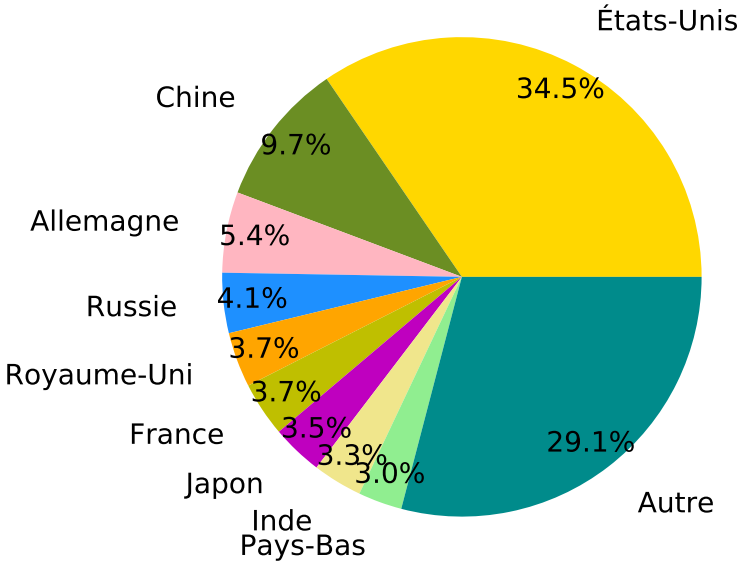


FIGURE 7.9 – Répartition des attaquants par pays

figure 7.9, les pays comptabilisant moins de 3% des attaquants sont labellisés comme "Autre". Cette méthode échoue à identifier 613 attaquants. Nous remarquons que les 136 968 (29.1%) attaquants restants sont localisés dans 215 *pays*, nous expliquons cela par 1) l'utilisation de VPN par les attaquants afin de masquer leur réelle origine, 2) l'adresse IP de l'attaquant peut être falsifiée. En effet, pour propager des attaques rapidement, notamment celles destinées au serveur HTTP(S), les commandes pour télécharger et exécuter un *malware* sont présentes dans une seule requête ainsi, l'adresse IP source utilisée par l'attaquant peut être falsifiée.

Par rapport aux attaques ciblant le serveur HTTP(S), 117 014 (1%) connexions ont été

observées vers le pot de miel, et initiées par 11 391 attaquants dont 6 440 distincts. Parmi ces connexions, 1 897 (1.6%) sont associées à des attaques de type *path traversal* initiées par 57 attaquants. Ces derniers ont effectué un total de 841 requêtes HTTP contenant des données distinctes et détectées comme malveillantes. Malgré la restriction de l'accès au serveur HTTP, nous observons des attaquants tenter d'exploiter le serveur HTTP du pot de miel en envoyant plusieurs requêtes malveillantes. Cela confirme que les réponses générées par notre pot de miel sont convaincantes et que le pot de miel est considéré comme une vraie cible potentielle. Enfin, 6 des 10 requêtes malveillantes les plus reçues essaient d'accéder au contenu du fichier `/etc/passwd` pour obtenir les identifiants et mot de passe du pot de miel.

Les commandes saisies par les attaquants dans les terminaux du pot de miel ont été analysées, et 72 attaquants ont été identifiés comme humain. Nous avons notamment observé l'utilisation erronée du raccourci `CTRL+W` ou de la touche `echap` pour quitter l'interface de connexion TELNET. Une fois connecté au pot de miel, les attaquants ont vérifié son environnement mais aucune commande observée n'est capable de détecter le pot de miel. D'ailleurs, les attaquants cherchent principalement un répertoire depuis lequel télécharger et exécuter un *malware*. Ainsi, notre pot de miel est resté totalement invisible auprès des attaquants.

Ces derniers ont téléchargé 921 *malwares* dont 35 distincts depuis le pot de miel. Cette différence reflète donc qu'à plusieurs reprises notre pot de miel a été compromis par un même attaquant. Nous avons donc reporté ces *malwares* à VT, et noté que 11 (31.42%) étaient inconnus mais bien détectés comme malveillants par VT. De plus, 15 (42.85%) autres étaient connus mais reportés moins de 7 jours avant notre pot de miel. Parmi les *malwares* téléchargés par les attaquants, 34 (97.14%) sont des variantes du *malware* associées au botnet Mirai, et 1 (2.85%) *malware* est lié au botnet Hajime [74]. Ainsi, notre pot de miel s'est montré capable d'intercepter des attaques à large échelle récentes, ce qui confirme qu'il est bien détecté comme un objet IdO. En effet, nous pouvons supposer que lors de la phase de reconnaissance effectuée par les attaquants pour sélectionner des cibles, notre pot de miel a répondu favorablement à un certain nombre de critères faisant de lui une cible convaincante.

7.5.4 Discussion

7.5.4.1 Considération éthique

Notre pot de miel étant à forte interaction, un attaquant peut effectuer tout type d'actions depuis ce dernier. C'est pourquoi, une fois infecté, le pot de miel peut transmettre un *malware* à d'autres machines connectés à l'Internet, ou participer à des attaques par déni de service. Afin de limiter les risques de propagation d'attaques, nous avons limité le débit sortant du pot de miel à 100 paquets par seconde[72], et activé le système de prévention d'intrusion Suricata. Ces deux mesures sont détaillées dans la section 7.3.7.2. De plus, chaque pot de miel est déployé sur une période de trois heures, ce qui limite également le temps disponible à l'attaquant pour propager ses attaques.

7.5.4.2 Améliorations des performances d'émulation

D'après l'expérimentation présentée dans la section 7.5.2.1, 443 objets (44.3%) des objets émulés peuvent être déployés en tant que pot de miel, alors que plus de 723 (72.3%) objets sont émulés et accessibles depuis le réseau. Ces derniers requièrent néanmoins l'exécution du binaire `sleep`, et donc la création d'un processus, ce qui réduit la furtivité en cas de déploiement en

72. Cette valeur est paramétrable, et peut dépendre du type de connexion (`NEW`, `ESTABLISHED` et `RELATED`)

pot de miel. Par défaut, le binaire `sleep` n'est pas exécuté, et ce choix est laissé à l'utilisateur. Dans notre cas, nos pots de miel n'exécutaient pas ce binaire afin de maintenir un haut niveau de furtivité.

Dans la section 7.5.2.2, certains objets émulés déploient peu de services. Pour pallier ce problème, une fonctionnalité capable de forcer le démarrage de binaires à été présentée dans la section 7.3.8. Cette dernière permet à un utilisateur, par le biais d'un fichier en format JSON, de spécifier les binaires à exécuter sur l'objet émulé. Cette implémentation est 1) plus flexible que FirmAE [90] car les binaires à démarrer ne sont pas codés en dur, et 2) l'exécution des binaires se fait d'une manière compatible avec le déploiement en pot de miel.

Forcer le démarrage de certains services permet d'augmenter l'exposition d'un objet émulé, et donc d'un pot de miel, afin de recevoir plus d'attaques. Cependant, suivant les services déployés, un attaquant averti pourrait détecter le pot de miel. En effet, un trop grand nombre de services ou des services incohérents avec le type de l'objet émulé, réduiraient la furtivité du pot de miel. Par conséquent, forcer le démarrage d'un binaire, et donc service, doit être effectué avec parcimonie.

7.6 Synthèse

Suite aux observations des chapitres 5 et 6 relatant la faible sécurité des objets IdO, l'objectif de ce chapitre était d'étudier les cyberattaques ciblant des objets IdO et ainsi, vérifier si des vulnérabilités spécifiques étaient exploitées par des attaquants. Pour ce faire, notre contribution présente une nouvelle infrastructure logicielle permettant l'émulation, puis le déploiement en pot de miel, d'objets IdO à partir de leurs firmwares. Cette approche s'inspire de la méthode d'émulation présentée dans [36] puis étendue dans [90] que nous avons modifié afin 1) d'améliorer les performances d'émulation et 2) intégrer les fonctionnalités inhérentes à un pot de miel.

L'infrastructure logicielle proposée permet une correction itérative des erreurs d'émulation jusqu'à obtenir un objet émulé sans erreur critique. Les erreurs sont corrigées par des *Enhancers* dont le rôle est de corriger un type d'erreur spécifique, par exemple créer les fichiers manquants ou configurer les interfaces réseau de l'objet émulé. Cette approche permet d'intégrer plus facilement de nouvelles fonctionnalités, c'est-à-dire des actions ou corrections à effectuer sur l'objet émulé, et est ainsi plus évolutive que les approches existantes.

Cependant, certaines erreurs d'émulation liées à l'absence de composants matériels spécifiques (NVRAM, `/dev/gpio`) nécessitent d'être corrigées depuis les espaces noyau et utilisateur. Pour ce faire, notre approche utilise 1) un module Linux capable dynamiquement d'instancier de faux composants matériels, 2) une librairie partagée dont le rôle est, entre autres, de simuler le fonctionnement de la NVRAM. De plus, chaque *Enhancer* peut interagir avec notre module à l'aide d'instructions spécifiques, et ainsi effectuer des actions depuis l'espace noyau.

La méthode d'émulation utilisée dans [36, 90] modifie le contenu du firmware de l'objet à émuler. Ainsi un attaquant peut facilement détecter la présence du pot de miel en vérifiant, par exemple si la librairie partagée `libnvram.so` est chargée dans la variable d'environnement `LD_PRELOAD`. C'est pourquoi, la correction les erreurs d'émulation, et plus globalement le processus d'émulation, sont effectués en prenant en compte ce besoin de furtivité. Contrairement à Honware [169], notre module Linux intègre également des fonctionnalités permettant de cacher la partie émulation, et est capable de capturer et d'exfiltrer les activités, dont les téléchargements de *malwares*, effectuées par les attaquants.

Nous avons émulé 1 000 firmwares d'objets IdO ayant une architecture processus MIPS *little-endian* ou *big-endian*, et montré que seuls 443 (44.3%) firmwares pouvaient être déployés nativement comme pot de miel. En forçant l'exécution de binaires, comme la suppression de

règles de pare-feu, plus de 723 (72.3%) firmwares peuvent être émulés et accessibles depuis le réseau soit environ 300 firmwares de plus que FirmAE [90].

Entre mai 2021 et avril 2022, nous avons déployé en tant que pot de miel plus de 600 objets IdO. Chacun de ces objets, et donc pots de miel, mettait à disposition au moins l'un de ces services : TELNET, SSH ou HTTP(S). La majorité des attaques (68.98%) ciblaient les services TELNET et SSH, alors que moins de 1% des attaques étaient dirigées vers HTTP(S). Cependant, de nombreuses attaques de type *path traversal* ont été observées malgré que le serveur HTTP(S) du pot de miel soit restreint. Certaines d'entre elles étaient consécutives, ce qui indique que les attaquants considèrent les réponses générées par le pot de miel comme étant réalistes.

Enfin, malgré son déploiement sur un seul serveur, notre pot de miel a également reçu de nombreuses attaques ayant entraîné le téléchargement de plus de 900 *malwares* dont 35 distincts. Une analyse de ces derniers par VirusTotal (VT) a montré que 26 (74.29%) d'entre eux étaient inconnus ou récents, c'est-à-dire publiés moins de 7 jours avant nous sur VT.

Les commandes saisies par les attaquants depuis le pot de miel ont été analysées, et 72 attaquants ont saisi des touches individuelles (`echap`) ou des raccourcis claviers (`CTRL+W`) laissant penser que ces derniers sont des humains. En combinant ces observations avec le fait que 1) notre pot de miel n'était pas considéré comme tel par l'outil `honeyscore` de Shodan, et 2) qu'aucune commande saisie par les attaquants n'était capable de détecter notre pot de miel ; nous avons montré que notre solution est capable de simuler le comportement d'un objet IdO physique, et d'observer des attaques récentes d'humains, et de botnets ciblant ces objets tout en maintenant un haut niveau de furtivité.

Conclusion générale

L'avènement de l'Internet des Objets (IdO) dans notre environnement personnel ou professionnel a bouleversé notre quotidien mais également introduit de nouveaux risques de sécurité. D'un côté, les objets IdO peuvent souffrir de vulnérabilités et être compromis. De l'autre, leur trafic réseau, chiffré ou non, peut être analysé par des attaquants et présenter un risque vis-à-vis de notre vie privée.

Dans cette thèse, nous nous sommes intéressés à ces deux aspects, en nous focalisant premièrement sur les problèmes de fuite de données d'une box domotique. Notre première contribution présente une méthode capable d'inférer les actions utilisateur. Dans une seconde contribution, nous avons présenté une méthode d'identification active élaborée à partir des données d'une analyse à grande échelle de firmwares d'objets IdO. Cette dernière peut être utilisée par un attaquant pour détecter des objets IdO, et surtout, identifier des propriétés liées à ces objets afin d'augmenter ses chances de compromettre l'objet ciblé tout en réduisant au maximum le nombre d'interactions.

Dans un second temps, nous nous sommes intéressés à la sécurité des objets IdO au travers des contributions suivantes : l'analyse de la composition des binaires présents dans les firmwares d'objets IdO, et le pot de miel à forte interaction simulant le fonctionnement d'un objet IdO à partir de son firmware. Notre analyse de firmwares est originale car elle se focalise sur les vulnérabilités connues et présentes dans les firmwares, ou bien le niveau d'obsolescence des binaires. Cela nous permet ainsi de mettre en lumière des tendances globales dans les pratiques de développement des fabricants d'objets IdO. Le pot de miel nous permet d'étudier les vulnérabilités exploitées par les attaquants afin de compromettre des objets connectés. Ces deux contributions ont été proposées en prenant compte les contraintes évoquées dans l'introduction.

Résumé des contributions

Nous résumons ci-dessous chacune de nos quatres contributions.

Analyse en boîte-grise des communications d'une box domotique

L'utilisation d'une box domotique permet à un utilisateur d'interagir avec un ou plusieurs de ses objets IdO, via une seule application mobile ou interface web. Ainsi, la box domotique sert de relai entre l'utilisateur et les objets IdO, et donc de point de passage aux requêtes à destination ou en provenance des objets IdO. L'objectif de cette contribution était de mesurer le niveau d'information qu'un attaquant localisé dans l'Internet est capable d'inférer à partir des requêtes destinées à la box. Plus précisément, nous souhaitions identifier les actions utilisateur effectuées sur les objets IdO.

Comparé aux travaux existants, notre positionnement ne permet pas d'intercepter les communications sans-fil des objets IdO. Ainsi, notre approche est confrontée aux deux challenges

suivants : la présence de chiffrement dans les paquets reçus et émis par la box domotique, et l'impossibilité d'identifier le ou les objets IdO destinaires d'une action utilisateur à partir du trafic réseau de la box.

Lorsqu'un utilisateur déclenche une action vers un ou plusieurs objets IdO associés à une box domotique, les requêtes effectuées par l'utilisateur ne sont pas directement transmises à la box domotique mais plutôt à un serveur, localisé dans le cloud, et dont le rôle est de retransmettre la requête à la box. En d'autres termes, deux requêtes sont nécessaires pour transmettre une action utilisateur à un objet IdO : une requête utilisateur-serveur puis une requête serveur-box.

Suivant les actions utilisateur et le nombre d'objets IdO concernés par ces dernières, nous avons observé que les tailles des données chiffrées des requêtes serveur-box variaient, mais restaient constantes si une même combinaison d'actions utilisateur était effectuée. Ainsi, nous avons supposé que la taille des données pouvait être utilisée pour inférer les actions utilisateur.

Pour ce faire, nous avons construit une méthode d'identification passive, 1) interceptant une requête serveur-box, et 2) décomposant la tailles des données chiffrées en une liste de combinaisons d'actions utilisateur possibles. Notre approche permet ainsi de décomposer la taille des données chiffrées en une liste de composants, c'est-à-dire d'actions utilisateur possibles. Par conséquent, contrairement à une méthode d'apprentissage automatique, notre solution ne requiert pas d'apprendre toutes les combinaisons d'actions utilisateur possibles.

Nous avons appliqué notre approche sur une box domotique associée à 16 objets IdO, et montré qu'un attaquant localisé dans l'Internet peut déduire les actions utilisateur avec une précision de 91.2%. Dans le cas de la box domotique étudiée, nous ne pouvons pas déduire la combinaison d'actions utilisateur exacte, car plusieurs combinaisons d'actions génèrent des tailles des données serveur-box équivalentes. Cela montre d'ailleurs que notre méthode peut inférer des combinaisons d'actions jamais vues auparavant.

Cependant, notre approche a plusieurs limites :

— nous avons supposé une similarité entre les données présentes dans les utilisateur-serveur et serveur-box.

— L'algorithme de chiffrement utilisé lors des requêtes serveur-box est déterministe. En effet, si l'algorithme de chiffrement change, il n'est pas possible de dériver la taille d'une action utilisateur.

Analyse hybride de firmwares d' objets IdO

Afin d'évaluer la sécurité des objets IdO disponibles sur le marché, il n'est pas possible d'acquérir physiquement tous ces derniers puis d'effectuer des tests de vulnérabilités sur chacun d'eux. D'un autre côté, les fabricants d'objets IdO mettent souvent à disposition sur leur site web, les firmwares utilisés dans leurs objets.

À partir d'un firmware, il est possible d'obtenir la liste des fichiers de configuration et binaires. Dans l'état de l'art, ces derniers sont généralement analysés par des méthodes d'analyse dynamique ou statique afin de détecter de nouvelles vulnérabilités.

Contrairement aux travaux existants, notre contribution n'a pas pour objectif de détecter de nouvelles vulnérabilités mais d'évaluer le niveau de sécurité d'un firmware à l'aide de trois métriques : le niveau d'exposition, le niveau d'obsolescence des binaires et le délai post-vulnérabilité. La première métrique retourne le nombre de binaires capables de déployer un service. La seconde métrique vérifie si les binaires inclus dans un firmware sont à jour, alors que la troisième métrique inspecte la présence de vulnérabilités connues dans le firmware sur des binaires prédéfinis.

Ces trois métriques nécessitent d'extraire la liste des binaires présents dans un firmware mais également d'identifier le nom et la version des binaires. Pour obtenir ces informations, nous avons

défini une méthode d'analyse de firmwares d'objets IdO combinant une analyse dynamique, et une analyse statique. L'intérêt de combiner ces deux méthodes est de maximiser nos chances d'extraire des informations d'un binaire. Par exemple, dans le cas d'une analyse dynamique, il n'est pas toujours possible d'exécuter des binaires à cause de l'absence de certains composants matériels. De plus, notre approche doit être capable de s'adapter aux architectures processeur (par exemple MIPS, ARM) rencontrées au cours d'une analyse.

Enfin, nous avons appliqué notre approche sur 4 730 firmwares d'objets IdO publiés entre 2009 et 2019. Nous avons noté que le niveau d'exposition des objets IdO a augmenté au cours de cette période jusqu'à atteindre 4 en 2019. Ainsi, les objets IdO exposent ainsi de plus en plus de services, ce qui augmente la surface d'attaque possible et donc pénalise leur sécurité. D'un autre côté, nous avons étudié le délai post-vulnérabilité et le niveau d'obsolescence des binaires utilisés pour déployer des services HTTP(S) et SSH, et mis en lumière l'utilisation massive de binaires, comme `lighttpd` pour HTTP(S), et `dropbear` pour SSH. Dès 2017, nous avons noté que les binaires associés à ces deux services étaient plus récents et moins vulnérables à des attaques connues. À partir de 2017, il semblerait donc que les fabricants d'objets IdO aient mieux considéré les risques de sécurité lors du développement de leur firmware. Cependant, notre analyse ne nous permet pas de confirmer les raisons de ce changement décisionnel. Nous pouvons supposer que les nombreuses attaques provenant de botnets, comme Mirai ou Bashlite, en 2016, ont pu motiver certains fabricants à améliorer la sécurité de leurs objets, même si ces botnets ciblent surtout le service TELNET.

Identification active d'un objet IdO

Au cours d'une attaque, la phase de reconnaissance est utilisée par un attaquant pour détecter des objets respectant les critères d'une attaque. Cette phase est généralement implémentée à l'aide d'une méthode d'identification active. Suivant la précision des critères à obtenir, la phase de reconnaissance peut générer énormément d'interactions avec la machine cible, ce qui peut réduire la furtivité de l'attaque.

Dans le contexte de l'Internet des Objets, il est difficile d'établir une méthode d'identification active étant donné le grand nombre d'objets IdO disponibles sur le marché et leur forte hétérogénéité dans leur fonctionnalité. De plus, les méthodes d'identification actives existantes ne garantissent pas d'identifier des propriétés, comme le nom ou la version, des binaires déployant des services.

La méthode d'identification active proposée dans notre troisième contribution est construite à partir des données extraites de firmwares d'objets IdO, et a pour objectif d'identifier les propriétés d'un objet IdO connecté suivantes :

— sa marque ;

— le nom des utilisateurs présents ;

— les mots de passe associés à ces utilisateurs ;

— le nom (et la version) du binaire déployant un service donné.

Chacune de ces propriétés est inférée à l'aide d'un classificateur entrainé spécifiquement à cette tâche. Afin de garantir la furtivité de notre approche, nous avons décidé d'utiliser comme données initiales les résultats supposés d'un scan de ports TCP/UDP, car ce type de scan est bénin et éveille peu les soupçons. Les résultats de ce scan de ports sont simulés à partir des binaires capables de déployer des services présents dans le firmware. En effet, nous supposons que ces derniers sont exécutés, par défaut, par l'objet IdO. Par exemple, si le binaire `telnetd` est présent dans un firmware, nous déduisons que l'objet IdO correspondant déploie le service `TELNET`.

À l'aide d'un premier classificateur entraîné à prédire la marque d'un objet à partir des résultats d'un scan de ports limité à huit services, nous avons montré qu'il était possible, dans 76.85% des cas, de prédire correctement la marque d'un objet. Une fois cette nouvelle connaissance obtenue, nous avons montré qu'il était possible de prédire dans plus de 89% et 73% des cas le nom ou la version des binaires utilisés pour déployer les services DNS, HTTP ou SSH. Enfin, contrairement aux solutions existantes, notre approche est capable d'identifier au moins un nom d'utilisateur ou un mot de passe valide et présent dans un objet connecté dans plus de 97% des cas. Les performances de notre approche sont variables suivant la marque à laquelle appartient un objet.

Notre solution est théorique, et n'a pas été vérifiée sur des objets IdO physiques. Par conséquent, il est possible d'un objet IdO ne déploie pas automatiquement tous ses services disponibles. D'ailleurs, il n'est pas garanti que lors d'un scan de ports TCP/UDP, les ports ouverts par l'objet correspondent aux services énoncés par l'IANA [73].

Ainsi, pour se protéger d'une telle attaque, il est nécessaire de fausser les résultats d'un scan de ports. En guise de contre-mesures, les fabricants d'objets IdO pourraient 1) utiliser des ports non standards ou dynamiques pour déployer les services, 2) ajouter de faux services sur des ports standards.

Implémentation d'un pot de miel à forte interaction pour objets IdO

Afin d'étudier les cyberattaques ciblant les objets IdO, nous avons présenté dans notre quatrième contribution, un pot de miel à forte interaction simulant le fonctionnement d'objets IdO. Pour ce faire, nous nous sommes inspirés de la méthode d'émulation d'objets IdO initialement présentée par FIRMADYNE [36] et étendue dans FirmAE [90], que nous avons modifié afin d'améliorer les performances d'émulation, et intégrer des fonctionnalités propres à un déploiement de pot de miel. Un pot de miel similaire a été proposé dans [169] mais celui-ci n'intègre pas le besoin en furtivité d'un pot de miel.

Trois éléments sont nécessaires à l'émulation d'un objet IdO : un firmware, un noyau et un module Linux. Dans les méthodes existantes [36, 169, 90], des fichiers sont ajoutés au firmware, et le module Linux implémenté a pour rôle de créer, par défaut, un grand nombre de fichiers spéciaux, par exemple `/dev/nvram` ou `/dev/gpio`, afin d'améliorer le processus d'émulation.

Dans notre approche, nous limitons ces modifications afin de rendre l'objet émulé aussi furtif que possible tout en garantissant des performances d'émulation supérieures aux méthodes existantes. En effet, un objet émulé avec un ensemble incohérent de fichiers spéciaux peut être considéré comme suspect par un attaquant averti. C'est pourquoi, il est impératif de construire un objet IdO émulé aussi similaire que possible de son équivalent physique.

Pour ce faire, nous avons défini une nouvelle infrastructure logicielle permettant une correction itérative des erreurs d'émulation, puis de déployer l'objet émulé en tant que pot de miel.

En complément de l'infrastructure logicielle proposée, nous avons redéfini le module Linux présenté dans [36, 90] pour y intégrer des mécanismes capables de 1) cacher aux attaquants certaines modifications appliquées à l'objet émulé, 2) créer dynamiquement de faux composants matériels, par exemple `/dev/nvram` ou `/proc/led`, à partir de requêtes générées par un *Enhancer*, et 3) intercepter puis exfiltrer les activités des attaquants effectuées depuis le pot de miel.

L'évaluation de notre approche se fait en deux parties. Dans la première partie, nous avons comparé nos performances d'émulation avec celles de FirmAE [90] et montré que les objets émulés

73. Disponible sur `https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml`, dernier accès 10/05/2022

par notre approche sont plus souvent accessibles sur le réseau mais déploient moins de services comparés à ceux de FirmAE.

Cependant, obtenir des performances supérieures à FirmAE se fait au détriment de la furtivité de l'objet émulé, et donc ne permet pas de déployer ces derniers comme pot de miel. En effet, sur les 1 000 firmwares testés, seuls 443 (44.3%) peuvent être émulés sans dégrader la furtivité contre 825 (82.5%) sans cette contrainte.

Dans la seconde évaluation, nous avons déployé plusieurs centaines d'objets émulés en tant que pots de miel sur un seul serveur pendant environ un an. Nous avons capturé plus de 900 virus informatiques (35 distincts) dont 11 (31.42%) étaient inconnus de la base de données de *malwares* VirusTotal (VT). En complément, nous avons observé des attaques venant de 72 attaquants humains, et noté qu'aucune commande effectuée par les attaquants depuis le pot de miel ne permettait de détecter la présence de ce dernier.

Le déploiement de notre pot de miel sur un unique serveur a sensiblement réduit le nombre d'attaques reçues. De plus, ce serveur appartient à notre laboratoire est dispose donc d'une adresse IP reflétant l'appartenance à notre institution, ce qui peut également expliquer le faible nombre d'attaques complexes reçues. Par exemple, le botnet Mirai ne propage pas son attaque à certaines plages d'adresses IP. Afin d'éviter de recevoir continuellement des attaques d'un même botnet, les déploiements de notre pot de miel sont espacés d'une heure et demi.

Perspectives

Les travaux présentés dans cette thèse ouvrent plusieurs perspectives notamment dans les domaines de l'analyse passive de trafic réseau de box domotique, l'analyse active d'objets IdO et l'étude de cyberattaques d'objets IdO via pot de miel.

Décomposition du trafic réseau d'une box domotique

Contrairement aux travaux existants sur l'identification passive des actions utilisateur présentés dans le chapitre 1, notre contribution s'est focalisée sur le trafic réseau d'une box domotique contrôlant un à plusieurs objets IdO. Dans cette configuration, un usager s'appuie généralement sur une seule application mobile pour interagir avec les objets IdO. Ainsi, une première requête est envoyée de l'utilisateur à un serveur localisé dans le cloud qui, à son tour, envoie une requête vers la box domotique contenant le ou les actions utilisateur demandées.

Nous avons supposé une similarité entre les données envoyées de l'utilisateur au serveur, et celles présentes dans la requête serveur-box. C'est à l'aide de cette relation que notre méthode est capable d'inférer le contenu de la requête serveur-box malgré le chiffrement de données. L'évaluation de notre contribution était limitée à une seule box domotique. Il conviendrait d'appliquer notre approche sur d'autres box domotique, comme celles implémentant plusieurs protocoles de communication sans-fil, comme Z-Wave ou ZigBee. De plus, dans notre cas d'étude, les objets IdO appartenaient à la même marque que la box domotique. Par conséquent, il serait également pertinent d'étudier comment les requêtes contenant des combinaisons d'actions utilisateur destinées à des objets IdO de marques différentes sont transmises de l'utilisateur au serveur, puis du serveur à la box domotique.

Détection automatique de vulnérabilités connues

Nous avons vu dans le chapitre 5 qu'il était possible d'identifier à partir du nom et de la version d'un binaire ses vulnérabilités connues. Cependant, certaines vulnérabilités peuvent

dépendre des options de configuration utilisées.

Notre analyse hybride de firmwares pourrait être étendue afin d'intégrer une méthode automatique de détection de vulnérabilités connues pour, par exemple, effectuer de la correction virtuelle par patch (ou *virtual patching* en anglais). En effet, il n'est pas toujours possible de corriger les binaires présents dans un objet IdO ainsi, cette méthode permet d'appliquer des correctifs sans modifier directement l'objet concerné. Dans ce cas d'utilisation, il est nécessaire d'obtenir le firmware de l'objet IdO à analyser. À partir des noms et versions de binaires extraits par notre analyse, les vulnérabilités connues et associées à ces binaires peuvent être obtenues via des bases de données de vulnérabilités publiques (par exemple CVE). Chacune d'elles peut ensuite être interprétée afin d'identifier ses prérequis (par exemple une option de configuration) qui, à leurs tours, sont vérifiés par l'analyse de firmwares. Enfin, suivant les vulnérabilités connues et vérifiées par cette approche, des correctifs virtuels de sécurité peuvent être appliqués.

Application de notre méthode d'identification active

Notre méthode d'identification active est principalement théorique, et chaque propriété à identifier se fait par un classificateur entraîné à cette fin. La principale hypothèse de notre approche est de supposer qu'à partir du firmware d'un objet IdO, il est possible de déduire les services déployés par ce dernier. En d'autres termes, supposer le résultat d'un scan de ports TCP/UDP.

Notre hypothèse est théorique, car nous considérons tout simplement que si un binaire capable de déployer un service est présent dans le firmware, alors ce service est déployé par l'objet. Or, il est possible que des fabricants incluent des binaires inutiles dans leurs firmwares. Une première piste d'amélioration serait d'analyser les scripts d'initialisation présents dans un firmware afin d'identifier les binaires exécutés au cours de l'initialisation de l'objet, et donc les services démarrés. Il serait pertinent ensuite de comparer ces derniers avec ceux déduits par notre approche.

Enfin, nous n'avons pas appliqué notre méthode sur des objets IdO physiques. Ainsi, évaluer notre approche sur des objets IdO physiques permettrait de confirmer l'applicabilité de notre approche dans le cadre de cyberattaques ou pour de la correction virtuelle par patch. Cette dernière application nécessite d'entraîner régulièrement nos classificateurs avec de nouvelles données afin de pouvoir identifier les propriétés associées aux nouveaux objets IdO connectés au réseau informatique surveillé. Pour ce faire, notre analyse hybride de firmwares pourrait être utilisée pour extraire des données de firmwares d'objets IdO, comme le nom et la version des binaires. Une fois les propriétés d'un objet identifiées, il est possible de détecter les vulnérabilités connues puis d'appliquer des correctifs virtuels de sécurité, par exemple des règles de pare-feu.

Utilisation d'objets IdO émulés en pots de miel

Nous avons vu que notre pot de miel à forte interaction était capable de recevoir des attaques d'humains et de botnets. Cependant, déployer ce dernier depuis un serveur appartenant à une institution académique est suspect. C'est pourquoi, des travaux existants [67, 162] utilisent des tunnels SSH entre des serveurs de fournisseurs cloud et les pots de miel. Une piste d'amélioration serait d'héberger notre pot de miel depuis des adresses IP résidentielles, c'est-à-dire des adresses IP appartenant à des fournisseurs d'accès internet (FAI). De cette façon, le pot de miel ressemblera à un objet IdO quelconque branché chez un utilisateur.

Dans [67], les auteurs ont montré qu'une fois leur pot de miel listé sur Shodan, ce dernier recevait trois fois plus d'attaques. Dans notre cas, l'adresse IP publique utilisée pour héberger

notre pot de miel était constamment en ligne. Suivant la fréquence à laquelle Shodan scanne les machines connectées à l'Internet, notre pot de miel pouvait avoir le profil de notre serveur sur Shodan, c'est-à-dire aucun service exploitable comme TELNET ou HTTP. Une expérience serait de rendre le serveur inaccessible une fois le pot de miel hors ligne, et ainsi voir si la fréquence des attaques changent également. Cependant, nos déploiements étant limités à 3 heures par firmware, il n'est pas garanti que Shodan scanne le pot de miel durant cet intervalle.

Une autre piste d'amélioration serait de bloquer les adresses IP d'attaquants ayant déjà effectué des attaques afin de limiter le nombre d'attaques identiques. Pour rappel, notre analyse a montré que seuls 35 (3.8%) des 921 *malwares* téléchargés sur notre pot de miel étaient différents. Ce problème est difficile, car un botnet utilise généralement plusieurs machines afin de détecter une cible potentielle ou effectuer ladite attaque. De plus, bloquer une adresse IP indique clairement aux attaquants qu'une action a été effectuée et que la machine ciblée est protégée. Cependant, cela permettrait de mesurer la motivation des attaquants à attaquer le pot de miel. En effet, protéger activement le pot de miel peut motiver certains attaquants à cibler particulièrement ce dernier, et donc effectuer des attaques complexes et inconnues.

Nous avons restreint l'accès au serveur HTTP(S) de notre pot de miel car des informations erronées pouvaient être visibles depuis ce dernier. Cependant, de nombreux objets IdO ou de l'Internet industriel des Objets (IIdO), comme une caméra connectée ou un thermostat connecté, peuvent afficher des données de capteurs ou du flux vidéo. Ainsi, étendre notre solution afin de pouvoir falsifier ce type de données permettrait d'analyser plus finement les cyberattaques reçues, notamment déterminer si un attaquant recherche des données précises sur le pot de miel ou s'il propage naïvement sa cyberattaque.

Productions

Publications

Les contributions présentées dans cette thèse ont donné lieu à plusieurs publications scientifiques :

- **Passive Inference of User Actions through IoT Gateway Encrypted Traffic Analysis**, publié à DISSECT 2019 [191], décrit notre méthode d'analyse du trafic réseau d'une box domotique présentée dans le chapitre 4.
- **Software-based Analysis of the Security by Design in Embedded Devices**, publié à IM 2021 [189], présente notre analyse hybride de firmwares d'objets IdO détaillée dans le chapitre 5.
- **Inferring Software Composition and Credentials of Embedded Devices from Partial Knowledge**, publié à CNSM 2021 [188], correspond à notre méthode d'analyse active des propriétés d'un objet IdO à l'aide de méthodes d'apprentissage automatique discutée dans le chapitre 6.
- **HiFiPot : a High-Fidelity Emulation Framework for IoT Honeypots**, soumission en cours [190], détaille le pot de miel à forte interaction capable de simuler le fonctionnement d'objets IdO présenté dans le chapitre 7.

Glossaire

AES Advanced Encryption Standard ou algorithme de chiffrement symétrique

CSRF Cross-Site Request Forgery ou injection de code malveillant exécuté de manière non consentie par un utilisateur généralement authentifié

CVE Common Vulnerabilities and Exposures, liste de fiches explicatives des failles de sécurité informatique existantes

DOS Denial of Service attack ou attaque par déni de service (revient à surcharger un service de requêtes)

DNS Domain Name System ou système de noms de domain

DT Decision Tree ou arbre de décision

FTP File Transfer Protocol ou protocole de transfert de fichier

GCM Galois/Counter Mode ou chiffrement par bloc itératif en cryptographie symétrique

GRU Gated Recurrent Unit ou réseaux de neurones récurrents à portes

HTTP HyperText Transfer Protocol ou protocole de transfert hypertextuel

HTTPS HyperText Transfer Protocol Secure ou protocole de transfert hypertextuel utilisant le chiffrement des données

ICMP Internet Control Message Protocol ou protocole de message de contrôle sur Internet

ICS Industrial Control Systems ou systèmes de contrôle industriel

IdO Internet de l'Objet ou Internet of Things (IoT)

IIdO Internet industriel des objets ou Industrial Internet of Things (IIoT)

IDPS Intrusion Detection and Prevention Systems ou système de détection et prévention d'intrusions

IoT Internet of Things ou Internet de l'Objet (IdO)

IIoT Industrial Internet of Things ou Internet industriel des objets (IIdO)

OWASP Open Web Application Security Project ou fondation sur la sécurité des applications web

PLC Programmable Logic Controller ou Contrôleur à Logique Programmable

kNN k-Nearest Neighbors ou méthode des k plus proches voisins

NTP Network Time Protocol ou protocole qui permet de synchroniser l'horloge d'une ou plusieurs machines

RF Random Forest ou forêts d'arbres décisionnels

RNN Recurrent Neural Network ou réseaux de neurones récurrents

SCADA Supervisory Control And Data Acquisition ou système de contrôle et d'acquisition de données en temps réel

SMB Server Message Block : protocole permettant une machine cliente d'utiliser des ressources, comme des fichiers, appartenant à une machine serveur

SSDP Simple Service Discovery Protocol : protocole de découverte d'hôtes dans un réseau

SSH Secure SHell ou protocole de communication sécurisé entre deux machines

TCP Transmission Control Protocol ou protocole de contrôle de transmission

TLS Transport Layer Security ou protocole permettant de chiffrer les données entre deux machines connectés à un réseau informatique

UDP User Datagram Protocol ou protocole de datagramme utilisateur

UPnP Universal Plug and Play ou protocole de découverte permettant à des objets connectés à un réseau informatique d'interagir automatiquement entre eux

XML Extensible Markup Language ou langage de balisage extensible

XMPP Extensible Messaging and Presence Protocol ou protocole extensible de présence et de messagerie

XSS Cross-site scripting ou injection de code malveillant interprétable côté client

Table des figures

Liste des tableaux

Annexes

Annexe A

Analyse de sites internet à l'aide de Scrapy

A.1 Extraction des informations

Scrapy [74] est un outil open source permettant la création de robots d'indexation consistant à parcourir des sites internet pour y extraire du contenu de manière prédéfinie. Dans cette annexe, nous détaillons le fonctionnement de Scrapy par rapport à notre cas d'utilisation qu'est le parcours de sites internet de fabricants d'objets connectés afin de télécharger les firmwares et extraire les informations de chacun des objets IdO disponibles.

À partir de l'infrastructure logicielle de Scrapy illustrée dans la figure A.1, le fonctionnement de cet outil peut être décrit de la manière suivante :

(a) Un composant de type *Spider* noté S est implémenté pour chacun des sites de fabricants d'objets IdO. Dans ce dernier, il est nécessaire de spécifier une ou plusieurs URL $U_1, ..., U_n$ avec $n \geq 1$ à explorer, puis pour chacune de ces URL, Scrapy va préparer la requête à envoyer dans l'étape (1), et la transmettre au *Scheduler* dans l'étape (2).

(b) Afin d'éviter d'être considéré par le serveur destinataire comme effectuant une attaque par déni de service ou *Denial of Service (DoS) attack* en anglais, il est possible de spécifier à Scrapy plusieurs paramètres limitant le nombre de requêtes par seconde, le temps à attendre entre deux requêtes ou associer une priorité à certaines requêtes. C'est une fois après avoir considéré ces paramètres que Scrapy effectue les étapes (3) et (4) et ainsi, initie une connexion avec U_i où $i \in [1, n]$.

(c) Une fois la réponse du serveur destinaire reçue à l'étape (5), par exemple une page au format *HyperText Markup Language (HTML)*, cette dernière est ensuite de nouveau transmise au *Spider* S ayant initié la requête correspondante à l'étape (a).

(d) Dans l'étape (6), S va effectuer plusieurs opérations, propres à chacun des sites de fabricants, sur la réponse du serveur, notée R. Les données dans R étant majoritairement dans un langage de balisage comme HTML ou *Extensible Markup Language (XML)*, alors il est possible d'utiliser le langage de requête XPath pour extraire certains éléments de R. Par exemple, dans une page HTML, si un lien de téléchargement vers un firmware se trouve dans une balise `<firmware>` alors l'instruction XPath '`//firmware`' permet d'extraire ce lien. Suivant le contenu de R, les requêtes XPath implémentées dans S permettent 1) de

74. Disponible sur `https://scrapy.org/`, dernier accès le 13/02/2022

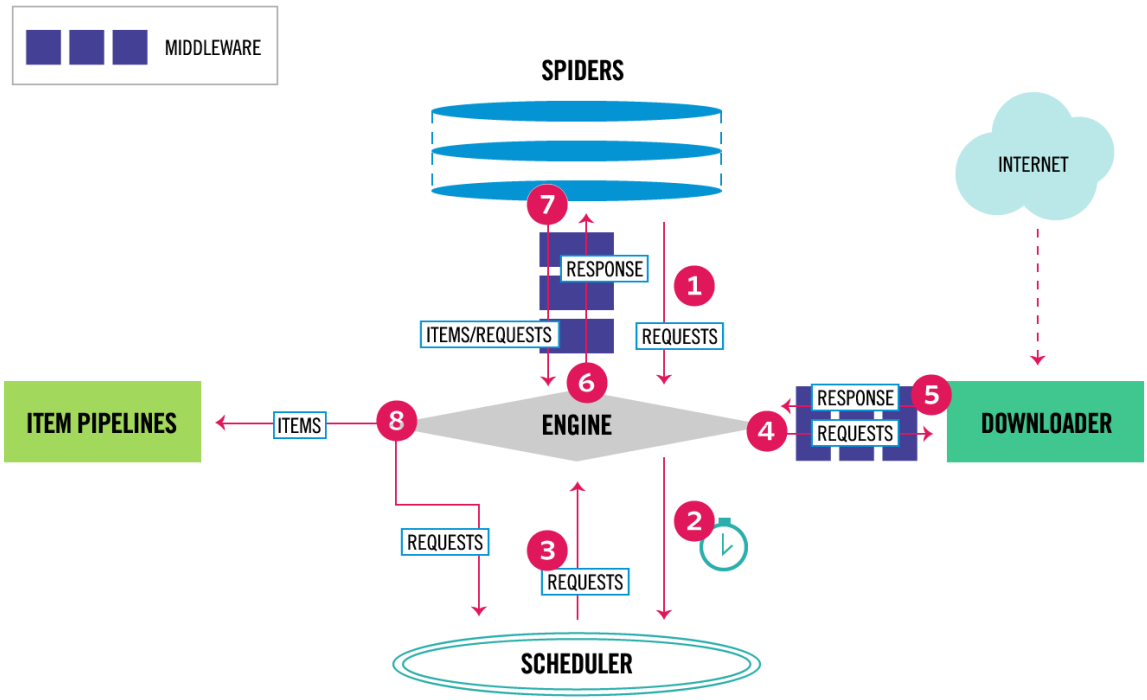


FIGURE A.1 – Schéma des huit interactions entre les composants utilisés par Scrapy pour télécharger des ressources dans l'Internet. Image reprise de la documentation officielle de la version 1.6 de Scapy

dériver l'URL de la prochaine requête à effectuer ou 2) initier le téléchargement d'un firmware et l'extraction des informations liées à ce dernier. En effet, si l'URL initiale à l'étape (1) est une page d'accueil, il est alors nécessaire de trouver à partir de cette dernière les URL associées aux produits. D'un autre côté, si R correspond à une page web d'un objet IdO, les instructions XPath implémentées dans S permettent d'extraire les informations suivantes :

- le modèle de l'objet IdO associé au firmware,
- le fabricant de ce produit,
- l'URL de téléchargement du firmware,
- la version du firmware et
- la date de sortie du firmware.

Enfin, S va transmettre l'ensemble de ces informations à l'*Item Pipeline*.

(e) Finalement, c'est lors de l'étape (8), que l'*Item Pipeline* effectue le traitement final des données qui, dans notre cas, correspond au téléchargement du firmware et au stockage de ses informations associées vers, par exemple, une base de données.

Comme évoqué à l'étape (a), il est nécessaire d'implémenter un *Spider* pour chaque fabricant d'objets IdO car chaque site internet a un plan de site différent ainsi, les instructions XPath à effectuer lors de l'étape (d) doivent être adaptées. Cependant, un seul composant de type *Item*

Pipeline est à implémenter car, peu importe le fabricant, l'objectif est de stocker les firmwares et leurs informations associées dans une base de données.

Annexe B

Catégorisation des firmwares

B.1 Identification du type de l'objet à partir de son firmware

À l'aide de la méthode présentée dans l'annexe A, les firmwares associés à des systèmes embarqués sont téléchargés, et de nombreuses propriétés propres à chacun d'eux, comme leur modèle ou leur fabricant, sont extraites. Cependant, le type auquel appartient chacun des systèmes embarqués n'est pas nécessairement disponible depuis les pages web d'où sont téléchargés les firmwares ainsi, afin d'avoir une vue plus fine sur ces derniers, nous assignons à chacun d'eux une catégorie principale, et une ou plusieurs sous-catégories parmi celles listées dans le tableau B.1.

Catégorie	`appareil réseau`	`objet IdO`
Sous-catégorie	routeur, point d'accès sans fil, commutateur, contrôleur sans file, CPL, VPN, répéteur wifi, hub, modem, pare-feu, serveur de stockage en réseau (NAS)	caméra, capteur, prise connectée, téléphone, enregistreur vidéo (NVR), projecteur, imprimante, CPE

Tableau B.1 – Distribution des sous-catégories disponibles pour les catégories `appareil réseau` et `objet IdO`

Pour ce faire, à partir du nom du fabricant f et du modèle de l'objet m associés à chaque firmware, les trois étapes suivantes sont réalisées :

1. Une recherche Google est effectuée sur m en restreignant les résultats aux sites internet appartenant à f. Par exemple, pour la caméra DCS-2630L de marque D-Link, la requête sera `DCS-2630L site:*dlink.com`. Ensuite, les sous-catégories listées dans le tableau B.1 sont recherchées dans le contenu d'une ou plusieurs balises HTML appartenant aux cinq premières pages web retournées par Google. Par exemple, pour la marque D-Link, nous analysons le contenu des éléments `h1` et `meta`. Enfin, les sous-catégories extraites sont associées au couple $< f, m >$.
2. Même s'il n'est pas garanti de trouver au moins une sous-catégorie pour chacun des couples $< f, m >$, nous remarquons que, pour un f donné, les sous-catégories de certains modèles pourraient être utilisées pour assigner des modèles non catégorisés. Par exemple, si le `commutateur` de marque D-Link `DSN-3400-10` est correctement assigné mais pas le modèle `DSN-5210-10` alors, il est normal d'assigner également à ce dernier la même sous-catégorie,

c'est-à-dire `commutateur`. Pour ce faire, nous calculons pour chacun des modèles d'un même fabricant f, les distances de Levenshtein [75] entre chacun d'eux et, si une de ces valeurs est inférieure à quatre alors nous considérons les deux modèles concernés comme étant similaires. Cette distance, entre deux chaînes de caractères a et b, mesurent le nombre d'opérations nécessaires pour transformer a en b, par exemple $Levenshtein(elo, hello) = 2$ car il faut rajouter un 'h' et un 'l'. Enfin, une fois les couples $< f, m >$ regroupés en $k \geq 1$ ensembles par similarité, nous assignons à chacun des $n \geq 1$ couples $< f, m >$ présents dans un de ces ensembles, les sous-catégories présentes dans la majorité des n couples.

3. Enfin, à partir des associations entre les catégories et sous-catégories présentées dans le tableau B.1, notre approche inspecte les sous-catégories précédemment assignées à un couple $< f, m >$ puis assigne comme catégorie principale celle regroupant la majorité des sous-catégories. Dans le cas où aucune sous-catégorie n'existe alors la catégorie `autre` sera affectée.

Suivant les sous-catégories trouvées, trois catégories principales sont possibles : `objet IdO`, `appareil réseau` et `autre`. Ainsi, chaque système embarqué associé à un firmware est assigné à une seule de ces catégories et à une ou plusieurs sous-catégories. Ce qui permet donc de pouvoir différencier les `objets IdO` des `appareils réseau` dans nos analyses.

75. Voir `https://fr.wikipedia.org/wiki/Distance_de_Levenshtein`, dernier accès le 19/02/2022

Annexe C

Détails techniques de notre pot de miel

C.1 Émulation d'objets IdO

La méthode d'émulation d'objets IdO est composée de plusieurs étapes dont certaines à forte connotation technique. Ces dernières sont détaillées dans les sections suivantes. Nous présentons la méthode utilisée pour intercepter les appels système, puis nous détaillons la construction de l'image QEMU. Enfin, nous décrivons le processus de démarrage d'un objet émulé et la méthode de simulation de la NVRAM, présentés initialement dans FIRMADYNE [36].

C.1.1 Interception des appels système

Similairement à [36, 90, 169], les appels système, ou *syscalls* en anglais, sont interceptés à l'aide de deux outils de débogage depuis un module Linux, c'est-à-dire depuis l'espace noyau. Ces derniers sont déclarés dans notre module et doivent être associés à une fonction présente dans le noyau Linux. Le fonctionnement de chacun de ces outils peut être décrit comme suit :

— `jprobe` : intercepte les appels vers une fonction f ainsi que ses arguments avant que f soit exécutée.

— `kretprobe` : intercepte la valeur retournée par une fonction f.

Il est possible d'assigner une `jprobe` et une `kretprobe` à une même fonction f afin d'étudier les valeurs retournées par une fonction par rapport aux arguments reçus. Par exemple, intercepter la fonction `do_sys_open` avec ces deux outils permet d'identifier les fichiers ne pouvant pas être ouvert, c'est-à-dire les fichiers manquants.

Les appels système interceptés par notre approche sont listés et décrits dans le tableau C.1.

Appel système	Intercepté par une		Nécessaire pour		Motivation
	kprobe	kretprobe	*emulation*	*pot de miel*	
do_sys_open	✓	✓	✓	×	Détecte les fichiers ouverts et manquants
vfs_stat	✓	✓	✓	×	
do_mount	✓	✓	✓	×	Détecte les répertoires manquants
inet_bind	✓	×	✓	×	Identifie le nom, le numéro de port ainsi que le protocole (TCP/UDP) de chaque service déployé par le système
register_vlan_dev	✓	×	✓	×	Détecte les opérations d'affectation d'un VLAN à une interface réseau
register_vlan_device	✓	×	✓	×	
__inet_insert_ifa	✓	×	✓	×	Identifie l'affectation d'adresse IP à une interface réseau
br_add_if	✓	×	✓	×	Identifie les opérations de configuration d'un pont réseau
br_dev_ioctl	✓	×	✓	×	
ip_rt_ioctl	✓	×	✓	×	Détecte les ajouts de route dans la table de routage
fib_magic	✓	×	✓	×	
n_tty_receive_buf	✓	×	×	✓	Intercepte les commandes saisies par un attaquant dans un terminal ou pseudo-terminal
inet_msg_ops	✓	×	×	✓	Identifie l'interface réseau à utiliser pour transmettre les logs et les paquets déchiffrés du pot de miel
sys_write	✓	×	×	✓	Intercepte les opérations d'écritures, exfiltre les données écrites, et cache la présence du pot de miel
sys_close	✓	×	×	✓	Détecte la fin d'une opération d'écriture
sys_exit	✓	×	×	✓	
sys_signal	✓	×	×	✓	
do_execve	✓	×	✓	✓	Ajoute automatiquement la librairie partagée à la variable d'environnement `LD_PRELOAD` de chaque commande
sys_reboot	✓	×	✓	✓	Simule un redémarrage système en exécutant le script d'initialisation `/sbin/init`
do_vfs_ioctl	✓	✓	✓	✓	Identifie les interfaces réseau manquantes, et créée de fausses interfaces réseau sans-fil

Tableau C.1 – Appels système interceptés par notre module

C.1.2 Construction d'une image QEMU

Pour émuler le firmware d'un objet IdO avec QEMU en mode système complet, ou *full system* en anglais, ce dernier requiert, en plus d'un noyau Linux, une image QEMU contenant le système de fichiers racine à émuler. Dans cette section, nous détaillons le processus de construction d'une image QEMU à partir du firmware d'un objet IdO, noté f_w.

Ce processus est illustré dans la figure C.1 et s'explique comme suit :

(1) Le contenu de f_w est extrait à l'aide de `Binwalk`[76] dans un nouveau répertoire noté u_{fw}.

(2) Parmi les systèmes de fichiers présents dans u_{fw}, celui ayant le plus grand nombre de dossiers UNIX [177] est considéré comme étant le système de fichiers racine, noté $root_{fs}$.

(3) La taille de $root_{fs}$ est mesurée et arrondie à la première puissance de deux supérieure. Cette dernière est ensuite transmise à QEMU pour créer une image Q à cette taille et au format *raw*. Le contenu de $root_{fs}$ est ensuite copié dans Q.

76. Disponible sur `https://github.com/ReFirmLabs/binwalk`, dernier accès le 19/04/2022

(4) Similairement à [36, 90, 169], un dossier d contenant les fichiers s'assurant de la bonne émulation du firmware est créé. Ce dossier est nommé `/emulation` dans la figure C.1, et contient 1) une librairie partagée `libnvram.so` simulant le fonctionnement d'une mémoire RAM non volatile (*Non-volatile random-access memory (NVRAM)* en anglais), 2) un script shell `preInit.sh` s'assurant du bon état du système de fichiers racine dans Q (à l'aide d'opérations comme `mkdir` ou `mount`), et 3) le script shell `fix.sh` servant à configurer les interfaces réseau ou exécuter des binaires depuis l'objet émulé.

Contrairement à [169, 90] où un script shell présent dans Q est modifié afin d'exécuter `fix.sh`, notre approche exécute ce dernier à l'aide de notre module Linux. De plus, pour ne pas dépendre des binaires présents dans Q, nous copions dans d le binaire `BusyBox` dans sa version 1.32.0, et compilé avec 401 applets, ou fonctionnalités. Le binaire `BusyBox` est compatible avec l'architecture processeur attendue par l'objet à émuler.

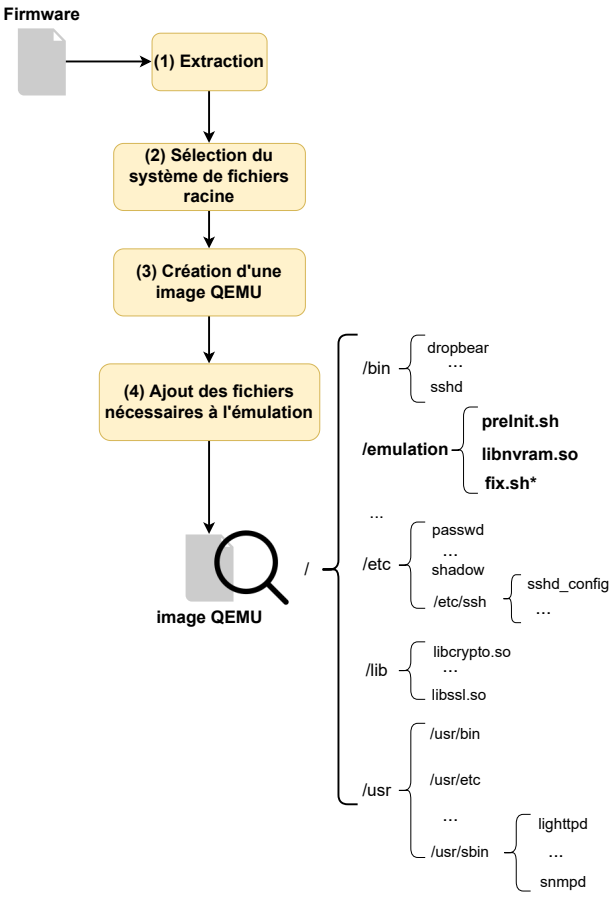


FIGURE C.1 – Construction d'une image QEMU à partir d'un firmware

C.1.3 Émulation d'une image QEMU

Dans cette section, nous détaillons le processus de démarrage d'une image QEMU dans FIRMADYNE [36].

À l'aide d'une image QEMU Q et d'un noyau Linux, l'émulateur QEMU requiert de connaître le script shell, ou binaire, dans Q à faire exécuter en premier par le noyau Linux.

Le script shell `preInit.sh`, présenté dans la section C.1.2, est premièrement exécuté par le noyau Linux à l'aide de l'option de démarrage `rdinit`, spécifiée à l'émulateur QEMU via l'argument `append`. Ce script shell vérifie l'état du système de fichiers en créant des répertoires, comme `/dev` ou `/proc`, et en montant des partitions.

Ensuite, le noyau Linux exécute par défaut un script d'initialisation parmi plusieurs possibilités. Par exemple, un noyau Linux version 2.6.36 exécute d'abord `/init` puis teste les binaires `/sbin/init`, `/etc/init` et `/bin/init`. Après cette étape, les binaires présents dans l'image QEMU sont exécutés suivants les instructions du script d'initialisation, et l'image QEMU est donc émulé. Si aucun script d'initialisation n'est exécuté, l'émulation échoue avec une erreur de type panique du noyau, ou *kernel panic* en anglais.

C.1.4 Simulation de la mémoire RAM non volatile

Dans cette section, nous présentons la méthode de simulation de la mémoire RAM non volatile, ou *Non-volatile random-access memory (NVRAM)*, proposée par FIRMADYNE [36]. Cette simulation est effectuée par une librairie partagée utilisée par l'objet émulé au cours de son exécution afin de récupérer ou stocker des propriétés associées à sa configuration, c'est-à-dire des couples <clé, valeur>. Cependant, ces propriétés ne sont pas nécessaires présentes dans les firmwares d'objets IdO.

Cette librairie partagée implémente un grand nombre de fonctions utilisées dans les objets IdO pour interagir avec la NVRAM. Par exemple, `nvram_get` ou `nvram_set` pour, respectivement, récupérer la valeur d'une clé, et assigner une valeur à une clé. Pour intercepter les fonctions interagissant avec la NVRAM, cette librairie partagée est chargée par tous les processus démarrés par l'objet émulé à l'aide de la variable d'environnement `LD_PRELOAD`. L'assignation de la librairie partagée à cette variable d'environnement est codée en dur dans le noyau Linux.

Afin de retourner des valeurs par défaut à certaines clés, 20 couples <clé, valeur> sont ajoutés dans la librairie partagée. Par exemple, l'adresse IP de l'objet est assignée par la clé `lan_ipaddr` avec la valeur `192.168.0.50`. Cette adresse IP sera celle de l'objet émulé si ce dernier demande à la NVRAM la valeur de la clé `lan_ipaddr`.

Enfin, des chemins vers des fichiers connus, et contenant des couples <clé, valeur> sont également spécifiés dans la librairie partagée. Celle-ci va, à chaque interaction, lire et charger les couples présents dans ces fichiers. Les clés et valeurs utilisées par l'objet durant son émulation sont stockées dans un repértoire monté sur `tmpfs`.

C.2 Implémentation des fonctionnalités du pot de miel

Cette section regroupe l'ensemble des détails techniques associés à chacune des fonctionnalités implémentées par notre pot de miel.

C.2.1 Interception des actions de l'attaquant depuis le pot de miel

Une fois connecté au pot de miel, un attaquant effectue des commandes shell depuis l'un des terminaux, puis essaie de télécharger et exécuter un *malware*. Dans cette section, nous présentons la méthode utilisée pour intercepter ces commandes, puis nous détaillons les étapes implémentées pour exfiltrer les fichiers téléchargés, comme les *malwares*, du pot de miel.

C.2.1.1 Surveillance des activités de l'attaquant

Surveiller les activités effectuées par un attaquant dans un terminal, ou pseudo-terminal, permet d'étudier la séquence d'actions faite par un attaquant pour mener son attaque. Ces actions ont généralement deux objectifs :

— inspecter les composants matériels de l'objet afin d'exploiter une vulnérabilité spécifique, ou vérifier que l'objet attaqué n'est pas un pot de miel ;

— mener l'attaque, c'est-à-dire trouver un programme disponible pour télécharger un *malware* puis l'exécuter.

Les actions des attaquants depuis les terminaux sont interceptées à l'aide de la fonction `n_tty_receive_buf` proposée par le driver `tty`, et dont les arguments sont tous accesibles depuis notre module, c'est-à-dire depuis l'espace noyau. Cette fonction permet d'identifier les touches, ou combinaisons de touches, saisies par un attaquant. De plus, son argument `tty` permet d'identifier le terminal depuis lequel est exécuté une commande, et ainsi détecter des changements de terminaux, ou différents attaquants connectés simultanément sur le pot de miel. Notre approche peut surveiller jusqu'à 256 terminaux.

Dès qu'un attaquant se connecte au pot de miel, notre approche alloue automatiquement une structure de données permettant de suivre ses activités. Cette structure contient :

— le nom du terminal,

— l'identifiant de l'utilisateur,

— le binaire d'où est originaire l'action,

— un tampon de 1024 octets, noté b.

Ces informations sont exfiltrées à l'aide de méthode décrite dans la section C.2.2 dès que b est plein ou qu'une touche comme `enter` ou `CTRL+M` est actionnée.

L'identification des raccourcis claviers, comme `CTRL+M` ou `CTRL+C`, se fait directement à l'aide des deux arguments de la fonction `n_tty_receive_buf` que sont : `cp` et `count` indiquant la valeur de la touche tapée et sa taille en octets. Dans le cas où la taille est supérieure à un octet alors une combinaison de touches (`ALT+F4`) a été saisie. Notre méthode traduit les combinaisons de touches ainsi que les touches *sensibles*, comme `enter` ou `backspace`, dans un format humainement lisible [77].

Inspecter les combinaisons de touches ou touches saisies individuellement au lieu des commandes complètes [169], nous permet de différencier les attaquants humains des robots. En effet, certaines frappes, comme `backspace` ou `suppr`, peuvent être utilisées pour déduire que l'attaquant est un humain.

C.2.1.2 Téléchargement de fichiers

Une fois connecté au pot de miel, il est commun de voir les attaquants utiliser un binaire, comme `wget`, `tftp` ou `curl`, pour télécharger des fichiers, notamment des *malwares*. Cependant, ces binaires peuvent être renommés par les attaquants afin d'augmenter la furtivité de leur attaque [127] ainsi, intercepter les appels de binaires spécifiques n'est pas fiable.

C'est pourquoi nous identifions les téléchargements de virus informatiques à l'aide de propriétés associées à son processus. En effet, nous supposons qu'un tel processus 1) possède au moins une connexion active, c'est-à-dire une connexion vers un serveur externe pour télécharger

77. Semblable à celui de l'enregistreur de frappe, ou *keylogger* en anglais, présenté dans `https://www.arsouyes.org/phrack-trad/phrack59/phrack59_0x0e.txt`, dernier accès le 06/05/2022

le *malware*, et 2) effectue des opérations d'écriture. Si tel est le cas, notre approche suit ces opérations, en exfiltrant au fur et à mesure les données écrites jusqu'à la fin de celles-ci.

Les opérations d'écriture sont interceptées par une `jprobe` liée à l'appel système `sys_write`. Ce dernier permet d'obtenir le descripteur de fichier `fd`, ainsi que les données `buf` à écrire dans `fd`. À chaque appel intercepté, notre module vérifie que le processus `current` ait au moins une connexion active avant d'exfiltrer le contenu de `buf`, c'est-à-dire qu'au moins un des descripteurs de fichiers soit un socket TCP ou UDP. Cette vérification s'effectue par la fonction `sock_from_file` et nous assure qu'il s'agit d'un téléchargement et non d'une opération d'écriture locale effectuée par un des binaires du pot de miel.

Exfiltrer naïvement l'intégralité du contenu écrit sur le pot de miel risque de générer des problèmes de congestion du trafic réseau, et de ralentir le débit des paquets du pot de miel. D'ailleurs, un attaquant peut générer volontairement des opérations d'écriture pour détecter le pot de miel, comme l'attaque `dd` pour Sebek [132]. C'est pourquoi des filtres sont à spécifier pour exfiltrer uniquement certains types de données du pot de miel. Par exemple, les virus informatiques étant généralement au format *Executable and Linkable Format (ELF)*, un filtre correspondant au nombre magique (*magic number* en anglais) du format ELF est spécifié à notre module. Un nombre magique est un ensemble de caractères permettant d'identifier un format de fichier. Dans le cas d'ELF, ce nombre magique est égal à `"\177ELF"`. Seules les opérations d'écriture dont le contenu de `buf` satisfait un des filtres implémentés par notre approche sont suivies.

Si ces deux conditions sont vérifiées, notre méthode exfiltre `fd` et le contenu de `buf` jusqu'à ce qu'un des appels système suivants soit intercepté et associé à `fd` : `sys_close`, `sys_exit` ou `sys_signal`. Ces trois appels permettent de détecter la fin d'une opération d'écriture.

Les données exfiltrées par le pot de miel ne devant pas dépasser les 1024 octets, il est probable que le pot de miel génére plusieurs paquets réseau pour exfiltrer l'intégralité des données associées à un descripteur de fichier `fd`. Si tel est le cas, notre approche est capable à partir de ces paquets réseau, de reconstruire le fichier correspondant à `fd`.

C.2.2 Exfiltration des actions de l'attaquant

Chaque action effectuée ou fichier téléchargé par un attaquant depuis le pot de miel doivent être exfiltrés sous forme de log du pot de miel vers la machine hôte h. Trois méthodes existent pour échanger ces informations entre un objet émulé o et h : 1) l'utilisation d'une mémoire partagée telle que o écrit les logs puis h lit ces derniers périodiquement [120], 2) o transmet via des paquets réseau les logs dans un format connu de h [132], ou 3) o écrit les logs directement dans l'objet émulé, puis h lit ces derniers une fois l'émulation terminée [169]. La première méthode est furtive mais implique de modifier le logiciel de virtualisation (dans notre cas QEMU), et de définir préalablement la fréquence à laquelle la mémoire partagée doit être lue par h. Si cette dernière est trop faible, il est possible de perdre certaines actions effectuées par l'attaquant. La troisième méthode n'est pas furtive et ne garantit pas de récupérer les logs.

Afin de rendre notre approche furtive, et non dépendante du logiciel de virtualisation, notre approche s'inspire de la méthode 2) proposée par Sebek [132]. Cette méthode transmet des paquets réseau sans que ces derniers soient perceptibles par des commandes comme `tcpdump` ou `ifconfig`. En plus du log, chaque paquet contient : le nom de l'utilisateur et du terminal, l'identifiant du processus (PID) ainsi que l'horodatage.

Pour transmettre ces logs, une interface réseau active est nécessaire. Pour ce faire, notre module inspecte périodiquement les interfaces réseau présentes à l'aide de la variable `init_net`, puis sélectionne la première active. Enfin, le paquet est envoyé en broadcast dont l'adresse est

dérivée de l'adresse IP de l'interface sélectionnée.

C.2.3 Camouflage de la librairie partagée

La librairie partagée `libnvram.so` implémente des fonctionnalités essentielles, comme la simulation de la NVRAM ou la restriction des accès au serveur HTTP(S), et est chargée par chaque processus exécuté par l'objet émulé à l'aide de la variable d'environnement `LD_PRELOAD`. Par conséquent, il est nécessaire de cacher aux attaquants cette variable dont le contenu est visible depuis 1) l'espace utilisateur, par exemple `echo $LD_PRELOAD` à partir d'un terminal, et 2) l'espace noyau via certains fichiers spéciaux, comme `/proc/self/environ` ou `/proc/self/maps`. Ainsi, les mesures prisent pour camoufler le contenu de `LD_PRELOAD` dans l'espace noyau sont détaillées dans la section C.2.3.1, alors que celles spécifiques à l'espace utilisateur sont présentées dans la section C.2.3.2

En réalité, notre librairie partagée `libnvram.so` a un nom généré aléatoirement, transmis à notre module. De même, le nom de la librairie partagée est inclus dans son code lors de sa compilation. Le nom de la librairie est donc disponible dans les espaces noyau et utilisateur.

C.2.3.1 Espace noyau

Nous modifions le noyau Linux afin de rendre la structure `proc_environ_operations` accessible depuis notre module. Cette structure gère les interactions vers les fichiers spéciaux `/proc/{self,pid}/environ`, permettant de visualiser les variables d'environnement 1) du processus courant `self`, ou 2) d'un processus ayant comme identifiant, ou *Process IDentifier (PID)* en anglais, `pid`.

Notre module ensuite redirige l'adresse de la fonction `read`, initialement présente dans cette structure, vers une fonction implémentée dans notre module modifiant les données comme suit : 1) la variable `LD_PRELOAD` est entièrement retirée si uniquement notre librairie partagée est chargée, sinon 2) seule `libnvram.so` est retirée de la liste des librairies partagées présentes dans `LD_PRELOAD`.

Les librairies partagées chargées par le processeur courant `self`, ou un processus ayant comme identifiant `pid`, sont également visibles depuis l'adressage mémoire du processus, c'est-à-dire depuis les fichiers `/proc/{self,pid}/maps`. La structure `proc_environ_operations` est rendue accessible comme détaillé précédemment, et l'adresse de sa fonction `show` est modifiée vers une fonction dans notre module. Cette dernière retire également notre librairie partagée des données à afficher, et s'assure également que les adresses mémoires affichées restent contiguës.

C.2.3.2 Espace utilisateur

En assignant `libnvram.so` à la variable d'environnement `LD_PRELOAD`, cette dernière est ajoutée à la liste des variables d'environnement stockée dans la variable `**environ` dont le contenu est accessible depuis n'importe quel programme exécuté depuis l'espace utilisateur.

Afin de modifier le contenu de cette variable le plus tôt possible, et empêcher l'attaquant de voir notre librairie partagée, nous définissons dans `libnvram.so`, une fonction de type *constructeur* directement exécutée dès le chargement en mémoire de la librairie partagée. C'est depuis cette dernière que la variable d'environnement `LD_PRELOAD` est altérée de la manière suivante : 1) `LD_PRELOAD` est entièrement retirée si uniquement notre librairie partagée est chargée, sinon 2) seule `libnvram.so` est retirée de la liste des librairies partagées présentes dans `LD_PRELOAD`.

C.2.4 Camouflage de répertoires

Lors de la construction de l'image QEMU Q, un répertoire spécifique d est créé afin de stocker les fichiers nécessaires au bon déroulement de l'émulation, et du déploiement du pot de miel. Un attaquant peut donc détecter le pot de miel si d est visible, ainsi ce répertoire doit être caché. Par conséquent, des commandes shell, comme `ls` ou `dir`, ne doivent pas intégrer d dans leur résultat.

Notre module modifie ainsi le comportement de la fonction `vfs_readdir`, et plus précisément son argument `filler`. Ce dernier est en réalité une fonction utilisée pour itérer sur les répertoires présents.

De plus, le nom de d est généré aléatoirement, c'est pourquoi il est nécessaire de le transmettre à notre module via une instruction spécifique afin 1) que notre fonction `filler` retire d de la liste des répertoires visibles, et 2) éviter que les attaquants puissent connaître à l'avance la valeur de d. En effet, notre approche retire d de la liste des répertoires visibles cependant, il est possible de se déplacer vers d à l'aide de la commande `cd`. C'est pourquoi, en assignant une valeur aléatoire à d, un attaquant ne pourra pas directement vérifier la présence de ce répertoire.

C.2.5 Camouflages des fonctions implémentées

Les fonctions implémentées dans le noyau Linux, et par extension notre module, sont listées dans le fichier `/proc/kallsyms`.

Afin de retirer certaines fonctions, nous rendons accessible la structure `kallsyms_op` en enlevant le mot-clé `static`. Depuis notre module, nous modifions l'adresse de la fonction `show` présente dans cette structure vers notre propre fonction capable de filtrer, et ne pas afficher, les fonctions présentes dans une liste L.

L'ajout d'une fonction à L s'effectue par une instruction à transmettre à notre module.

C.2.6 Restriction de l'accès au serveur HTTP

Expérimentalement, nous avons remarqué que le serveur HTTP(S) d'un objet émulé peut contenir des informations laissant penser que l'objet est émulé. Par exemple, des adresses MAC affectées à `00:00:00:00:00`, ou un schéma de l'état du réseau affichant un problème de connexion à l'Internet peuvent être visibles sur l'interface graphique du serveur HTTP(S). C'est pourquoi, notre pot de miel restreint l'accès des attaquants à ce dernier.

Nous considérons le cas où l'accès au serveur HTTP(S) est conditionné à une page d'identification. Ainsi, pour restreindre l'accès aux autres pages, il suffit de bloquer le processus d'identification. En d'autres termes, les noms d'utilisateurs et mots de passe insérés par un attaquant sont discrètement modifiés en temps réel afin d'être systématiquement erronés.

En raison du fait que certains serveurs HTTP peuvent utiliser le chiffrement (HTTPS), modifier les données requiert préalablement de les déchiffrer. C'est pourquoi notre approche diffère de [169] où le trafic réseau associé au serveur HTTPS est uniquement déchiffré une fois l'émulation terminée. Notre pot de miel implémente tout de même ces deux méthodes de déchiffrement : 1) en temps réel, et 2) en différé.

Le déchiffrement en temps réel ne peut pas être effectué depuis notre module, mais uniquement depuis l'espace utilisateur. En effet, une librairie partagée, comme `libssl.so`, implémente les fonctions pour chiffrer et déchiffrer des données. Ce sont ces fonctions que notre pot de miel doit être capable d'intercepter, et plus précisément les appels vers la fonction `SSL_read` chargée du déchiffrement des données.

Pour ce faire, nous utilisons notre librairie partagée `libnvram.so` dans laquelle nous implémentons une nouvelle fonction `SSL_read` 1) déchiffrant les données à l'aide de la vraie fonction `SSL_read`, et 2) modifiant ces dernières si besoin. Les données doivent être modifiées si et seulement si elles sont destinées à un serveur HTTPS. Pour détecter cela, notre fonction inspecte les descripteurs de fichiers ouverts par le processus courant, et si l'un d'eux est associé à un socket dont le numéro de port est 443 (HTTPS), alors les noms d'utilisateurs connus et trouvés dans les données déchiffrées seront altérés. La liste de ces 15 identifiants connus (par défaut) est prédéfinie dans `libnvram.so`, et contient, entre autres, `root`, `admin`, `ubnt` etc.

Cependant, deux prérequis sont nécessaires au fonctionnement de cette approche :

— La librairie partagée `libdl.so` doit être présente dans l'image QEMU Q de l'objet émulé. Cette dernière implémente la fonction `dlsym` qui permet à notre fonction d'appeler la vraie fonction `SSL_read`.

— La fonction `SSL_read` doit être implémentée dans une librairie partagée présente dans Q, comme `libssl.so`, sous la forme d'un symbole global qui permet à cette dernière d'être utilisée par d'autres programmes.

Dans le cas d'un serveur HTTP, les fonctions `recv`, `recvfrom` et `recvmsg` sont interceptées par notre librairie partagée, et les données sont modifiées de la même manière que décrite ci-dessus. À la différence que les descripteurs de fichiers du processus courant doivent avoir un socket ouvert sur le port 80 (HTTP).

Ces prérequis sont vérifiés lors de la compilation de la librairie partagée.

C.2.7 Falsification des données des fichiers de configuration

Notre module peut modifier le contenu des fichiers spéciaux dans `/proc`. Par défaut, notre approche modifie les fichiers `/proc/cpuinfo`, `/proc/cmdline` et `/proc/modules`. Les nouvelles données à assigner sont transmises à notre module via des instructions spécifiques. Suivant le nom du fichier à modifier dans `/proc`, notre module applique une des trois méthodes suivantes :

— le fichier est `/proc/cmdline` : notre module assigne à la variable `saved_command_line`, les nouvelles données.

— le fichier est `/proc/modules` : le contenu de ce fichier n'est pas entièrement altéré. En effet, au cours de l'émulation, l'objet peut insérer plusieurs modules. Au lieu de fournir une liste prédéfinie de modules à afficher, notre approche détecte les modules insérés par l'objet émulé et transmet chacun d'eux à notre module afin de mettre à jour le contenu du fichier `/proc/modules`. Pour ce faire, notre module redirige préalablement la fonction `read` de la structure `proc_modules_operations` vers une fonction personnalisée capable d'afficher la liste des modules demandés par l'objet émulé.

— le fichier f est localisé dans `/proc` : les fichiers dans `/proc` étant sous forme d'arborescence, avec la racine stockée dans la variable `proc_root`, notre module parcourt cette arborescence de la racine jusqu'à atteindre f. Si f n'existe pas, notre module le crée avec la fonction `proc_create_data` en assignant les données souhaitées. Sinon, nous modifions les données déjà affectées.

Les contenus de `/proc/cpuinfo` et `/proc/cmdline` sont propres à chaque marque d'objet IdO afin de maintenir une furtivité optimale.

C.2.8 Création d'interfaces réseau sans-fil

Un attaquant peut inspecter les composants matériels de l'objet émulé dont les interfaces réseau. Cependant, un objet IdO sans interface réseau sans-fil, comme `wlan0`, est suspect, et un attaquant pourrait déduire qu'il s'est connecté à un objet émulé, ou un pot de miel. Contrairement aux interfaces Ethernet, des interfaces réseau sans-fil ne peuvent être allouées par QEMU à l'objet émulé. Ainsi, notre module doit être capable de créer ces dernières.

Notre module intercepte les appels vers la fonction `do_vfs_ioctl` et détecte l'erreur `ENODEV` associée à un composant matériel m manquant seulement s'il s'agit d'un composant réseau. Enfin, m est créé, et de fausses statistiques, comme le nombre d'octets reçus et envoyés, sont assignées à cette nouvelle interface. Pour la création de ces fausses interfaces réseau, nous nous sommes inspirés du driver *dummy*[78]. La création d'une fausse interface réseau se fait uniquement durant le processus de démarrage de l'objet émulé afin d'éviter à un attaquant de créer une fausse interface réseau et détecter le pot de miel.

78. Disponible sur `https://elixir.bootlin.com/linux/v2.6.36/source/drivers/net/dummy.c`, dernier accès le 25/04/2022

Bibliographie

[1] Dionaea – catching bugs. Disponible sur `https://www.honeynet.org/projects/active/dionaea/`. Dernier accès le 16/12/2021.

[2] Kippo. Disponible sur `https://github.com/desaster/kippo`. Dernier accès le 16/12/2021.

[3] Owasp zed attack proxy (zap). Disponible à `https://owasp.org/www-project-zap/`. Dernier accès le 03/12/2021.

[4] w3af : Web application attack and audit framework. Disponible à `http://w3af.org/`. Dernier accès le 03/12/2021.

[5] The heartbleed bug. Disponible sur `https://heartbleed.com/`, 2014. Dernier accès le 01/12/2021.

[6] Cowrie ssh and telnet honeypot. Disponible sur `https://www.cowrie.org/`, 2015. Dernier accès le 16/12/2021.

[7] Mohamed Abomhara and Geir M. Køien. Cyber security and the internet of things : Vulnerabilities, threats, intruders and attacks. *J. Cyber Secur. Mobil.*, 4 :65–88, 2015.

[8] Abbas Acar, Hossein Fereidooni, Tigist Abera, Amit Kumar Sikder, Markus Miettinen, Hidayet Aksu, Mauro Conti, Ahmad-Reza Sadeghi, and Selcuk Uluagac. Peek-a-boo : I see your smart home activities, even encrypted ! In *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '20, page 207–218, New York, NY, USA, 2020. Association for Computing Machinery.

[9] Ahmet Aksoy and Mehmet Hadi Gunes. Automated iot device identification using network traffic. In *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, pages 1–7, 2019.

[10] Mohammed Ali Al-Garadi, Amr Mohamed, Abdulla Khalid Al-Ali, Xiaojiang Du, Ihsan Ali, and Mohsen Guizani. A survey of machine and deep learning methods for internet of things (iot) security. *IEEE Communications Surveys Tutorials*, 22(3) :1646–1685, 2020.

[11] Tanweer Alam. A reliable communication framework and its use in internet of things (iot). *EngRN : Communication System (Topic)*, 2018.

[12] Husain Aljazzar and Stefan Leue. Kstar : A heuristic search algorithm for finding the k shortest paths. *Artificial Intelligence*, 175(18) :2129–2154, 2011.

[13] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. Understanding the mirai botnet. In *26th USENIX Security Symposium*, 2017.

[14] Noah J. Apthorpe, Dillon Reisman, and Nick Feamster. A smart home is no castle : Privacy vulnerabilities of encrypted iot traffic. *CoRR*, abs/1705.06805, 2017.

[15] Noah J. Apthorpe, Dillon Reisman, Srikanth Sundaresan, Arvind Narayanan, and Nick Feamster. Spying on the smart home : Privacy attacks and defenses on encrypted iot traffic. *CoRR*, abs/1708.05044, 2017.

[16] Ofir Arkin and Fyodor Yarochkin. Xprobe v2.0 : A "fuzzy" approach to remote active operating system fingerprinting. Technical report, august 2002. Disponible sur `http://www.ouah.org/Xprobe2.pdf`, dernier accès le 07/05/2022.

[17] Sharad Gore Asmaa Shaker Ashoor. Importance of intrusion detection system (ids). *International Journal of Scientific Engineering Research*, 2 :1–4, 2011.

[18] AspenCore. 2019 embedded markets study. Technical report, AspenCore, 03 2019. Disponible sur `https://www.embedded.com/wp-content/uploads/2019/11/EETimes_Embedded_2019_Embedded_Markets_Study.pdf`, dernier accès le 15/02/2022.

[19] Paul Baecher, Markus Koetter, Thorsten Holz, Maximillian Dornseif, and Felix Freiling. The nepenthes platform : An efficient approach to collect malware. pages 165–184, 09 2006.

[20] Roberto Baldoni, Emilio Coppa, Daniele Cono D'elia, Camil Demetrescu, and Irene Finocchi. A survey of symbolic execution techniques. *ACM Comput. Surv.*, 51(3), may 2018.

[21] Davide Balzarotti, Marco Cova, Vika Felmetsger, Nenad Jovanovic, Engin Kirda, Christopher Kruegel, and Giovanni Vigna. Saner : Composing static and dynamic analysis to validate sanitization in web applications. In *2008 IEEE Symposium on Security and Privacy (sp 2008)*, pages 387–401, 2008.

[22] Jason Bau, Elie Bursztein, Divij Gupta, and John Mitchell. State of the art : Automated black-box web application vulnerability testing. In *2010 IEEE Symposium on Security and Privacy*, pages 332–345, 2010.

[23] Reto Baumann and Christian Plattner. Honeypots. Master's thesis, Swiss Federal Institute of Technology Zurich, 02 2002.

[24] Fabrice Bellard. Qemu, a fast and portable dynamic translator. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '05, page 41, USA, 2005. USENIX Association.

[25] Stephan Berger, Olga Bürger, and Maximilian Röglinger. Attacks on the industrial internet of things – development of a multi-layer taxonomy. *Computers & Security*, 93 :101790, 2020.

[26] Elisa Bertino and Nayeem Islam. Botnets and internet of things security. *Computer*, 50(2) :76–79, 2017.

[27] Bitdefender. Cracking the victure ipc360 monitor - remote control and cloud misconfiguration. Disponible sur `https://www.bitdefender.com/files/News/CaseStudies/study/402/Bitdefender-PR-Whitepaper-VictureIPC-creat5590-en-EN.pdf`. Dernier accès le 02/03/2022.

[28] Sofia Björnehaag. Test of a home energy management system at e.on - an evaluation of users' expectations and experience. Master's thesis, Université de Lund, 2012.

[29] Hristo Bojinov, Elie Bursztein, Eric Lovett, and Dan Boneh. Embedded management interfaces : Emerging massive insecurity. Disponible à `http://seclab.stanford.edu/websec/embedded/bh09-paper.pdf`, 2009. Dernier accès le 03/12/2021.

[30] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. COLT '92, page 144–152, New York, NY, USA, 1992. Association for Computing Machinery.

[31] Hamid Bostani and Mansour Sheikhan. Hybrid of anomaly-based and specification-based ids for internet of things using unsupervised opf based on mapreduce approach. *Computer Communications*, 98 :52–71, 2017.

[32] Robert S. Boyer, Bernard Elspas, and Karl N. Levitt. Select—a formal system for testing and debugging programs by symbolic execution. *SIGPLAN Not.*, 10(6) :234–245, apr 1975.

[33] Sergey Bratus, Cory Cornelius, David Kotz, and Daniel Peebles. Active behavioral fingerprinting of wireless devices. In *Proceedings of the First ACM Conference on Wireless Network Security*, WiSec '08, page 56–61, New York, NY, USA, 2008. Association for Computing Machinery.

[34] Pierre-Olivier Brissaud. *Analyse de trafic HTTPS pour la supervision d'activités utilisateurs*. Theses, Université de Lorraine, December 2020.

[35] Cristian Cadar, Daniel Dunbar, and Dawson Engler. Klee : Unassisted and automatic generation of high-coverage tests for complex systems programs. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, OSDI'08, page 209–224, USA, 2008. USENIX Association.

[36] Daming D. Chen, Maverick Woo, David Brumley, and Manuel Egele. Towards automated dynamic analysis for linux-based embedded firmware. In *23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*. The Internet Society, 2016.

[37] Kai Cheng, Qiang Li, Lei Wang, Qian Chen, Yaowen Zheng, Limin Sun, and Zhenkai Liang. Dtaint : Detecting the taint-style vulnerability in embedded device firmware. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 430–441, 2018.

[38] Bill Cheswick. An evening with berferd in which a cracker is lured, endured, and studied. In *In Proc. Winter USENIX Conference*, pages 163–174, 1992.

[39] Cybersecurity & Infrastructure Security Agency (CISA). Alert (ta14-017a) - udp-based amplification attacks. Technical report, United States Department of Homeland Security (DHS), 12 2019. Disponible sur `https://www.cisa.gov/uscert/ncas/alerts/TA14-017A`, dernier accès le 15/02/2022.

[40] Fred Cohen. A note on the role of deception in information protection. In *Computers & Security*, pages 483–506. Elsevier Science, 01 1998.

[41] Bogdan Copos, Karl Levitt, Matt Bishop, and Jeff Rowe. Is anybody home ? inferring activity from smart home network traffic. In *2016 IEEE Security and Privacy Workshops (SPW)*, pages 245–251, 2016.

[42] Jose Antonio Coret. Kojoney - a honeypot for the ssh service. Disponible sur `http://kojoney.sourceforge.net/`, 2005. Dernier accès le 16/12/2021.

[43] Symantec Corporation. Internet security threat report (istr). Technical report, 02 2019. Disponible à `https://docs.broadcom.com/doc/istr-24-2019-en`, dernier accès le 02/12/2021.

[44] Andrei Costin, Jonas Zaddach, Aurélien Francillon, and Davide Balzarotti. A large-scale analysis of the security of embedded firmwares. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 95–110, San Diego, CA, August 2014. USENIX Association.

[45] Andrei Costin, Apostolis Zarras, and Aurélien Francillon. Automated dynamic firmware analysis at scale : A case study on embedded web interfaces. In *Proceedings of the 11th*

ACM on Asia Conference on Computer and Communications Security, ASIA CCS '16, page 437–448, New York, NY, USA, 2016. Association for Computing Machinery.

[46] Marco Cova, Viktoria Felmetsger, Greg Banks, and Giovanni Vigna. Static detection of vulnerabilities in x86 executables. In *2006 22nd Annual Computer Security Applications Conference (ACSAC'06)*, pages 269–278, 2006.

[47] Ang Cui. Embedded device firmware vulnerability hunting using frak. BlackHat USA, 2012. Disponible sur `https://infocondb.org/con/black-hat/black-hat-usa-2012/embedded-device-firmware-vulnerability-hunting-using-frak`, dernier accès le 07/05/2022.

[48] Ang Cui, Michael Costello, and Salvatore J. Stolfo. When firmware modifications attack : A case study of embedded exploitation. In *20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24-27, 2013*. The Internet Society, 2013.

[49] Ang Cui and Salvatore J. Stolfo. A quantitative analysis of the insecurity of embedded network devices : Results of a wide-area scan. In *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, page 97–106, New York, NY, USA, 2010. Association for Computing Machinery.

[50] Johannes Dahse and Thorsten Holz. Simulation of built-in php features for precise static code analysis. In *NDSS*, 2014.

[51] Fan Dang, Zhenhua Li, Yunhao Liu, Ennan Zhai, Qi Alfred Chen, Tianyin Xu, Yan Chen, and Jingyu Yang. Understanding fileless attacks on linux-based iot devices with honeycloud. In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '19, page 482–493, New York, NY, USA, 2019. Association for Computing Machinery.

[52] Aveek K. Das, Parth H. Pathak, Chen-Nee Chuah, and Prasant Mohapatra. Uncovering privacy leakage in ble network traffic of wearable fitness trackers. In *Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications*, HotMobile '16, page 99–104, New York, NY, USA, 2016. Association for Computing Machinery.

[53] Drew Davidson, Benjamin Moench, Thomas Ristenpart, and Somesh Jha. FIE on firmware : Finding vulnerabilities in embedded systems using symbolic execution. In *22nd USENIX Security Symposium (USENIX Security 13)*, pages 463–478, Washington, D.C., August 2013. USENIX Association.

[54] Nitesh Dhanjani. Hacking lightbulbs : Security evaluation of the philips hue personal wireless lighting system. `https://img2.helpnetsecurity.com/dl/articles/Hacking_Lightbulbs.pdf`, 2013. Dernier accès le 18/11/2021.

[55] T. Dierks and E. Rescorla. The transport layer security (tls) protocol version 1.1. RFC 4346, RFC Editor, April 2006. `http://www.rfc-editor.org/rfc/rfc4346.txt`.

[56] T. Dierks and E. Rescorla. The transport layer security (tls) protocol version 1.2. RFC 5246, RFC Editor, August 2008. `http://www.rfc-editor.org/rfc/rfc5246.txt`.

[57] Brendan Dolan-Gavitt, Josh Hodosh, Patrick Hulin, Tim Leek, and Ryan Whelan. Repeatable reverse engineering with panda. In *Proceedings of the 5th Program Protection and Reverse Engineering Workshop*, PPREW-5, New York, NY, USA, 2015. Association for Computing Machinery.

[58] Lautaro Dolberg, Quentin Jérôme, Jérôme François, Radu State, and Thomas Engel. Ramses : Revealing android malware through string extraction and selection. In Jing

Tian, Jiwu Jing, and Mudhakar Srivatsa, editors, *International Conference on Security and Privacy in Communication Networks*, pages 498–506, Cham, 2015. Springer International Publishing.

[59] Zakir Durumeric, David Adrian, Ariana Mirian, Michael Bailey, and J. Alex Halderman. A search engine backed by internet-wide scanning. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, page 542–553, New York, NY, USA, 2015. Association for Computing Machinery.

[60] Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. Zmap : Fast internet-wide scanning and its security applications. In *22nd USENIX Security Symposium (USENIX Security 13)*, pages 605–620, Washington, D.C., August 2013. USENIX Association.

[61] Peter Eckersley. How unique is your web browser ? In Mikhail J. Atallah and Nicholas J. Hopper, editors, *Privacy Enhancing Technologies*, pages 1–18, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[62] Viktoria Felmetsger, Ludovico Cavedon, Christopher Kruegel, and Giovanni Vigna. Toward automated detection of logic vulnerabilities in web applications. In *Proceedings of the 19th USENIX Conference on Security*, USENIX Security'10, page 10, USA, 2010. USENIX Association.

[63] Xuan Feng, Qiang Li, Haining Wang, and Limin Sun. Acquisitional rule-based engine for discovering internet-of-thing devices. In *Proceedings of the 27th USENIX Conference on Security Symposium*, SEC'18, page 327–341, USA, 2018. USENIX Association.

[64] Anna Fensel, Vikash Kumar, and Slobodanka Dana Kathrin Tomic. End-user interfaces for energy-efficient semantically enabled smart homes. *Energy Efficiency*, 7, 08 2014.

[65] John Graham-Cumming. Understanding and mitigating ntp-based ddos attacks. Disponible sur `https://blog.cloudflare.com/understanding-and-mitigating-ntp-based-ddos-attacks/`, 09 2014. Dernier accès le 15/02/2022.

[66] Margherita Grandini, Enrico Bagli, and Giorgio Visani. Metrics for multi-class classification : an overview, 2020.

[67] Juan David Guarnizo, Amit Tambe, Suman Sankar Bhunia, Martin Ochoa, Nils Ole Tippenhauer, Asaf Shabtai, and Yuval Elovici. Siphon : Towards scalable high-interaction physical honeypots. In *Proceedings of the 3rd ACM Workshop on Cyber-Physical System Security*, CPSS '17, page 57–68, New York, NY, USA, 2017. Association for Computing Machinery.

[68] Hang Guo and John Heidemann. Ip-based iot device detection. In *Proceedings of the 2018 Workshop on IoT Security and Privacy*, IoT S&P '18, page 36–42, New York, NY, USA, 2018. Association for Computing Machinery.

[69] Muhammad A. Hakim, Hidayet Aksu, A. Selcuk Uluagac, and Kemal Akkaya. U-pot : A honeypot framework for upnp-based iot devices. In *37th IEEE International Performance Computing and Communications Conference, IPCCC 2018, Orlando, FL, USA, November 17-19, 2018*, pages 1–8. IEEE, 2018.

[70] Craig Heffner. Littleblackbox : Database of private ssl/ssh keys for embedded devices. Disponible sur `https://code.google.com/p/littleblackbox/`. Dernier accès le 30/11/2021.

[71] Craig Heffner. Exploiting surveillance cameras - like a hollywood hacker. BlackHat USA, 02 2013. Disponible sur `https://paper.bobylive.com/Meeting_Papers/BlackHat/USA-2013/US-13-Heffner-Exploiting-Network-Surveillance-Cameras-Like-A-Hollywood-Hacker-WP.pdf`, dernier accès le 07/03/2022.

[72] Armijn Hemel. Introducing the binary analysis tool (bat). Disponible sur `https://github.com/armijnhemel/binaryanalysis`, 05 2013. Dernier accès le 07/05/2022.

[73] Andrew Henderson, Aravind Prakash, Lok Kwong Yan, Xunchao Hu, Xujiewen Wang, Rundong Zhou, and Heng Yin. Make it work, make it right, make it fast : Building a platform-neutral whole-system dynamic binary analysis platform. ISSTA 2014, page 248–258, New York, NY, USA, 2014. Association for Computing Machinery.

[74] Stephen Herwig, Katura Harvey, George Hughey, Richard Roberts, and Dave Levin. Measurement and analysis of hajime, a peer-to-peer iot botnet. *Proceedings 2019 Network and Distributed System Security Symposium*, 2019.

[75] Chen Hongsong, Fu Zhongchuan, and Zhang Dongyan. Security and trust research in m2m system. In *Proceedings of 2011 IEEE International Conference on Vehicular Electronics and Safety*, pages 286–290, 2011.

[76] Danny Yuxing Huang, Noah Apthorpe, Frank Li, Gunes Acar, and Nick Feamster. Iot inspector : Crowdsourcing labeled network traffic from smart home devices at scale. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 4(2), jun 2020.

[77] Yao-Wen Huang, Fang Yu, Christian Hang, Chung-Hung Tsai, Der-Tsai Lee, and Sy-Yen Kuo. Securing web application code by static analysis and runtime protection. In *Proceedings of the 13th International Conference on World Wide Web*, WWW '04, page 40–52, New York, NY, USA, 2004. Association for Computing Machinery.

[78] Richard Hummel, Carol Hildebrand, Hardik Modi, Chris Conrad, Roland Dobbins, Steinthor Bjarnson, Jon Belanger, Gary Sockrider, Philippe Alcoy, and Tom Bienkowski. Netscout threat intelligence report - ddos in a time of pandemic. Technical report, Netscout, 2020. Disponible sur `https://www.netscout.com/sites/default/files/2021-04/ThreatReport_2H2020_FINAL_0.pdf`, dernier accès le 15/02/2022.

[79] Edgar Iglesias. Status of 'cris' architecture support in linux kernel. Disponible sur `https://lkml.org/lkml/2014/9/15/1082`. Dernier accès le 17/02/2022.

[80] IOActive. Disponible sur `https://ioactive.com/pdfs/IOActive_DASDEC_vulnerabilities.pdf`, jun 2013. Dernier accès le 30/11/2021.

[81] IOActive. Disponible sur `https://ioactive.com/article/ioactive-lights-up-vulnerabilities-for-over-half-a-million-belkin-wemo-users/`, feb 2014. Dernier accès le 30/11/2021.

[82] Andreas Jacobsson, Martin Boldt, and Bengt Carlsson. A risk analysis of a smart home automation system. *Future Generation Computer Systems*, 56 :719–733, 2016.

[83] Julian Jang-Jaccard and Surya Nepal. A survey of emerging threats in cybersecurity. *Journal of Computer and System Sciences*, 80(5) :973–993, 2014. Special Issue on Dependable and Secure Computing.

[84] N. Jovanovic, C. Kruegel, and E. Kirda. Pixy : a static analysis tool for detecting web application vulnerabilities. In *2006 IEEE Symposium on Security and Privacy (S P'06)*, pages 6 pp.–263, 2006.

[85] Markus Kammerstetter, Christian Platzer, and Wolfgang Kastner. Prospect : Peripheral proxying supported embedded code testing. In *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '14, page 329–340, New York, NY, USA, 2014. Association for Computing Machinery.

[86] Poul-Henning Kamp and Robert N. M. Watson. Jails : Confining the omnipotent root. Disponible sur https://docs.freebsd.org/44doc/papers/jail/jail.html. Dernier accès le 16/12/2021.

[87] Richard Killam, Paul Cook, and Natalia Stakhanova. Android malware classification through analysis of string literals. In *Workshop Programme*, page 27, 2016.

[88] Hwankuk Kim, Taeun Kim, and Daeil Jang. An intelligent improvement of internet-wide scan engine for fast discovery of vulnerable iot devices. *Symmetry*, 10(5), 2018.

[89] Juhwan Kim, Jihyeon Yu, Hyunwook Kim, Fayozbek Rustamov, and Joobeom Yun. Firmcov : High-coverage greybox fuzzing for iot firmware via optimized process emulation. *IEEE Access*, 9 :101627–101642, 2021.

[90] Mingeun Kim, Dongkwan Kim, Eunsoo Kim, Suryeon Kim, Yeongjin Jang, and Yongdae Kim. Firmae : Towards large-scale emulation of iot firmware for dynamic analysis. In *Annual Computer Security Applications Conference*, ACSAC '20, page 733–745, New York, NY, USA, 2020. Association for Computing Machinery.

[91] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. Ddos in the iot : Mirai and other botnets. *Computer*, 50(7) :80–84, 2017.

[92] Karl Koscher, Tadayoshi Kohno, and David Molnar. SURROGATES : Enabling near-real-time dynamic analyses of embedded systems. In *9th USENIX Workshop on Offensive Technologies (WOOT 15)*, Washington, D.C., August 2015. USENIX Association.

[93] McAfee Labs. Threats report. Technical report, McAfee, 03 2018.

[94] Pierre Laperdrix, Nataliia Bielova, Benoit Baudry, and Gildas Avoine. Browser fingerprinting : A survey, 2019.

[95] Pavel Laskov, Patrick Düssel, Christin Schäfer, and Konrad Rieck. Learning intrusion detection : Supervised or unsupervised ? In Fabio Roli and Sergio Vitulano, editors, *Image Analysis and Processing – ICIAP 2005*, pages 50–57, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[96] Jinkyung Lee, Chaetae Im, and Hyuncheol Jeong. A study of malware detection and classification by comparing extracted strings. In *Proceedings of the 5th International Conference on Ubiquitous Information Management and Communication*, ICUIMC '11, New York, NY, USA, 2011. Association for Computing Machinery.

[97] Jun Li, Bodong Zhao, and Chao Zhang. Fuzzing : a survey. *Cybersecurity*, 1(1) :6, Jun 2018.

[98] Martin Libicki. *Cyberspace in Peace and War*. Transforming war. Naval Institute Press, 2021.

[99] Bryson Lingenfelter, Iman Vakilinia, and Shamik Sengupta. Analyzing variation among iot botnets using medium interaction honeypots. In *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 0761–0767, 2020.

[100] Rapid7 LLC. Disponible à https://www.metasploit.com/. Dernier accès le 03/12/2021.

[101] Tie Luo and Sai G. Nagarajan. Distributed anomaly detection using autoencoder neural networks in wsn for iot. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–6, 2018.

[102] Tongbo Luo, Zhaoyan Xu, Xing Jin, Yanhui Jia, and Xin Ouyang. Iotcandyjar : Towards an intelligent-interaction honeypot for iot devices. BlackHat USA, 2017. Disponible sur https://www.blackhat.com/docs/us-17/thursday/us-17-Luo-Iotcandyjar-Towards

-An-Intelligent-Interaction-Honeypot-For-IoT-Devices-wp.pdf, dernier accès le 07/05/2022.

[103] Ali Maetouq, Salwani Mohd Daud, Noor Azurati Ahmad, Nurazean Maarop, Nilam Nur Amir Sjarif, and Hafiza Abas. Comparison of hash function algorithms against attacks : A review. *International Journal of Advanced Computer Science and Applications*, 9, 2018.

[104] Alejandro Guerra Manzanares. Honeyio4 : the construction of a virtual, low-interaction iot honeypot. Master's thesis, Universitat Politècnica de Catalunya, 2017. Disponible sur https://upcommons.upc.edu/bitstream/handle/2117/108166/Alejandro_Guerra_Manzanares.pdf?isAllowed=y&sequence=1, dernier accès le 14/12/2021.

[105] Valentin J.M. Manès, HyungSeok Han, Choongwoo Han, Sang Kil Cha, Manuel Egele, Edward J. Schwartz, and Maverick Woo. The art, science, and engineering of fuzzing : A survey. *IEEE Transactions on Software Engineering*, 47(11) :2312–2331, 2021.

[106] Samuel Marchal, Markus Miettinen, Thien Duc Nguyen, Ahmad-Reza Sadeghi, and N. Asokan. Audi : Toward autonomous iot device-type identification using periodic communication. *IEEE Journal on Selected Areas in Communications*, 37(6) :1402–1412, 2019.

[107] Artur Marzano, David Alexander, Osvaldo Fonseca, Elverton Fazzion, Cristine Hoepers, Klaus Steding-Jessen, Marcelo H. P. C. Chaves, Ítalo Cunha, Dorgival Guedes, and Wagner Meira. The evolution of bashlite and mirai iot botnets. In *2018 IEEE Symposium on Computers and Communications (ISCC)*, pages 00813–00818, 2018.

[108] John McHugh, Alan Christie, and Julia Allen. Defending yourself : The role of intrusion detection systems. *IEEE Software*, 17(5) :42–51, 2000.

[109] Yair Meidan, Michael Bohadana, Yael Mathov, Yisroel Mirsky, Asaf Shabtai, Dominik Breitenbacher, and Yuval Elovici. N-baiot—network-based detection of iot botnet attacks using deep autoencoders. *IEEE Pervasive Computing*, 17(3) :12–22, 2018.

[110] Yair Meidan, Michael Bohadana, Asaf Shabtai, Juan David Guarnizo, Martín Ochoa, Nils Ole Tippenhauer, and Yuval Elovici. Profiliot : A machine learning approach for iot device identification based on network traffic analysis. In *Proceedings of the Symposium on Applied Computing*, SAC '17, page 506–509, New York, NY, USA, 2017. Association for Computing Machinery.

[111] Yair Meidan, Michael Bohadana, Asaf Shabtai, Martin Ochoa, Nils Ole Tippenhauer, Juan Davis Guarnizo, and Yuval Elovici. Detection of unauthorized iot devices using machine learning techniques, 2017.

[112] Markus Miettinen, Samuel Marchal, Ibbad Hafeez, N. Asokan, Ahmad-Reza Sadeghi, and Sasu Tarkoma. Iot sentinel : Automated device-type identification for security enforcement in iot. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 2177–2184, 2017.

[113] Barton Miller, Lars Fredriksen, and Bryan So. An empirical study of the reliability of unix utilities. *Commun. ACM*, 33 :32–44, 12 1990.

[114] H. D. Moore. Security flaws in universal plug and play - unplug. don't play. Technical report, Rapid7 LLC, 01 2013.

[115] Keaton Mowery and Hovav Shacham. Pixel perfect : Fingerprinting canvas in HTML5. In Matt Fredrikson, editor, *Proceedings of W2SP 2012*. IEEE Computer Society, May 2012.

[116] Marius Muench, Dario Nisi, Aurélien Francillon, and Davide Balzarotti. Avatar^2 : A multi-target orchestration platform. In ISOC, editor, *BAR 2018, Workshop on Binary Analysis Research, colocated with NDSS Symposium, 18 February 2018, San Diego, USA*, San Diego, 2018.

[117] Marius Muench, Jan Stijohann, Frank Kargl, Aurélien Francillon, and Davide Balzarotti. What you corrupt is not what you crash : Challenges in fuzzing embedded devices. In ISOC, editor, *NDSS 2018, Network and Distributed Systems Security Symposium, 18-21 February 2018, San Diego, CA, USA*, San Diego, 2018. © ISOC. Personal use of this material is permitted. The definitive version of this paper was published in NDSS 2018, Network and Distributed Systems Security Symposium, 18-21 February 2018, San Diego, CA, USA and is available at : http ://dx.doi.org/10.14722/NDSS.2018.23166.

[118] Antonio Nappa, Richard B. Johnson, Leyla Bilge, Juan Caballero, and Tudor Dumitras. The attack of the clones : A study of the impact of shared code on vulnerability patching. *2015 IEEE Symposium on Security and Privacy*, pages 692–708, 2015.

[119] Thien Duc Nguyen, Samuel Marchal, Markus Miettinen, Minh Hoang Dang, N. Asokan, and Ahmad-Reza Sadeghi. Dïot : A crowdsourced self-learning approach for detecting compromised iot devices. *CoRR*, abs/1804.07474, 2018.

[120] Vincent Nicomette, Mohamed Kaâniche, Eric Alata, and Matthieu Herrb. Set-up and deployment of a high-interaction honeypot : experiment and lessons learned. *Journal in Computer Virology*, 7(2) :143–157, May 2011.

[121] Nmap. Nmap : the network mapper. `https://nmap.org/`. Dernier accès le 24/11/2021.

[122] Mehdi Nobakht, Vijay Sivaraman, and Roksana Boreli. A host-based intrusion detection and mitigation framework for smart home iot using openflow. In *2016 11th International Conference on Availability, Reliability and Security (ARES)*, pages 147–156, 2016.

[123] National Technology Security Coaliation (NTSC). Cyber security report. Technical report, 2020. Disponible à `https://www.ntsc.org/assets/pdfs/cyber-security-report-2020.pdf`, dernier accès le 18/11/2021.

[124] TJ OConnor, Reham Mohamed, Markus Miettinen, William Enck, Bradley Reaves, and Ahmad-Reza Sadeghi. Homesnitch : Behavior transparency and control for smart home iot devices. In *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '19, page 128–138, New York, NY, USA, 2019. Association for Computing Machinery.

[125] Marten Oltrogge, Yasemin Acar, Sergej Dechand, Matthew Smith, and Sascha Fahl. To pin or not to pin helping app developers bullet proof their tls connections. In *Proceedings of the 24th USENIX Conference on Security Symposium*, SEC'15, page 239–254, USA, 2015. USENIX Association.

[126] Openwall. John the ripper password cracker. Disponible sur `https://www.openwall.com/john/`. Dernier accès le 30/11/2021.

[127] Yin Pa, Shogo Suzuki, Katsunari Yoshioka, Tsutomu Matsumoto, Takahiro Kasama, and Christian Rossow. Iotpot : A novel honeypot for revealing current iot threats. *Journal of Information Processing*, 24 :522–533, 05 2016.

[128] Adrian Pauna and Ion Bica. Rassh - reinforced adaptive ssh honeypot. In *2014 10th International Conference on Communications (COMM)*, pages 1–6, 2014.

[129] Adrian Pauna, Andrei-Constantin Iacob, and Ion Bica. Qrassh - a self-adaptive ssh honeypot driven by q-learning. In *2018 International Conference on Communications (COMM)*, pages 441–446, 2018.

[130] Venkat Pothamsetty and Matthew Franz. Scada honeynet project : Building honeypots for industrial networks. Technical report, Cisco Systems, Inc., 2004.

[131] Fabien Pouget, Marc Dacier, and Hervé Debar. White paper : honeypot, honeynet, honeytoken : terminological issues. *Rapport technique EURECOM*, 1275, 09 2003.

[132] The Honeynet Project. Know your enemy : Sebek - a kernel based data capture tool. Disponible sur `https://www.cs.jhu.edu/~rubin/courses/sp04/sebek.pdf`. Dernier accès le 30/12/2021.

[133] Niels Provos. Honeyd : A virtual honeypot daemon (extended abstract). 2003. Code open source disponible sur `http://www.honeyd.org/`, dernier accès le 13/12/2021.

[134] Sakthi Vignesh Radhakrishnan, A. Selcuk Uluagac, and Raheem Beyah. Gtid : A technique for physical device and device type fingerprinting. *IEEE Transactions on Dependable and Secure Computing*, 12(5) :519–532, 2015.

[135] Shahid Raza, Linus Wallgren, and Thiemo Voigt. Svelte : Real-time intrusion detection in the internet of things. *Ad Hoc Networks*, 11(8) :2661–2674, 2013.

[136] Nilo Redini, Aravind Machiry, Ruoyu Wang, Chad Spensky, Andrea Continella, Yan Shoshitaishvili, Christopher Kruegel, and Giovanni Vigna. Karonte : Detecting insecure multibinary interactions in embedded firmware. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1544–1561, 2020.

[137] E. Rescorla. The transport layer security (tls) protocol version 1.3. RFC 8446, RFC Editor, August 2018.

[138] Lukas Rist, Johnny Vestergaard, Daniel Haslinger, Andrea Pasquale, and John Smith. Conpot : Ics/scada honeypot. Disponible sur `http://conpot.org/`, 2014. Dernier accès le 14/12/2021.

[139] Bardia Safaei, Amir Mahdi Hosseini Monazzah, Milad Barzegar Bafroei, and Alireza Ejlali. Reliability side-effects in internet of things application layer protocols. In *2017 2nd International Conference on System Reliability and Safety (ICSRS)*, pages 207–212, 2017.

[140] Tara Salman and Raj Jain. A survey of protocols and standards for internet of things. *ArXiv*, abs/1903.11549, 2019.

[141] Ruben Santamarta. Here be backdoors : A journey into the secrets of industrial firmware. BlackHat USA, 2012. Disponible sur `https://paper.bobylive.com/Meeting_Papers/BlackHat/USA-2012/BH_US_12_Santamarta_Backdoors_WP.pdf`, dernier accès le 01/12/2021.

[142] Karen A. Scarfone and Peter M. Mell. Sp 800-94. guide to intrusion detection and prevention systems (idps). Technical report, Gaithersburg, MD, USA, 2007.

[143] Charles Scott. Designing and implementing a honeypot for a scada network. Technical report, SANS, 06 2014. Disponible sur `https://www.sans.org/white-papers/35252/`, dernier accès le 15/12/2021.

[144] Zain Shamsi, Daren B. H. Cline, and Dmitri Loguinov. Faulds : A non-parametric iterative classifier for internet-wide os fingerprinting. *IEEE/ACM Transactions on Networking*, 29(5) :2339–2352, 2021.

[145] Zain Shamsi, Ankur Nandwani, Derek Leonard, and Dmitri Loguinov. Hershel : Single-packet os fingerprinting. In *The 2014 ACM International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '14, page 195–206, New York, NY, USA, 2014. Association for Computing Machinery.

[146] Wazen M. Shbair, Thibault Cholez, Jerome Francois, and Isabelle Chrisment. A multi-level framework to identify https services. In *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, pages 240–248, 2016.

[147] Itamar Sher, Dean Sysman, and Imri Goldberg. Mtpot. Disponible sur `https://github.com/Cymmetria/MTPot`, 2016. Dernier accès le 15/12/2021.

[148] Yan Shoshitaishvili, Ruoyu Wang, Christophe Hauser, Christopher Kruegel, and Giovanni Vigna. Firmalice - automatic detection of authentication bypass vulnerabilities in binary firmware. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*. The Internet Society, 2015.

[149] Yan Shoshitaishvili, Ruoyu Wang, Christopher Salls, Nick Stephens, Mario Polino, Andrew Dutcher, John Grosen, Siji Feng, Christophe Hauser, Christopher Kruegel, and Giovanni Vigna. Sok : (state of) the art of war : Offensive techniques in binary analysis. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 138–157, 2016.

[150] Rajesh Kumar Shrivastava, Bazila Bashir, and Chittaranjan Hota. Attack detection and forensics using honeypot in iot environment. In Günter Fahrnberger, Sapna Gopinathan, and Laxmi Parida, editors, *Distributed Computing and Internet Technology*, pages 402–409, Cham, 2019. Springer International Publishing.

[151] Michael S. Sink. The use of honeypots and packet sniffers for intrusion detection. Disponible sur `https://www.giac.org/paper/gsec/670/honeypots-packet-sniffers-intrusion-detection/101548`, 04 2001. Dernier accès le 10/12/2021.

[152] Arunan Sivanathan, Hassan Habibi Gharakheili, and Vijay Sivaraman. Inferring iot device types from network behavior using unsupervised clustering. In *2019 IEEE 44th Conference on Local Computer Networks (LCN)*, pages 230–233, 2019.

[153] Arunan Sivanathan, Hassan Habibi Gharakheili, and Vijay Sivaraman. Detecting behavioral change of iot devices using clustering-based network traffic modeling. *IEEE Internet of Things Journal*, 7(8) :7295–7309, 2020.

[154] Arunan Sivanathan, Hassan Habibi Gharakheili, and Vijay Sivaraman. Managing iot cybersecurity using programmable telemetry and machine learning. *IEEE Transactions on Network and Service Management*, 17(1) :60–74, 2020.

[155] Arunan Sivanathan, Daniel Sherratt, Hassan Habibi Gharakheili, Adam Radford, Chamith Wijenayake, Arun Vishwanath, and Vijay Sivaraman. Characterizing and classifying iot traffic in smart cities and campuses. In *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 559–564, 2017.

[156] Joseph Slowik. Evolution of ics attacks and the prospects for future disruptive events. Technical report, Dragos Inc, 2019. Disponible sur `https://www.dragos.com/resource/evolution-of-ics-attacks-and-the-prospects-for-future-disruptive-events/`, dernier accès le 07/05/2022.

[157] Lance Spitzner. *Honeypots : Tracking Hackers*. Addison-Wesley Longman Publishing Co., Inc., USA, 2002.

[158] Prashast Srivastava, Hui Peng, Jiahao Li, Hamed Okhravi, Howard Shrobe, and Mathias Payer. Firmfuzz : Automated iot firmware introspection and analysis. In *Proceedings of the 2nd International ACM Workshop on Security and Privacy for the Internet-of-Things*, IoT S&P'19, page 15–21, New York, NY, USA, 2019. Association for Computing Machinery.

[159] Lukas A. Stafira. Examining effectiveness of web-based internet of things honeypots. Master's thesis, Air Force Institute of Technology, Wright-Patterson Air Force Base, Ohio, 2019. Disponible sur `https://scholar.afit.edu/etd/2284/`, dernier accès le 13/12/2021.

[160] George Stergiopoulos, Dimitris Gritzalis, and Evangelos Limnaios. Cyber-attacks on the oil & gas sector : A survey on incident assessment and attack patterns. *IEEE Access*, 8 :128440–128475, 2020.

[161] Clifford Stoll. *The Cuckoo's Egg : Tracking a Spy through the Maze of Computer Espionage*. Pocket Books, 2000.

[162] Amit Tambe, Yan Lin Aung, Ragav Sridharan, Martín Ochoa, Nils Ole Tippenhauer, Asaf Shabtai, and Yuval Elovici. *Detection of Threats to IoT Devices Using Scalable VPN-Forwarded Honeypots*, page 85–96. Association for Computing Machinery, New York, NY, USA, 2019.

[163] Cyber Research Team. Risk : Is this your webcam ? you're being watched. Disponible sur `https://www.wizcase.com/blog/webcam-security-research/`, 2019. Dernier accès le 02/03/2022.

[164] Vrizlynn L. L. Thing. Ieee 802.11 network anomaly detection and attack classification : A deep learning approach. In *2017 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6, 2017.

[165] Rahmadi Trimananda, Janus Varmarken, Athina Markopoulou, and Brian Demsky. Ping-pong : Packet-level signatures for smart home device events, 2020.

[166] Konstantinos Tsiknas, Dimitrios Taketzis, Konstantinos Demertzis, and Charalabos Skianis. Cyber threats to industrial iot : A survey on attacks and countermeasures. *IoT*, 2(1) :163–186, 2021.

[167] Emmanouil Vasilomanolakis, Shankar Karuppayah, Mathias Fischer, Max Mühlhäuser, Mihai Plasoianu, Lars Pandikow, and Wulf Pfeiffer. This network is infected : Hostage - a low-interaction honeypot for mobile devices. In *Proceedings of the Third ACM Workshop on Security and Privacy in Smartphones & Mobile Devices*, SPSM '13, page 43–48, New York, NY, USA, 2013. Association for Computing Machinery.

[168] Emmanouil Vasilomanolakis, Shreyas Srinivasa, Carlos Garcia Cordero, and Max Mühlhäuser. Multi-stage attack detection and signature generation with ics honeypots. In *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, pages 1227–1232, 2016.

[169] Alexander Vetterl and Richard Clayton. Honware : A virtual honeypot framework for capturing cpe and iot zero days. In *2019 APWG Symposium on Electronic Crime Research (eCrime)*, pages 1–13, 2019.

[170] Gérard Wagener, Radu State, Alexandre Dulaunoy, and Thomas Engel. Heliza : Talking dirty to the attackers. *Journal in Computer Virology*, 7 :221–232, 08 2011.

[171] He Wang, Ted Tsung-Te Lai, and Romit Roy Choudhury. Mole : Motion leaks through smartwatch sensors. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, MobiCom '15, page 155–166, New York, NY, USA, 2015. Association for Computing Machinery.

[172] Meng Wang, Javier Santillan, and Fernando Kuipers. Thingpot : an interactive internet-of-things honeypot. *CoRR*, abs/1807.04114, 2018.

[173] Peter Weidenbach and Johannes vom Dorp. Home router security report 2020. Technical report, Fraunhofer Institute for Communication, Information Processing and Ergonomics, jun 2020. Disponible sur `https://www.fkie.fraunhofer.de/content/dam/fkie/de/documents/HomeRouter/HomeRouterSecurity_2020_Bericht.pdf`, dernier accès le 07/05/2022.

[174] JANOG NTP Information Exchange WG. Ntp reflection ddos attack - explanatory document. Technical report, JANOG, 2015. Disponible sur https://www.janog.gr.jp/wg/doc/ntp-wg-en.pdf, dernier accès le 23/03/2022.

[175] Wikipedia. Executable and Linkable Format — Wikipedia, the free encyclopedia. Disponible sur http://en.wikipedia.org/w/index.php?title=Executable%20and%20Linkable%20Format&oldid=1071645138, 2022. Dernier accès le 16/02/2022.

[176] Kyle Wilhoit. Who's really attacking your ics equipment ? Technical report, Trend Micro Incorporated, 2013.

[177] LSB Workgroup. Filesystem hierarchy standard. Technical report, The Linux Foundation, 03 2015. Disponible sur https://refspecs.linuxfoundation.org/FHS_3.0/fhs-3.0.pdf, dernier accès le 14/02/2022.

[178] J. W. Wright. The change-making problem. *J. ACM*, 22(1) :125–128, January 1975.

[179] Chris Wulff. Altera niosii support. Disponible sur https://lists.gnu.org/archive/html/qemu-devel/2012-09/msg01229.html. Dernier accès le 17/02/2022.

[180] Chao Xu, Parth H. Pathak, and Prasant Mohapatra. Finger-writing with smartwatch : A case for finger and hand gesture recognition using smartwatch. In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, HotMobile '15, page 9–14, New York, NY, USA, 2015. Association for Computing Machinery.

[181] Fabian Yamaguchi, Alwin Maier, Hugo Gascon, and Konrad Rieck. Automatic inference of search patterns for taint-style vulnerabilities. In *2015 IEEE Symposium on Security and Privacy*, pages 797–812, 2015.

[182] Kai Yang, Qiang Li, and Limin Sun. Towards automatic fingerprinting of iot devices in the cyberspace. *Computer Networks*, 148 :318–327, 2019.

[183] Mahmood Yousefi-Azar, Vijay Varadharajan, Len Hamey, and Uday Tupakula. Autoencoder-based feature learning for cyber security applications. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 3854–3861, 2017.

[184] Jonas Zaddach, Luca Bruno, Aurélien Francillon, and Davide Balzarotti. Avatar : A framework to support dynamic security analysis of embedded systems' firmwares. In *NDSS*, 2014.

[185] Yaowen Zheng, Ali Davanian, Heng Yin, Chengyu Song, Hongsong Zhu, and Limin Sun. Firm-afl : High-throughput greybox fuzzing of iot firmware via augmented process emulation. In *Proceedings of the 28th USENIX Conference on Security Symposium*, SEC'19, page 1099–1114, USA, 2019. USENIX Association.

[186] Ye Zhou. Chameleon : Towards adaptive honeypot for internet of things. In *Proceedings of the ACM Turing Celebration Conference - China*, ACM TURC '19, New York, NY, USA, 2019. Association for Computing Machinery.

[187] Haris Šemić and Sasa Mrdovic. Iot honeypot : A multi-component solution for handling manual and mirai-based attacks. In *2017 25th Telecommunication Forum (TELFOR)*, pages 1–4, 2017.

[188] Pierre-Marie Junges, Jérome François, and Olivier Festor. Inferring software composition and credentials of embedded devices from partial knowledge. In *2021 17th International Conference on Network and Service Management (CNSM)*, pages 437–445, 2021.

[189] Pierre-Marie Junges, Jérome François, and Olivier Festor. Software-based analysis of the security by design in embedded devices. In *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 558–564, 2021.

[190] Pierre-Marie Junges, Jérome François, and Olivier Festor. Hifipot : a high-fidelity emulation framework for iot honeypots. Non publié, 2022.

[191] Pierre-Marie Junges, Jérôme François, and Olivier Festor. Passive inference of user actions through iot gateway encrypted traffic analysis. In *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 7–12, 2019.

Résumé

L'usage des objets de l'Internet des Objets (IdO) dans nos environnements personnels ou professionnels facilite nos interactions du quotidien, mais ces derniers souffrent souvent de problèmes de sécurité. L'enjeu de cette thèse est d'évaluer la sécurité des objets IdO à l'échelle de l'Internet selon plusieurs axes. Pour ce faire, les travaux proposés doivent satisfaire plusieurs contraintes comme le passage à l'échelle, la gestion de l'hétérogénéité des objets IdO ou, dans le cas d'une analyse de trafic réseau, l'impossibilité d'intercepter les communications sans-fil d'objets IdO.

Nous nous sommes d'abord intéressés aux risques de fuite de données utilisateur introduits par les box domotiques, dont le rôle est d'agir comme un relai entre un utilisateur et des objets IdO. De plus, la box agrège le trafic réseau des objets ce qui empêche l'utilisation des méthodes existantes pour identifier les actions utilisateur effectuées. Nous proposons une méthode capable d'inférer les actions utilisateur en décomposant la taille des données dans les requêtes reçues par la box en une ou plusieurs combinaisons d'actions utilisateur possibles. Notre méthode a ensuite été appliquée sur une box domotique connectée à 16 objets IdO, et nous avons montré qu'un attaquant pouvait inférer les actions utilisateur dans plus de 91.2% des cas.

Dans une deuxième contribution, nous évaluons le niveau de sécurité d'objets IdO à l'aide d'une analyse hybride de firmwares combinant une analyse statique et dynamique. Contrairement aux solutions existantes, notre objectif n'est pas de détecter de nouvelles vulnérabilités mais plutôt d'analyser la composition des binaires présents dans les firmwares, notamment par rapport à la présence de vulnérabilités connues ou à l'utilisation de versions de binaires obsolètes. Nous avons ensuite analysé 4 730 firmwares d'objets IdO publiés entre 2009 et 2019, et noté une utilisation prépondérante de certains binaires, mais également une vague de patches en 2017.

L'analyse de firmwares permet d'obtenir 1) des informations précises sur les binaires (nom et version), 2) les noms et mots de passe des utilisateurs. À l'aide de ces informations, nous avons élaboré une méthode d'identification active permettant à un attaquant d'inférer, à partir des résultats supposés d'un scan de ports, des propriétés précises sur un objet IdO connecté, comme le nom du binaire utilisé pour déployer le serveur HTTP ou les noms d'utilisateurs. Notre méthode consiste à entraîner des classificateurs à partir des données extraites des firmwares. Elle est capable de prédire correctement ($>$73.14%) le nom ou la version d'un binaire déployant HTTP, SSH ou DNS. La prédiction des mots de passe nécessite, elle, moins de deux tentatives. L'utilisation de notre approche semble plus efficace, et furtive, qu'une approche par *brute-force*.

Notre quatrième contribution s'intéresse aux cyberattaques ciblant les objets IdO. Pour ce faire, nous avons défini un pot de miel à forte interaction basé sur une méthode d'émulation utilisant des firmwares d'objets IdO. Le déploiement d'un pot de miel est difficile à cause de la contrainte de furtivité que les méthodes existantes d'émulation d'objets IdO n'intègrent pas. Notre contribution est capable d'émuler un objet IdO en 1) maximisant sa furtivité, et 2) en intégrant des fonctionnalités inhérentes à un pot de miel (par exemple exfiltrer les activités des attaquants). Notre approche peut émuler jusqu'à 825 (82.5%) objets IdO contre 454 (45.4%) avec la méthode de l'état de l'art. Enfin, nous avons déployé notre pot de miel durant environ un an, et montré que ce dernier recevait des attaques inconnues ou récentes de botnets, et d'humains.

Mots-clés: Internet des Objets (IdO), Analyse de trafic chiffré, Détection d'actions utilisateur, Méthode d'identification active, Pot de miel, Émulation d'objets IdO

Abstract

Nowadays, the use of Internet of Things (IoT) devices in our personal and work space makes our everyday life easier, but those IoT devices often suffer from security issues. The objective of this thesis is to evaluate the security of IoT devices. On one hand, we investigate the risk of user privacy leakage introduced by IoT hubs (or IoT gateways). Those IoT hubs act as a middlebox between a user and the IoT devices. Existing passive fingerprinting techniques are not applicable in this configuration considering that the network traffic of each individual IoT device attached to the IoT hub is not accessible. We propose a passive fingerprinting technique to infer the user actions by analysing the network traffic of the IoT gateway. Our method works on encrypted network traffic, and consists of decomposing a packet payload size into a set of, potential, user actions. We applied our technique on one IoT gateway controlling up to 16 IoT devices and show that an attacker, located on the Internet, is able to infer the user actions in more than 91.2% of the investigated cases.

In a further step, we propose a hybrid firmware analysis technique to evaluate the security of an IoT device by inspecting the content of its firmware. Our analysis combines a dynamic analysis and a static analysis to improve our chances to extract data. Our objectives are not to detect unknown vulnerabilities but only the known ones, and inspect if the binaries included are deprecated. We applied our analysis on 4,730 firmwares belonging to IoT devices released between 2009 and 2019, and noticed the widespread use of a small set of binaries, notably to deploy HTTP and SSH services. From 2017, we observed that IoT manufacturers implemented many updates which reduced the exposure to known vulnerabilities.

Using those firmwares, we defined an active fingerprinting technique allowing an attacker to infer details about a connected IoT devices, such as its brand or the binary used to deploy the HTTP server. Thanks to the firmware content, we can 1) obtain precise information about the binaries (name, version), and 2) assume the services actually deployed by the device *i.e.,* the results of a TCP/UDP port scans. Considering those two aspects, our method consists of training classifiers to predict one particular property of a connected IoT device from, among others, the supposed results of a TCP/UDP port scans. Our method allows to predict fine details such as the name or version of a binary, the usernames or the passwords present in an IoT device. Using our approach, we noticed that the predictions of the name and version of the HTTP, SSH and DNS binaries are achieved with a precision superior to 73.14%. On the other hand, the prediction of at least one valid password is more challenging and requires up to two tries. Our method is more effective and furtive than a naive brute-force method.

Knowing the vulnerabilities present in a IoT device does not guarantee that attackers use them on a regular basis. Hence, we propose in our fourth contribution, a high interaction honeypot capable of intercepting cyberattacks targeting IoT devices. The defined honeypot is based on an existing emulation technique that uses IoT devices firmwares. Implementing an honeypot is hard, and because of the stealth constraint, the existing emulation technique could not be used as-is. Due to this constraint, we implemented a framework capable of emulating IoT devices while assuring their furtivity, and adding honeypot-specific capabilities, such as exfiltrating the attackers activities. We then compared our approach to the state of the art one, and showed that ours can emulate up to 825 (82.5%) devices compared to 454 (45.4%). Our honeypot was deployed on one server during about one year and captured unknown and recent attacks from botnets, and sometimes humans.

Keywords: Internet of Things (IoT), Encrypted network traffic analysis, Detecting user actions, Active fingerprinting technique, Honeypot, Emulation of IoT devices