9 JUIN 2022

RAPPORT PÔLE I.O.T

SÉCURITÉ ET DÉFENSE
ÉCOUTE ET HACKING

Machine Learning appliqué à la détection et à la reconnaissance d'objets connectés ZigBee

PRÉSENTÉ PAR

Abdennacer Badaoui
Louis Fouché
Emma Guetta
Reda Mouqed
Rémy Taha

CentraleSupélec

CentraleSupélec | université PARIS-SACLAY

SÉCURITÉ ET DÉFENSE, ÉCOUTE ET HACKING

RECONNAISSANCE ET DÉTECTION D'OBJETS CONNECTÉS

Sous la direction de
JOCELYN FIORINA

Avec l'équipe de
L'EXOCUBE

Rapporteurs
Abdennacer Badaoui
Louis Fouché
Emma Guetta
Reda Mouqed
Rémy Taha

CENTRALESUPELEC

JUIN 2022

Contents

1 Introduction

On ne peut imaginer un avenir sans objet connecté. Les objets connectés, aussi appelés IoT[1], ont été conçus comme une réponse aux nombreux défis d'aujourd'hui. Optimisation énergétique, défi sécuritaire mais aussi progrès technique, économique et ergonomique, le paysage des enjeux de l'adoption généralisée des IoTs est vaste et divers, et possiblement sans frontière.

Néanmoins, forger l'avenir[2] ne se fait jamais sans risque. En octobre 2016, les IoTs se sont être révélés être la cible d'une attaque massive, la première jamais organisée sur ceux-ci. Le logiciel malveillant Mirai visait à prendre possession de nombreux objets connectés pour organiser des attaques massives par déni de service. Les objets ainsi contaminés ont pu être impliqués dans la cyberattaque de l'hébergeur français OVH et d'autres services informatiques (DNS). Même si de nombreux efforts ont été faits en matière de sécurité, en témoigne le nombre de recherches sur le sujet, de nombreuses variantes[3] du malware Mirai ont vu le jour, toutes s'appuyant sur d'autres failles plus récentes mais avec le même objectif que le virus originel, nous rappelant que les risques se sont développés à la même vitesse, voire plus vite, que la sécurité.

Puisqu'il nous paraît ainsi nécessaire de se pencher sur la sécurisation des IoTs, nous nous proposons d'établir une méthode simulant une attaque afin d'en comprendre les mécanismes. Plus précisément, nous simulerons une attaque passive, de l'écoute sur un réseau, qui vise à identifier les objets connectés actifs sur celui-ci.

Mais dans un premier temps, il convient de définir le périmètre sur lequel nous agissons, d'étudier les techniques d'écoute et de hacking déjà mises en place et les attaques déjà réalisées sur les objets.

[1] En réalité, le terme IoT désigne le mot Internet of Things, faisant état de la rencontre entre l'Internet et les objets, désormais communicants. Par abus de langage, on désigne souvent ces objets par le terme IoT.

[2] Avenir en japonais

[3] Okiru, Satori, Bashlite...

2 L'état de l'art

2.1 Les protocoles de transport pour les IoT

2.1.1 Zigbee

Zigbee est une technologie sans fil basée sur des normes, développée pour permettre des réseaux sans fil de machine à machine (M2M) et d'internet des objets (IoT) à faible coût et à faible puissance.

Zigbee est destiné aux applications à faible débit de données et à faible puissance et est une norme ouverte. En théorie, cela permet de mélanger les implémentations de différents fabricants, mais dans la pratique, les produits Zigbee ont été étendus et personnalisés par les fournisseurs et, par conséquent, ils sont en proie à des problèmes d'interopérabilité. Contrairement aux réseaux Wi-Fi utilisés pour connecter des points d'extrémité à des réseaux à haut débit, Zigbee prend en charge des débits de données beaucoup plus faibles et utilise un protocole de réseau maillé pour éviter les concentrateurs et créer une architecture auto-réparatrice.

La structure du système Zigbee se compose de trois types de dispositifs différents : le coordinateur Zigbee, le routeur et le dispositif final. Chaque réseau Zigbee doit être composé d'au moins un coordinateur qui agit comme une racine et un pont du réseau. Le coordinateur est responsable de la manipulation et du stockage des informations tout en effectuant des opérations de réception et de transmission de données. Les routeurs Zigbee agissent comme des dispositifs intermédiaires qui permettent aux données de transiter par eux vers d'autres dispositifs. Les dispositifs finaux ont une fonctionnalité limitée pour communiquer avec les nœuds parents, ce qui permet d'économiser l'énergie de la batterie. Le nombre de routeurs, de coordonnateurs et de dispositifs finaux dépend du type de réseau (en étoile, en arbre ou maillé). L'architecture du protocole Zigbee se compose d'une pile de différentes couches où l'IEEE 802.15.4 est défini par les couches physiques et MAC tandis que ce protocole est complété par l'accumulation des couches réseau et application propres à Zigbee.

Avec Zigbee, la vitesse de transfert des données est inférieure à celle du WiFi, et la vitesse la plus élevée est simplement de 250 kbps. C'est très peu par rapport à la vitesse réduite du WiFi.
Une autre qualité de Zigbee est le taux d'utilisation de l'énergie ainsi que la durée de vie de la batterie. Son protocole dure plusieurs mois car une fois qu'il est monté, on peut l'oublier.

2.1.2 Z-Wave

Z-Wave est né d'une idée de la société danoise Zensys en 1999, il est arrivé aux États-Unis en 2002. Z-Wave, comme Zigbee, est un protocole sans fil qui se concentre essentiellement sur la connectivité au sein de la maison intelligente.

Avec l'explosion de la popularité de la maison intelligente, de plus en plus de dispositifs connectés sont ajoutés aux maisons des gens. Un grand nombre de ces appareils - capteurs, ampoules, commandes de chauffage, serrures, prises et autres - utilisent Z-Wave pour communiquer entre eux. En fait, il y a plus de 100 millions d'appareils Z-Wave dans les maisons intelligentes du monde entier, avec un choix de plus de 3 500 appareils compatibles Z-Wave.

Moins puissant que le Wi-Fi, mais d'une portée bien plus grande que le Bluetooth, Z-Wave utilise des ondes radio à faible énergie pour communiquer d'un appareil à l'autre. D'un point de vue technique, Z-Wave fonctionne sur la gamme de fréquences radio 800-900 MHz, mais la seule raison pour laquelle vous pourriez vous y intéresser est que, contrairement à Zigbee qui fonctionne sur 2,4 GHz (une fréquence majeure pour le Wi-Fi), Z-Wave ne souffre pas vraiment de problèmes d'interférence majeurs.

La fréquence réelle à laquelle un appareil Z-Wave fonctionne dépend du pays dans lequel il est utilisé. Par exemple, les États-Unis utilisent 908,40, 908,42 et 916 MHz, tandis que le Royaume-Uni et l'Europe utilisent 868,40, 868,42 et 869,85 MHz. Il est donc important de s'assurer que vous achetez un appareil Z-Wave conçu pour votre région.

Contrairement au Wi-Fi, où les appareils doivent se connecter à un concentrateur central (généralement un routeur ou un autre point d'accès), les appareils Z-Wave se connectent tous ensemble pour former un réseau maillé.

Le concentrateur central de maison intelligente que vous utilisez pour gérer vos appareils Z-Wave se connecte à l'Internet, mais les appareils eux-mêmes (capteurs, ampoules, etc.) ne disposent d'aucune connexion Wi-Fi, ils utilisent simplement la connectivité Z-Wave pour communiquer avec le concentrateur, et cette connectivité n'a pas besoin d'être directe ; le réseau maillé signifie que les signaux peuvent sauter d'un appareil à l'autre.

Cependant, tous les appareils Z-Wave ne peuvent pas répéter un signal, certains peuvent simplement transmettre leur propre signal. C'est la raison pour laquelle les répéteurs Z-Wave sont populaires ; il s'agit essentiellement d'un appareil alimenté par le secteur, tel qu'une prise intelligente, qui peut recevoir et renvoyer un signal d'un autre nœud vers le concentrateur.

2.1.3 LoRa (long range and low capacity networks)

LoRa est une technologie sans fil qui offre une transmission de données longue portée, basse consommation et sécurisée pour les applications M2M et IoT.

LoRa peut être utilisé pour connecter sans fil des capteurs, des passerelles, des machines, des appareils, des animaux, des personnes, etc. au cloud. Les technologies LoRa fonctionnent dans différentes bandes de fréquences dans différentes régions : Aux États-Unis, elle fonctionne dans la bande 915 MHz, en Europe dans la bande 868 MHz et en Asie dans les bandes 865 à 867 MHz, 920 à 923 MHz. Cliquez ici pour en savoir plus sur les bandes de fréquences LoRa. La technologie LoRa a été créée par une société française appelée Cycleo, qui a ensuite été rachetée par Semtech en 2012. Semtech a été le membre fondateur de l'Alliance LoRa qui est maintenant l'organe directeur de la technologie LoRa. L'Alliance LoRa est l'une des alliances technologiques à la croissance la plus rapide. Cette association à but non lucratif compte plus de 500 entreprises membres, qui s'engagent à permettre le déploiement à grande échelle de réseaux étendus à faible consommation (LPWAN) IoT.

Grâce à une technique exclusive de modulation à spectre étalé dérivée de la technologie CSS (Chirp Spread Spectrum) existante, LoRa offre un compromis entre sensibilité et débit de données, tout en fonctionnant dans un canal à largeur de bande fixe de 125 KHz ou 500 KHz (pour les canaux de liaison montante), et de 500 KHz (pour les canaux de liaison descendante). En outre, LoRa utilise des facteurs d'étalement orthogonaux. Cela permet au réseau de préserver la durée de vie de la batterie des nœuds finaux connectés en optimisant de manière adaptative les niveaux de puissance et les débits de données d'un nœud final individuel. Par exemple, un dispositif final situé à proximité d'une passerelle doit transmettre des données avec un facteur d'étalement faible, car le budget de liaison est très limité. En revanche, un dispositif final situé à plusieurs kilomètres d'une passerelle devra transmettre avec un facteur d'étalement beaucoup plus élevé. Ce facteur d'étalement plus élevé permet d'augmenter le gain de traitement et la sensibilité de réception, mais le débit de données sera nécessairement plus faible.

LoRa est une mise en œuvre purement physique (PHY), ou couche "bits", telle que définie par le modèle de réseau à sept couches de l'OSI. Au lieu d'un câblage, l'air est utilisé comme support pour transporter les ondes radio LoRa d'un émetteur RF dans un dispositif IoT à un récepteur RF dans une passerelle, et vice versa.

En annexe, se trouve un tableau de comparaison entre LoRa et ZigBee. De même, s'y trouve une illustration de leur architecture réseau.

2.2 Matériel propre aux objets connectés grand public

2.2.1 Serveurs domotiques Jeedom

Tout d'abord, c'est quoi la domotique ? C'est rendre une maison intelligente et autonome. Une maison qui s'auto gère et agit pour nous, selon nos habitudes, sans pour avoir besoin de lui demander via une application. La domotique a plusieurs utilisations : l'économie d'énergie, gestion des éclairages ou des appareils électriques, détection de fuite d'eau, commande vocale, gestion intelligente du chauffage, gestion de l'ouverture-fermeture des volets, de l'arrosage ou encore protection anti-intrusion avec une alarme domotique constituée de quelques détecteurs d'ouvertures, mouvements et des caméras IP. Tout cela, en se reposant sur trois axes principaux : Sécurité, Energie et Confort.

Par exemple, une habitation qui vous protège des vols ou des accidents dans votre logement (détection de fuites ou d'incendies), tout en surveillant votre confort et en vous faisant économiser (réglage du chauffage, allumage optimisé de l'éclairage, suivi des consommations), le tout sans intervention de votre part. Aussi, une maison qui passera le chauffage en mode confort pendant les heures où vous êtes présent contribuant ainsi à votre confort mais aussi à l'économie, pour le reste il se chargera du passage réglé à une température plus basse.

Il s'agit également d'une fonction d'automatisation intérieure qui allumera les lumières pour vous lorsque vous êtes dans la pièce ou lorsque la nuit tombe à l'extérieur sans avoir à actionner un interrupteur. Il saura également ouvrir et fermer automatiquement les volets en fonction de la luminosité extérieure et de vos habitudes, et optimiser automatiquement le temps de filtration de la piscine, bien de consommation très appréciée. Tout cela est contrôlé par un serveur domotique (solution domotique) qui fait office de cerveau et s'appuie sur des objets domotiques, communément appelés périphériques. Ils fonctionnent et communiquent selon un protocole. Il en existe plusieurs comme Zwave, ZigBee ou Enocean qui sont les plus connus. Mais de nombreux autres protocoles existent et sont compatibles avec les solutions domotiques ouvertes. On y retrouve par exemple le Wifi, le Bluetooth. Pour notre projet, nous utiliserons Jeedom.

Concernant Jeedom, il s'agit d'une solution domotique française et présente plusieurs avantages : Il est autonome, donc indépendant du cloud. Il est multi-protocoles et compatible avec ZigBee, ZWave, EnOcean, KNX, LoRaWAN, BACnet, Modbus et plus encore. Le système de plugin via Jeedom Market garantit une compatibilité avec de nombreux protocoles actuels et futurs. Par conséquent, il fonctionne avec des centaines d'appareils domotiques. Enfin, il est open source et n'importe qui peut l'installer dans n'importe quelle configuration et développer des projets selon ses besoins.

Comment ajouter un appareil Zigbee à Jeedom ?
Pour faire reconnaitre votre appareil, allez dans "Plugins" "Protocole domotique" "ZigBee". Entrez votre appareil en mode appairage en suivant les instructions du fabricant et en activant "Mode inclusion" sur Jeedom, puis sélectionnez votre appareil dans la liste. Un message vert vous indiquera que l'inclusion est active pendant 3 minutes. L'appareil sera reconnu en quelques secondes. Pour les marques déjà répertoriées, vous aurez même le nom du constructeur et la référence de l'appareil. Dans les réglages généraux, renommez si vous le souhaitez votre appareil, choisissez sa catégorie, la pièce (objet parent) où il est installé et surtout sélectionnez "Activer" et "Visible". L'activation signifie que l'appareil sera contrôlé par Jeedom. Visible signifie que vous pourrez le voir dans votre tableau de bord. Il ne faut pas oublier de sauvegarder.

2.2.2 Le dongle

Un dongle ressemble à une clé USB mais n'en est pas toujours une. C'est un petit appareil électronique, qui se branche à un ordinateur via un port USB et qui a une fonction spécifique.

Un dongle peut désigner toutes sortes de matériels comme des périphériques de stockage ou mémoire (clés USB), permettant de détecter, scanner ou se connecter aux réseaux publics Wi-Fi, bluetooth, ou encore à du Zigbee etc. Il peut aussi se servir comme un outil d'antipiratage, Il agit également

comme un outil anti-piratage composé d'une carte mémoire et d'une carte MS avec un programme qui empêche le jeu de fonctionner en l'absence de clé.

Classiquement, on utilise le ConBee II. Il s'agit d'une passerelle ZigBee qui fonctionne sur un simple port USB qui permet de gérer directement dans ZigBee un grand nombre d'appareils domotiques utilisant ce protocole comme Ikea, Philips Hue, Xiaomi Aqara etc. Il est compatible avec de nombreux logiciels domotiques comme Jeedom via plugin "Deconz".

2.3 Protocoles de couches réseau et protocoles applicatifs

La couche transport (couche 4 dans le modèle TCP/IP) sert à ouvrir un canal de discussion pour les applications. Les protocoles correspondant sont généralement implémentés directement au niveau des machines des utilisateurs (on aura donc jamais de garantie de débit sur ceux-là).

On peut entendre parler de multiplexage et démultiplexage, cela signifie que ces protocoles rassemblent les messages des différentes applications sur le même canal, et les séparent à l'autre bout. Pour ça, on utilise des ports. Chaque protocole a son set de ports de 0 à 65536 qui identifie ce qui correspond à telles applications. Au niveau de l'application, on parle de socket, qui sont des objets qui permettent de gérer les ports. Il y a donc un port de source et un de destination, ils sont précisés dans le header du paquet.

Les ports 0 à 1023 sont des ports "connus" qui ne peuvent être utilisés que pour certaines applications.Ceux de 1024 à 49151 sont globalement libres d'utilisation, mais correspondent parfois à des produits connus (le port TCP 1194 pour OpenVPN par ex). Le reste sont les ports dynamiques qui peuvent être utilisés comme on veut. Les principaux protocoles sont l'UDP et le TCP que l'on va étudier dans la suite.

2.3.1 TCP/UDP

Protocole UDP :
Le protocole UDP est simple et rapide. Il est connectionless, il n'a pas de garanti de livraison ni d'ordre et pas de contrôle de flux. Il suffit donc d'avoir l'adresse IP et le port sur lequel on envoie les données. Il est utilisé pour des protocoles légers comme le DNS ou le DHCP. Les contrôles de qualité sont alors plus au niveau de l'application que du protocole. Le schéma est le suivant : port de source, port de destination, longueur, checksum, données.

Protocole TCP :
Le protocole TCP est beaucoup plus fiable que l'UDP. Il est connection-oriented (on établit une connection qu'on garde pour l'échange), renvoie les paquets perdus, garantit l'ordre, adapte le flow et la congestion. Le protocole est en full-duplex et est donc caractérisé par les IPs des deux membres et les deux ports correspondant.

Chaque segment envoyé a un numéro de séquence (son décalage par rapport au début de l'échange) et une longueur (les deux sont en octets). Lorsqu'on reçoit un paquet, on envoie immédiatement un paquet accusé de réception qui indique le numéro de séquence suivant à utiliser. Cela permet d'assurer la fiabilité. En effet, si on ne reçoit pas l'accusé de réception assez vite, on renvoie le paquet. En réalité, on n'attend pas qu'un paquet soit bien reçu avant d'envoyer le suivant, l'accusé de réception signifie que tous les paquets précédents ont été bien reçus.

Le header contient les ports de source et destination, les numéros de séquence et d'accusé de réception, et divers flags qui indique des choses sur le paquet (par exemple ACK signifie que le paquet est un accusé de réception)

2.3.2 HTTP

HTTP est un protocole de la couche applicative. HTTP signifie HyperText Transfer Protocol et est le protocole utilisé pour transférer des pages Web depuis 1990. Le protocole est bâti sur le protocole TCP. Il utilise le port 80 (ou 443 pour le HTTPS). Il utilise des requêtes indépendantes "stateless".

Le format d'une requête est le suivant : une première ligne avec la commande ou réponse, les headers puis les données. Le tout est en texte.

Il existe différentes commandes : GET, POST, HEAD, PUT, DELETE... La commande GET permet de demander une ressource au serveur. La commande HEAD permet de demander uniquement les headers au serveur. Ce sont des commandes safe, idempotent et cacheable. La commande POST envoie un payload (données utiles) au serveur. C'est une commande qui n'est ni safe, ni idempotent ni cacheable. La commande PUT permet de mettre à jour une cible avec les données et la commande DELETE de détruire la cible.

2.4 Protection des paquets

2.4.1 Clefs SSL

Introduction au protocole SSL :
Le protocole SSL (Secure Sockets Layer) est un protocole cryptographique qui permet de sécuriser les communications sur internet. En effet, il permet d'établir une connexion sécurisée entre deux machines communiquant sur internet ou sur un réseau privé. Il sert le plus souvent à crypter les données envoyées entre un navigateur web et un serveur web. Pour cela, il utilise des algorithmes de chiffrement pour brouiller les données pendant l'échange. Si le protocole SSL est utilisé, l'adresse HTTP passe à HTTPS, le "S" pour "secure". Le protocole SSL se situe entre la couche transport et la couche applicative.

Le protocole SSL permet donc d'assurer :

- **L'authentification :** le client et le serveur s'assurent de leur identité mutuelle. Cela est possible grâce à des certificats SSL.
- **La confidentialité :** Cela est possible grâce à des algorithmes de chiffrement.
- **L'identification et l'intégrité :** Les messages échangés ne sont pas modifiés et proviennent du bon expéditeur.

SSL utilise donc différents concepts cryptographiques tels que le chiffrement symétrique, le chiffrement asymétrique et la signature.

Chiffrement symétrique :
Le chiffrement symétrique utilise la même clé secrète pour chiffrer et déchiffrer le message. Tant que la clef reste secrète, il permet d'assurer la confidentialité. Malheureusement, il est difficile d'assurer que la clé reste secrète, elle peut être interceptée par un tiers lors de son échange. Le chiffrement symétrique est simple et requiert peu de temps de calcul.

Chiffrement asymétrique et signature :
Le chiffrement asymétrique utilise une clé différente pour chiffrer et déchiffrer le message. On chiffre avec une clé publique, accessible à tout le monde, et on déchiffre avec une clé privée que seul le destinataire possède.

Le chiffrement asymétrique permet également de signer et ainsi d'assurer l'identité de l'expéditeur du message. En effet, l'expéditeur code le message avec la clé privée, le destinataire le décode avec la clé publique et vérifie que le message correspond bien à celui d'origine.

En combinant signature et chiffrement clé publique, deux parties peuvent communiquer, mais cela est une méthode lourde et compliquée qui utilise beaucoup de ressources.

Fonction de hachage :
Une fonction de hachage calcule le "résumé" du texte, qui est bien plus court que le texte. Il est impossible de remonter au texte à partir du résumé, une légère modification du texte entraîne une grande modification de son résumé et deux textes différents ne peuvent pas avoir le même résumé. En envoyant un message et son résumé, le destinataire, en calculant le résumé, peut s'assurer de l'intégrité du message.

Le protocole SSL utilise le chiffrement symétrique pour assurer la confidentialité. Il utilise par contre le chiffrement asymétrique pour l'échange des clés. Il utilise des fonctions de hachage et la signature pour assurer l'intégrité et l'identification.

Les certificats SSL :
Le protocole SSL a recours à des certificats SSL. Le certificat SSL est un fichier de données qui lie une clé cryptographique aux informations d'une organisation. En effet, lorsqu'un navigateur ou un serveur tente de se connecter à un site Web sécurisé par le SSL, le navigateur/serveur demande au site Web de s'identifier. Alors, le serveur Web lui envoie une copie de son certificat SSL. Le navigateur/serveur vérifie le certificat SSL puis donne son accord au serveur Web. Celui-ci renvoie un accord signé électroniquement pour commencer une session SSL chiffrée. Dès lors, les données chiffrées sont échangées entre le navigateur/serveur et le serveur Web. Le certificat SSL contient entre autres les informations suivantes : l'algorithme de signature du certificat, la validité de celui-ci, le détenteur du certificat, les informations sur la clé publique (l'algorithme de clé publique et la clé publique).

Les protocoles SSL
Le protocole SSL est divisé en 4 sous-protocole : le protocole Handshake, le Change Cipher Spec Protocol, le protocole Alarm, le protocole Record

1) Le protocole Handshake :
Il permet au client et au serveur de s'authentifier mutuellement, de négocier les techniques de cryptage, les algorithmes MAC (Message Authentication Code) et enfin les clés symétriques qui seront utilisées pour le cryptage.
L'échange commence avec un message HELLO_CLIENT envoyé par le client au serveur en clair. Ce message contient entre autres la plus haute version SSL que peut utiliser le client, la session ID (le nombre 0 signifie que le client veut commencer une session), la CipherSuite qui est une liste des algorithme d(échange de clé et de chiffrement que supportent le client. Le serveur répond ensuite par un message HELLO_SERVEUR en clair. Il contient : l'identifiant que la session qui débute, La CipherSuite qui est cette fois la séquence d'algorithme choisis pour la session. Le serveur va pouvoir maintenant s'authentifier auprès du client et lui demander un certificat SSL. Une fois celui-ci vérifié, les clés de chiffrement symétriques sont générées. Il y a quatres clés utilisées : la clef pour la signature des messages du serveur, de même pour la signature des messages du client, la clef pour chiffrer les données émises par le serveur et de même pour le client. Ensuite, le client envoie le message CLIENT_FINISHED. Les deux parties communiquent maintenant à l'aide des clés. Le serveur fait de même.

2) Le protocole Change Cipher Spec:
Il n'y a qu'un seul message dans ce protocole : change_cipher_spec. Les deux parties l'envoient à la fin du protocole de négociation. Cette communication est chiffrée à l'aide de l'algorithme symétrique qui a été convenu précédemment.

3) Le protocole Alarm :
Les messages d'erreur qui peuvent être transmis entre les clients et les serveurs sont définis par ce protocole. Ces messages sont constitués de deux octets chacun. Le premier est soit un avertissement, soit fatal. La connexion est abandonnée si le niveau est fatal. Les autres connexions de la même session ne sont pas interrompues, mais aucune nouvelle connexion ne peut être établie. Le code d'erreur est encodé dans le deuxième octet.

4) Le protocole Record :
Le protocole Record encadre les autres protocoles SSL. Il assure l'encapsulation (les données sont transmises et reconnues sous une unique forme), la confidentialité grâce aux clés produites lors de la négociation, l'intégrité et l'identité grâce aux signatures MAC.

2.5 Logiciels d'écoute : capture de paquets Ethernet

2.5.1 Wireshark

Wireshark est un sniffer de paquets réseau qui s'occupe principalement de la capture de données brutes au niveau des paquets. Il est également utilisé pour analyser divers protocoles autres que http/https/tcp. Il agit aux niveaux inférieurs du modèle OSI (1 à 4). Cet outil est principalement utilisé par les ingénieurs réseau/sécurité.

Il propose trois méthodes de base pour capturer les paquets : l'interface graphique, Tshark (l'équivalent en ligne de commande de Wireshark) et Dumpcap (un programme dont le seul but est de capturer le trafic réseau, tout en conservant des fonctionnalités avancées comme la capture vers plusieurs fichiers). L'interface graphique est la technique la plus utilisée par les analystes réseau, mais ceux qui veulent capturer à partir de scripts ou qui ne veulent tout simplement pas passer par l'interface graphique utilisent Tshark ou Dumpcap. Pour une capture à long terme, Dumpcap est le meilleur choix.

2.5.2 BurpSuite

Burp Suite est un outil de test de pénétration des applications. Cet outil est considéré comme un serveur proxy web entre le navigateur et l'application cible et il agit sur la couche application (OSI-7) en trouvant des exploits et des vulnérabilités. Il est également appelé outil MITM qui traite le protocole http/https. Il est principalement utilisé par les développeurs et les responsables de la sécurité des applications.

2.6 Les Algorithmes de Learning

Dans cette sous-partie, nous étudierons comment, à partir des paquets récoltés grâce aux outils décrits précédemment, l'on concrétisera l'écoute du système. Concrètement, nous posons un cadre pour l'apprentissage des données et nous détaillons les outils pouvant être mis en place.

2.6.1 Les objectifs de prédiction

À l'aide des paquets obtenus lors d'une capture, l'on peut définir plusieurs objectifs de niveaux de difficulté variable mais dépendant aussi du protocole des objets :

- Identifier les objets connectés actifs sur le réseau : étude des paquets TCP/IP à partir des captures Wireshark et reconnaissance de patterns (apprentissage, cf. critères définis dans la suite du rapport).
- Identifier l'état des objets sur le réseau : étude du flux TCP/IP, détermination des objets connectés à un instant donné et à l'aide des logiciels d'écoute étudiés (Wireshark, Nmap), vérifier s'ils sont encore actifs sur le réseau.
- Identifier le niveau de criticité des paquets : déterminer quels paquets contiennent des informations importantes (i.e. confidentielles : identifiant, mot de passe, informations sur l'utilisateur...) par étiquetage des paquets avec BurpSuite puis apprentissage.
- Identifier le contenu des paquets : lecture en clair ou déchiffrage des paquets grâce à l'obtention de la clé.

2.6.2 Méthodologie

1) Identification des objets connectés actifs
On cherche ici à construire un algorithme d'apprentissage supervisé (cf. plus bas), c'est-à-dire que l'on est capable a priori d'associer chaque paquet à un objet. C'est sur cette base que l'on pourra vérifier

si notre algorithme est fiable. Dans le cas où l'on souhaite apprendre sans labellisation des données, il faudrait procéder autrement.
Il existe plusieurs manières de construire nos bases de données pour l'apprentissage à partir des captures réalisées.

1. On peut conserver la capture telle quelle et, d'une manière plutôt classique, construire une base de données qui liste les paquets. Elle contient alors les caractéristiques contenues dans ceux-ci, par exemple : le port source, le port de destination, l'adresse I.P. source, l'adresse I.P. de destination, la taille du paquet etc. Comme dans tout apprentissage statistique, on souhaite corréler les variables explicatives et la variable expliquée, pour qu'on puisse déterminer avec succès les objets possédant ces caractéristiques. L'algorithme d'apprentissage se construit alors sur les caractéristiques définies ci-dessus : les variables explicatives sont les caractéristiques que l'on peut extraire des paquets et la variable expliquée, l'identifiant de l'objet.

2. On peut toutefois procéder autrement en regroupant les paquets obtenus par session. On implémente des filtres dans la capture Wireshark pour regrouper les paquets par sessions (on rappelle que l'on est capable d'associer les paquets aux objets, et par conséquent de les regrouper par session). On est alors capables d'utiliser d'autres caractéristiques statistiques propres au regroupement par session : le temps d'émission moyen entre deux paquets, entropie de Shannon, taille moyenne d'une session.. (cf. caractéristiques plus bas dans le rapport).

On procède de la même manière en analysant le flux TCP/IP pour déterminer si un objet est actif ou non sur le réseau.

2) Identification du niveau de criticité d'un paquet
Dans la littérature dédiée, cette étude du niveau de criticité n'est réalisée que dans le cadre d'un réseau d'objets Wi-Fi tous disposant d'une application dédiée et connectés à une passerelle renvoyant elle-même à un cloud. Il s'agit encore d'apprentissage supervisé.
Le flux http est collecté par attaque Man in The Middle entre le téléphone et le cloud et on marque les paquets HTTP contenant les informations sensibles à l'aide du logiciel Burp Suite.
La base de données ne peut être construite par session et l'on étudie dès lors le flux http paquet par paquet. Comme précédemment, on choisit les caractéristiques et l'algorithme d'apprentissage afin de maximiser le taux de réussite.

3) Identification du contenu des paquets
Cette partie est délicate tant elle dépend du protocole utilisé et de la sécurité de celui-ci. Cette partie dépend moins de l'apprentissage que les autres dans la mesure où l'on pourrait, par apprentissage, préalable déterminer quels paquets contiennent les clés de sécurité.
Il semble possible d'obtenir dans certains cas les clés de sécurité dans un réseau Zigbee. L'obtention des clés de sécurité conditionne la lecture systématique du contenu des paquets. Il existe des frameworks (Killerbee, Secbee) capables d'intercepter les différentes clés utilisées dans le réseau Zigbee par sniffing (surveillance du trafic) et donc de déchiffrer les communications.

2.6.3 Caractéristiques courantes des algorithmes de reconnaissance :

Il existe déjà de nombreux algorithmes d'apprentissage pour la reconnaissance d'objets.
La plupart des caractéristiques sont directement extraites de la capture, d'autres sont plus complexes et calculées à partir des informations collectées :

- Protocole réseau/transport/applicatif utilisé
- Taille des paquets, nombre de paquets, temps moyen entre deux paquets
- Taille moyenne d'une session TCP

- IP Source/IP de destination, nombre d'adresses I.P.
- Calculs à partir du TTL
- Nombre de requêtes DNS, noms des DNS
- Protocole NTP, temps moyen entre deux synchronisations NTP
- Entropie de Shannon sur le payload

Si l'utilité de certaines caractéristiques est évidente, d'autres le sont visiblement moins. Le temps moyen entre deux synchronisations NTP permettrait d'identifier un objet dont les données seraient systématiquement datées, par exemple une caméra connectée.

2.6.4 Point sur l'apprentissage supervisé

En apprentissage supervisé, on dispose de données labelisées. C'est-à-dire qu'on dispose de n couples (X_i, Y_i) où Xi est dans $\mathbf{R^p}$ (p un entier non nul) et Y_i dans $\mathbf{R}$. On ne connaît pas la loi de $Y|X$ (Y sachant X), c'est-à-dire la distribution de Y connaissant les données d'entrée. C'est la relation que l'on souhaite retrouver, puisqu'en connaissant X, on sera capable de retrouver Y. On parle de régression lorsque Y est un réel quelconque et classification si on sait que Y appartient à un ensemble fini de labels. Dans le cas d'identification d'I.O.Ts sur le réseau, on s'intéresse plutôt à la classification.

On souhaite construire une fonction h, un prédicteur, telle qu'à toute donnée, h lui associe sa sortie correspondante (sa valeur Y_i). Disposant de données de départ labellisées, on est capable de savoir si notre prédicteur est adéquat ou non. Ce prédicteur se construit comme un minimiseur de distance, on minimise : $\sum_{i=1}^{n} d(Yi, h(Xi))$. La fonction d peut être comprise comme une distance entre deux points, elle est appelée fonction de perte et détermine le prédicteur.
Dans le cas de la régression linéaire, on cherche à trouver une relation linéaire entre Y et X, c'est-à-dire que l'on souhaite trouver un vecteur bêta, vérifiant : $\forall i \in \{1, ..., n\}, Y_i = \beta^T X_i$. On minimisera alors $\sum_{i=1}^{n} d(Y_i, \beta^T X_i))$. Et dans le cas d'une perte quadratique, cela revient à minimiser : $\sum_{i=1}^{n} ||Y_i - \beta^T X_i||^2$.
Ce principe est réutilisé plus largement dans les algorithmes suivants.

2.6.5 Les Algorithmes

Le Random Forest Classifier
Le Random Forest Classifier est sans doute l'un des modèles d'apprentissage les plus populaires pour à la fois son efficacité, sa complexité de calcul et sa relative simplicité théorique. On le retrouve souvent dans les algorithmes d'apprentissage pour la reconnaissance des objets connectés (Y. Meidan, M. Miettinen, A.Sivanathan, A. Acar).
Le Random Forest Classifier propose une amélioration à la classification par arbre de décision grâce aux phénomènes aléatoires.

Un arbre de décision est un algorithme de classification des données, il se présente visuellement sous la forme d'un arbre binaire constitué de nœuds intermédiaires, appelés nœuds de décision, et de nœuds finaux. Le résultat de la classification se trouve dans les nœuds finaux, c'est-à-dire dans les dernières feuilles de chaque branche. Les nœuds de décision contiennent des conditions sur les caractéristiques permettant de diviser la base de données. Chaque nœud de décision crée deux nœuds fils, qu'on souhaite plus proche de la pureté (i.e. d'une même classe) que le nœud précédent. Le choix de ces conditions se fait localement par la maximisation du gain d'information lors d'un split. Le gain d'information est un indicateur permettant de mesurer à quel point le désordre a été réduit. Or le désordre induit par un état est mesuré par une fonction d'impureté, qui nous indique la pertinence de la séparation réalisée. On choisit généralement l'entropie, qu'on cherche donc à minimiser. On rappelle la formule de l'entropie (insérer la formule de l'entropie) où les probabilités sont chacune définies

comme la probabilité d'obtenir une classe plutôt qu'une autre. On maximise l'information récupérée par la séparation d'un nœud parent, et on obtient deux nœuds fils optimaux dont on est certains qu'ils contiennent le maximum d'information pour la classification. C'est ainsi qu'on détermine quels sont les nœuds de décision optimaux. Concrètement, sur un plan, un nœud de décision est une droite séparant les données entre elles, et l'on atteint la pureté maximale lorsque l'on obtient des sous-ensembles ne contenant qu'un seul type de donnée, c'est-à-dire lorsque les données du sous-ensemble sont classifiées. Ainsi, récursivement, on obtient un certain nombre de nœuds finaux, tous contenant des objets différents de la base de données, et d'entropies minimales. Dans la phase d'apprentissage, il s'agit d'identifier les nœuds de décision et d'attribuer les classes aux nœuds finaux d'après la labellisation fournie. Dans la phase de test, il s'agit simplement d'appliquer la même méthode et de trier les données selon les nœuds de décision obtenus.

L'algorithme Random Forest Classifier utilise des propriétés aléatoires pour réduire les erreurs d'estimation. Tout d'abord, n bases de données vont être créées de manière aléatoire à partir de la base de données originale. Les lignes d'un dataframe crée aléatoirement peuvent être redondantes (si l'on s'intéresse à la classification de capteurs, on pourrait par exemple avoir plusieurs fois la même adresse I.P. dans les tableaux aléatoires). Cette méthode est appelée le bootstrapping.
À chaque base de données va être associé un nombre aléatoire de caractéristiques, différentes au sein de chaque arbre, mais pouvant réapparaître d'un arbre à l'autre. Ces caractéristiques sont issues de la base de données originale et ont été calculées au préalable. Nous obtenons ainsi des bases de données construites aléatoirement, tant au niveau du contenu brut que des caractéristiques. Un arbre de décision est construit pour chacune de ces bases de données aléatoires, de la même manière que précédemment. Ils dépendent généralement de caractéristiques différentes, donc les prédictions pour une même valeur d'entrée peuvent différer. Chaque arbre renvoie une prédiction, il vote pour une valeur. Le prédicteur final est construit par vote à la majorité, les arbres possédant tous le même poids dans le vote final. Si la majorité des arbres renvoie par exemple la valeur 0, ce sera celle-ci qui sera choisie pour le prédicteur final. 'Cette méthode est appelée l'agrégation.
L'apprentissage se construit ainsi par bootstrapping et par agrégation, méthode appelée bagging. L'utilisation de procédés aléatoires permet à l'algorithme de réduire la dépendance aux données d'origine, de réduire la corrélation entre les arbres (qu'on pourrait construire avec les mêmes caractéristiques, mais qui renverraient des prédictions similaires) et d'évaluer l'importance des caractéristiques. L'on peut s'attendre à ce que certains arbres construits sur des caractéristiques peu significatives renvoient des prédictions faussées, néanmoins l'utilisation de procédés aléatoires permet une compensation certaine des résultats extrêmes.

Concernant les inconvénients, sa complexité de calcul reste élevée et il est difficile de comprendre où se situe le problème lorsque les résultats s'avèrent peu satisfaisants.

Gradient Boosting XGBoost:
Le XGBoost est un algorithme d'apprentissage très performant lorsqu'il s'agit de problèmes peu complexes de régression ou de classification. L'algorithme est une version optimisée du Gradient Boosting. Il est utilisé par E.Tsukerman dans Machine Learning for Cybersecurity (2019) et par B. Bezawada dans son algorithme de reconnaissance IoTSense.

Le Gradient Boosting est un algorithme de classification sous forme de forêt. Les arbres sont construits les uns après les autres, chacun dépendant de la prédiction réalisée par l'arbre précédent.
Le premier arbre a une unique prédiction : la moyenne des Yi, soit la moyenne des observations. Le second arbre calcule l'écart entre la moyenne des observations et la valeur réelle des labels. On appelle résidu l'écart entre la valeur prédite par un arbre et la valeur réelle observée. Chaque arbre va essayer d'estimer les résidus de l'arbre précédent (en minimisant une fonction de perte choisie au préalable), multiplié par une pondération appelée Learning Rate. À partir de cette estimation des résidus on obtient une nouvelle estimation pour les labels donc un nouveau résidu. Les estimations déjà très proches des valeurs réelles seront inchangées. La prédiction finale est l'estimation du label issue des calculs du dernier arbre.

L'intérêt du XGBoost est de réduire la complexité du Gradient Boosting en ne conservant que les arbres performants.

2.6.6 Évaluation de la performance des algorithmes

Les indicateurs d'évaluation de la performance des algorithmes dépendent de la variable à estimer. Dans le cas de la régression, l'on peut décider d'un écart de la valeur prédite à la valeur réelle satisfaisant, par exemple à 5
Dans le cas de la classification, il existe des indicateurs classiques significatifs qui peuvent aider au choix de l'algorithme. Dans le cas de la classification binaire (la variable à prédire prend la valeur 0 ou 1), on a :

- Le nombre de Vrais Positifs (VP) : éléments positifs classés positifs
- Le nombre de Vrais Négatifs (VN) : éléments négatifs classés négatifs
- Le nombre de Faux Positifs (FP) : éléments négatifs classés positifs
- Le nombre de Faux Négatifs (FN) : éléments positifs classés négatifs
- La sensibilité (TVP) : ratio entre le nombre de vrais positifs et le nombre réel de positifs bien ou mal classés (VP + FN)
- La spécificité (TVN) : ratio entre le nombre de vrais négatifs et le nombre réel de négatifs bien ou mal classés (VN + FP) L'idée de positif et de négatif fait référence aux valeurs que peut prendre Y, ici 0 ou 1. On détermine également :
- Le taux de non-détection, 1-TVP, égal au taux de faux négatifs
- Le taux de fausses alarmes, 1-TVN, égal au taux de faux positifs Dans de nombreux cas, on souhaite avoir un taux de non-détection plutôt élevé, quitte à avoir plus de fausses alarmes.

Dans le cas de la classification non binaire, on peut se référer à une matrice de confusion qui généralise à un nombre de cas plus élevés les indicateurs présentés ci-dessus.

2.7 Point sur le HackRF

Introduction
HackRF One a été créé en 2014 par la société Great Scott Gadget. C'est un émetteur-récepteur semi-duplex capable de travailler sur des fréquences allant de 1MHz à 6000 MHz avec une puissance de signal allant de 1 MW à 30 MW selon la bande. C'est un périphérique de radio logiciel. HackRF one permet donc de pirater toutes les fréquences radios comprises entre 1MHZ et 6GHz.
HackRF One permet d'hacker les IoT. En effet, comme vu précédemment, les appareils IoT utilisent la communication sans fil via RF ou Zigbee ou LoRa. Il peut capter une large gamme de protocoles du GSM au Z-wave.Il est utlisé notamment pour effectuer des attaques BlueBorne (détaillé plus tard) ou Airborne, par exemple le brouillage, la relecture, ect...

Utilisation
HackRF One s'utilise avec une antenne. Ubuntu Linux est recommandé comme système d'exploitation. On branche le HackRF One sur l'ordinateur à l'aide du câble Micro-USB vers USB. Les bibliothèques du HackRF One doivent être installées sur l'ordinateur à l'aide des commandes sur le terminal "sudo apt-get install hackrf". Il faut également installer le logiciel GNU Radio Companion. Ce logiciel permet de recevoir et d'émettre des signaux radio FM.

2.8 Attaques sur les IoTs

2.8.1 Man in The Middle

Une attaque de type "man-in-the-middle" est un type d'attaque par écoute, où les attaquants interrompent une conversation existante ou un transfert de données. Après s'être insérés au milieu du transfert, les attaquants se font passer pour les deux participants légitimes. Cela permet à l'attaquant d'intercepter des informations et des données de l'une ou l'autre partie tout en envoyant des liens malveillants ou d'autres informations aux deux participants légitimes d'une manière qui pourrait ne pas être détectée avant qu'il ne soit trop tard.
On peut comparer ce type d'attaque au jeu du téléphone, où les mots d'une personne sont transmis d'un participant à l'autre jusqu'à ce qu'ils soient modifiés au moment où ils atteignent la dernière personne. Dans une attaque de type "man-in-the-middle", le participant intermédiaire manipule la conversation à l'insu des deux participants légitimes, dans le but de récupérer des informations confidentielles ou de causer d'autres dommages.
on recense différents types d'attaques :

- IP Spoofing implique un attaquant qui se déguise en application en modifiant les en-têtes de paquets dans une adresse IP. En conséquence, les utilisateurs qui tentent d'accéder à une URL connectée à l'application sont envoyés sur le site web de l'attaquant.
- ARP Spoofing (Poisoning) consiste à associer l'adresse MAC d'un attaquant à l'adresse IP d'un utilisateur légitime sur un réseau local à l'aide de faux messages ARP. En conséquence, les données envoyées par l'utilisateur à l'adresse IP de l'hôte sont plutôt transmises à l'attaquant. (exemple: Ettercap)
- DNS Spoofing (Poisoning) consiste à infiltrer un serveur DNS et à modifier l'enregistrement de l'adresse d'un site web. En conséquence, les utilisateurs qui tentent d'accéder au site sont envoyés par l'enregistrement DNS modifié vers le site de l'attaquant.

Application ZigBee : KillerBee
ZigBee est l'un des protocoles les plus largement utilisés dans les appareils IoT d'aujourd'hui. Depuis l'introduction de ZigBee en 2004, il a rapidement gagné en popularité et est devenu l'un des protocoles les plus largement utilisés pour tout ce qui nécessite une automatisation et un contrôle radio à courte portée.
Les fonctionnalités supplémentaires du ZigBee, telles que la faible consommation d'énergie, la prise en charge intégrée des réseaux maillés et la faible utilisation de la bande passante, ont également contribué à accroître encore l'adoption de cette nouvelle technologie. Actuellement, il s'agit de l'un des protocoles les plus largement utilisés dans les appareils IoT et comprend des produits populaires tels que Samsung Smart Things et Philips Hue.
En raison du manque de sensibilisation à la sécurité du ZigBee, il est courant de rencontrer des appareils basés sur ZigBee qui n'ont même pas la moindre protection de sécurité. L'impact de ces vulnérabilités peut être n'importe où, de l'espionnage de données sensibles en transit au détournement d'infrastructures critiques. KillerBee est un outil pour tester et auditer les réseaux ZigBee et IEEE 802.15.4. Il est conçu pour simplifier le processus de détection des paquets à partir de l'interface radio ou des fichiers de capture de paquets pris en charge.
Ses caractéristiques sont:

- Permet la simplification de l'insertion de n'importe quel paquet.
- Faire du sniffing et l'injection de trafic, décodage et manipulation de paquets
- La reconnaissance, outils d'assistance à l'exploitation
- Minimise les dépendances de la bibliothèque
- Fonctionne bien avec Python.

2.8.2 Attaque par rejeu (replay attack)

Une attaque par rejeu se produit lorsqu'un utilisateur non autorisé capture le trafic réseau et envoie une communication à sa destination d'origine. La destination d'origine agit comme l'expéditeur d'origine. Pour empêcher une attaque par rejeu réussie, on peut implémenter des horodatages et des numéros de séquence. Cela permet au système d'authentification d'accepter uniquement les paquets réseau contenant le numéro de séquence d'horodatage approprié. Si l'horodatage est après un certain temps, le paquet sera abandonné. On considère un serveur, et un attaquant qui renifle et transmet le trafic au serveur. Le pirate intercepte la communication et force la requête comme s'il s'agissait de l'expéditeur. Les attaques par rejeu sont toujours l'une des vulnérabilités sans fil les plus courantes dans les appareils IoT, mais ce n'est que le début des types de vulnérabilités et d'exploits qui peuvent être mis en œuvre.

2.8.3 Exploit

L'exemple de Blueborn

BlueBorne est un vecteur d'attaque par lequel les pirates peuvent tirer parti des connexions Bluetooth pour pénétrer et prendre le contrôle total des appareils ciblés. BlueBorne affecte les ordinateurs ordinaires, les téléphones mobiles et le domaine en expansion des appareils IoT. L'attaque ne nécessite pas que l'appareil ciblé soit apparié à l'appareil de l'attaquant, ni même qu'il soit réglé en mode découvrable. Armis Labs a identifié huit vulnérabilités de type "zero-day" jusqu'à présent, ce qui indique l'existence et le potentiel du vecteur d'attaque. Armis pense que de nombreuses autres vulnérabilités attendent d'être découvertes dans les différentes plateformes utilisant Bluetooth. Ces vulnérabilités sont pleinement opérationnelles et peuvent être exploitées avec succès, comme l'ont démontré nos recherches. Le vecteur d'attaque BlueBorne peut être utilisé pour mener un large éventail d'infractions, notamment l'exécution de code à distance et les attaques de type Man-in-The-Middle.
Le vecteur d'attaque BlueBorne possède plusieurs qualités qui peuvent avoir un effet dévastateur lorsqu'elles sont combinées. En se propageant par voie aérienne, BlueBorne cible le point le plus faible de la défense des réseaux - et le seul qu'aucune mesure de sécurité ne protège. En se propageant par voie aérienne d'un appareil à l'autre, BlueBorne est également très contagieux. De plus, comme le processus Bluetooth jouit de privilèges élevés sur tous les systèmes d'exploitation, son exploitation permet un contrôle quasi total de l'appareil. Malheureusement, cet ensemble de capacités est extrêmement intéressant pour un pirate. BlueBorne peut servir n'importe quel objectif malveillant, comme le cyber espionnage, le vol de données, les ransomwares, et même la création de grands botnets à partir de dispositifs IoT comme le botnet Mirai ou de dispositifs mobiles comme avec le récent botnet WireX. Le vecteur d'attaque BlueBorne surpasse les capacités de la plupart des vecteurs d'attaque en pénétrant dans des réseaux sécurisés "air-gapped" qui sont déconnectés de tout autre réseau, y compris internet.

Logiciels : Nmap et Metasploit

Nmap

Nmap, abréviation de Network Mapper, est un outil gratuit et open source utilisé pour la vérification des vulnérabilités, l'analyse des ports et, bien sûr, la cartographie des réseaux. Bien qu'il ait été créé en 1997, Nmap reste l'étalon-or par rapport auquel tous les autres outils similaires, qu'ils soient commerciaux ou open source, sont jugés. Nmap a conservé sa prééminence grâce à l'importante communauté de développeurs et de codeurs qui contribuent à sa maintenance et à sa mise à jour. La communauté Nmap rapporte que l'outil, que tout le monde peut obtenir gratuitement, est téléchargé plusieurs milliers de fois chaque semaine. Grâce à son code source ouvert et flexible, il peut être modifié pour fonctionner dans la plupart des environnements personnalisés ou hautement spécialisés. Il existe des distributions de Nmap spécifiques aux environnements Windows, Mac et Linux, mais Nmap supporte également des systèmes d'exploitation moins populaires ou plus anciens comme Solaris, AIX ou AmigaOS. Le code source est disponible en C, C++, Perl et Python.

Le cœur de Nmap est le scan de ports. Son fonctionnement est le suivant : les utilisateurs désignent une liste de cibles sur le réseau sur lesquelles ils veulent obtenir des informations. Les utilisateurs n'ont pas besoin d'identifier des cibles spécifiques, ce qui est une bonne chose car la plupart des administrateurs n'ont pas une image complète de tout ce qui utilise les milliers de ports potentiels sur leur réseau. Au lieu de cela, ils compilent une série de ports à analyser. Il est également possible d'analyser tous les ports du réseau, mais cela prendrait beaucoup de temps et consommerait une grande partie de la bande passante disponible. De plus, selon le type de défenses passives utilisées sur le réseau, un tel scan massif des ports déclencherait probablement des alertes de sécurité. C'est pourquoi la plupart des gens utilisent Nmap dans des déploiements plus limités ou divisent différentes parties de leur réseau pour un scan programmé dans le temps.

En plus de configurer une série de cibles à scanner, les utilisateurs peuvent également contrôler la profondeur de chaque scan. Par exemple, une analyse légère ou limitée peut renvoyer des informations sur les ports ouverts et ceux qui ont été fermés par les paramètres du pare-feu. Des scans plus détaillés peuvent également capturer des informations sur le type d'appareils qui utilisent ces ports, les systèmes d'exploitation qu'ils utilisent et même les services qui y sont actifs. Nmap peut également découvrir des informations plus approfondies, comme la version des services découverts. Cela en fait un outil parfait pour trouver des vulnérabilités ou aider aux efforts de gestion des correctifs.

Le contrôle des scans nécessitait auparavant des commandes de la console, ce qui signifie bien sûr qu'une certaine formation était nécessaire. Mais la nouvelle interface graphique de Zenmap permet à tout le monde de dire à Nmap ce qu'il veut qu'il découvre, avec ou sans formation. Pendant ce temps, les professionnels peuvent continuer à utiliser les commandes de la console qu'ils ont toujours utilisées, ce qui en fait un outil utile pour les experts comme pour les novices.

Metasploit

Le framework Metasploit est un outil très puissant qui peut être utilisé par les cybercriminels comme par les hackers éthiques pour sonder les vulnérabilités systématiques des réseaux et des serveurs. Comme il s'agit d'un framework open-source, il peut être facilement personnalisé et utilisé avec la plupart des systèmes d'exploitation.

Avec Metasploit, l'équipe de pen testing peut utiliser du code prêt à l'emploi ou personnalisé et l'introduire dans un réseau afin de rechercher les points faibles. Une fois les failles identifiées et documentées, les informations peuvent être utilisées pour remédier aux faiblesses systémiques et hiérarchiser les solutions.

En raison de son large éventail d'applications et de sa disponibilité en open-source, Metasploit est utilisé par tout le monde, des professionnels de DevSecOps aux pirates informatiques. Il est utile à tous ceux qui ont besoin d'un outil fiable et facile à installer, capable de faire le travail, quelle que soit la plate-forme ou la langue utilisée. Le logiciel est populaire auprès des pirates et largement disponible, ce qui renforce la nécessité pour les professionnels de la sécurité de se familiariser avec le framework, même s'ils ne l'utilisent pas.

Metasploit comprend maintenant plus de 1677 exploits organisés sur 25 plateformes, dont Android, PHP, Python, Java, Cisco, et plus encore. Le framework comporte également près de 500 charges utiles, dont certaines comprennent :

- Des charges utiles de shell de commande qui permettent aux utilisateurs d'exécuter des scripts ou des commandes aléatoires contre un hôte.
- Des charges utiles dynamiques qui permettent aux testeurs de générer des charges utiles uniques pour échapper aux logiciels antivirus.
- Charges utiles de compteur qui permettent aux utilisateurs de réquisitionner les moniteurs de périphériques à l'aide de VMC et de prendre le contrôle de sessions ou de charger et télécharger des fichiers
- Charges utiles statiques permettant le transfert de port et les communications entre réseaux

L'architecture du framework est la suivante :

- Exploits : Ce sont des scripts nous permettent d'exploiter une vulnérabilité sur une machine distante.
- Payload (charge utile): le code exécuté après s'être introduit dans la machine cible, afin de contrôler la machine victime.
- Axillaires: Des petits codes qui servent pour diverses taches (scan de port,. . .)
- Encodeurs: le travail des encodeurs est d'obscurcir notre exploit et charge utile de telle manière que dans le système cible, il passe inaperçu par tous les systèmes de sécurité.
- NOPS : nous évite d'être détecté
- Post: des modules qui contiennent divers scripts qui nous aident à filtrer davantage notre système cible après une exploitation réussie
- Evasion: nous aide à modifier les charges utiles pour éviter la détection.

3 Rapport Projet

3.1 Environnement

L'Exocube a mis à notre disposition une infrastructure simple, permettant de simuler un réseau domestique grand-public.

Ce réseau regroupe les appareils suivants :

- 1 machine Linux Ubuntu (PC)
- 1 Raspberry Pi
- 1 dongle Zigbee de marque ConBee (v2)
- 1 doorSensor de marque SonOff
- 3 détecteurs de Présence de marque SonOff, Xiaomi et eWeLink
- 1 détecteur de fumée de marque Frient
- 1 prise connectée de marque Aqara
- 1 capteur de pression/température de marque Aqara

Ce réseau comporte exclusivement des objets connectés Zigbee. Sur la Raspberry Pi, un serveur Jeedom tourne en permanence. Il permet d'administrer le réseau (appareillage des objets Zigbee via le dongle, édition de scénarios ...) et de stocker les informations émises par les objets. L'accès au serveur Jeedom se fait via le browser de la machine Linux, en envoyant une requête à son adresse IP.

La Raspberry Pi est un nœud central du réseau : elle est connectée au switch par le port RJ45, elle héberge le serveur Jeedom et le dongle Zigbee est connecté à son port USB. Tous les paquets émis par les objets connectés ainsi que les requêtes Jeedom transitent par donc la Raspberry Pi. Grâce à cette configuration, il est possible de capturer tous les paquets d'intérêt sur l'interface Loopback de la Raspberry Pi.

Problèmes

1. **Le dongle Zigbee doit avoir une configuration IP**
   Un des deux dongles Zigbee de l'Exocube ne possède pas de configuration IP : il transmet les paquets au serveur Jeedom via un protocole réseau (ou transport) inconnu. Du point de vue TCP/IP, ce dongle est transparent . Le problème est que le logiciel de capture Wireshark n'identifie pas les paquets qui transitent par ce dongle. De plus, les modules python d'analyse de fichiers pcap sont développés pour traiter essentiellement des frames TCP/IP. C'est pourquoi nous avons dû nous replier sur le dongle ConBee 2 qui possède par défaut une configuration IP.

2. **L'administration du réseau par Jeedom**
   L'administration du réseau par Jeedom est superficielle du point de vue de l'appareillage des objets. La fonctionnalité déconnexion ne désappareille pas les objets Zigbee : elle donne seulement l'ordre à Jeedom de ne plus afficher les objets. En arrière-plan, les sensors continuent d'émettre et le dongle continue de POST les données au serveur Jeedom. Pour isoler un objet Zigbee du réseau, il faut donc le désappareiller manuellement.

3. **Les requêtes Browser $->$ Jeedom**
   Lorsque que l'on cherche à administrer le serveur Jeedom via le browser, les requêtes http entre le serveur et le browser viennent perturber les captures Wireshark sur l'interface loopback de la Raspberry Pi. Tout comme les frames provenant du dongle Zigbee, ces frames sont à destination du port 80. Il faut trouver un moyen d'exclure ces paquets des fichiers de capture, car ils ne sont le fait d'aucun objet Zigbee et risqueraient de tromper notre algorithme. La solution trouvée est de ne pas afficher Jeedom dans un browser.

4. **Les requêtes autonomes du serveur Jeedom**
   Le serveur Jeedom ne fonctionne pas de manière passive . Pour actualiser ses données, Jeedom envoie parfois des requêtes au dongle. Lors de l'activation d'un scénario, Jeedom envoie des commandes aux sensors/actuators concernés sur le réseau. Ces requêtes sont doublement problématiques :

   - Elles ne sont le fait d'aucun objet connecté.
   - Elles poussent le dongle et les objets Zigbee à réémettre leurs données en cache, ce qui fausse leur fonctionnement normal.

   Pour ces deux raisons, il faut empêcher Jeedom d'émettre des requêtes : pas d'affichage dans un browser, limitation des fonctionnalités non-essentielles, encadrement des scénarios...

3.2 Objectifs

Construire un algorithme qui détecte et identifie les appareils connectés d'un réseau domestique. L'identification des objets connectés peut être complète (type, constructeur, modèle) ou bien simplement par type (capteur de présence, capteur de température...).

L'identification des objets se fait à partir de captures des trames réseaux sur une durée moyenne (30 minutes).
La Base de données construite à partir des captures Wireshark ne peut contenir que des informations lisibles en clair , c'est-à-dire situées dans l'entête de la couche TCP/IP : taille des paquets, timestamp, port source destination, IP source et destination...

Problèmes

1. **La capture sur l'interface Loopback**
   Dans la configuration réseau décrite plus haut, il convient de capturer sur l'interface loopback de la Raspberry Pi pour intercepter toutes les frames provenant du dongle Zigbee. L'inconvénient est que l'on perd des informations exploitables, car les adresses IP source destination des frames sont remplacée par l'IP de l'interface (127.0.0.1).

2. **Le formatage des paquets par le dongle ConBee**
   L'utilisation du dongle Conbee supprime l'essentiel des informations exploitables, car le processus d'envoi des données à Jeedom est identique quel que soit l'objet :

   - Ouverture d'une session TCP avec le port 80 du serveur Jeedom. Le numéro de port source choisi n'est lié ni l'objet émetteur, ni au type d'information à transmettre ; il est égal au numéro de port de la dernière sessions TCP ouverte incrémenté de 2.
   - Formatage des informations à transmettre dans un fichier json, envoyé à Jeedom dans une requête HTTP POST.
   - Fermeture de la session TCP

   De ce fait, quel que soit l'objet Zigbee, les paquets capturés sont quasiment identiques : même IP source (IP du dongle), même IP dest (Jeedom) et port dest (80), même nombre de paquets par sessions, contenu des frames HTTP formaté...

3.3 Méthodologie

Notre travail se divise en trois parties/modules :

- La création d'une base de données d'apprentissage
- L'entrainement des algo de Machine Learning
- La création d'une base de données dite de test , à partir de laquelle les algorithmes doivent faire des prédictions.

On commence donc par construire une large base de données d'apprentissage. Cette regroupe des données tirées de 78 captures réseau de 30 minutes chacune. Les étapes d'extraction des données pour une capture sont résumées ci-dessous:

- Regroupement des frames IP par session TCP.
- Filtrage des sessions d'intérêt, c'est-à-dire à destination du port 80 (Jeedom)
- Extraction des données contenues dans les paquets de chaque session : timestamp, taille, ID Zigbee de l'appareil émetteur si le paquet est un POST HTTP (Lecture du fichier JSON).
- Construction des données pour chaque session : début et fin de la session, taille de la session, nombre de paquets, port ouvert par le dongle, ID simplifié de l'appareil émetteur.
- Regroupement des sessions par appareil émetteur (grâce à l'ID simplifié).
- Construction de données pour chaque appareil émetteur : temps moyen entre deux sessions consécutives (sleepTimeMean) et variance des temps entre les sessions (sleepTimeMean), durée moyenne de sessions et taille moyenne des sessions.

Voici un aperçu des données tirées d'une capture réseau :

	SensorID	SessionDurationMean	SessionBytesMean	SleepTimeMean	SleepTimeVariance
2	3.0	0.03844970402527076	2088	3.249177442104883	0.8140156744033704
3	2.0	0.02044940320547945	2259	24.19317429263889	687.1465284391641
4	6.0	0.021005359103448275	2273	62.94858784335714	75.58195319926637
5	4.0	0.014486074199999999	2213	301.2606232165	315430.2305065531
6	1.0	0.020889759	2273		
7	7.0	0.018957961727272727	2188	77.13543499085715	17346.853534137066
8	5.0	0.041063308500000006	2191	0.24353170400000002	

La construction de cette database repose sur deux conditions contraignantes :

- Avoir des droits root sur la machine qui héberge le serveur Jeedom pour effectuer une capture sur l'interface loopback
- Pouvoir lire en clair le contenu des paquets transmis pour repérer l'ID de l'appareil Zigbee émetteur. Même si la database ne comporte in fine que des données des headers TCP/IP, l'étape de regroupement des sessions par objet émetteur nécessite d'avoir accès à cet ID.

La seconde partie consiste à fusionner les données extraites des 78 captures puis à les passer en paramètre de deux algorithmes de machine Learning : XGBoost et RandomForest. On ajuste les paramètres des deux modules (taux d'entrainement, nombre d'arbres...) pour améliorer les résultats, et on vérifie qu'il n'y a pas de surapprentissage (cf. Analyse des résultats).

Enfin, on construit les bases de données test à partir desquelles l'algorithme fait des prédictions. Cette fois ci, on doit construire les bases de données en respectant les objectifs établis plus haut. Par conséquent, on ne peut donc plus lire les paquets JSON pour distinguer les sessions TCP émis par tel ou tel autre objet. On se place alors dans une configuration simplifiée : le réseau ne comporte qu'un seul objet Zigbee (en plus du dongle) qu'il faut reconnaitre.
On procède de la manière suivante :

- Déconnexion manuelle de tous les sensors Zigbee à l'exception d'un seul.
- Capture sur l'interface loopback de la Raspberry Pi (30 minutes)
- Regroupement des frames par Session TCP, filtrage des sessions à destination du port 80
- Extraction des données dans les hearder TCP/IP
- Construction des données par session, puis par capture

Voici un aperçu de la database envoyée à l'algorithme de prédiction :

	A	B	C	D
	SessionDurationMean	SessionBytesMean	SleepTimeMean	SleepTimeVariance
	0.020046640735294118	2264	53.410923018606056	443.8363095525379

Attention, la base de données test est calculée en fusionnant les sessions du sensor Zigbee et celles émises par le dongle (car on ne peut plus les distinguer). Par conséquent, les informations transmises à l'algorithme sont bruitées par les frames du dongle. Voici la base de données que l'on aurait obtenu en filtrant préalablement les frames du dongle :

	SessionDurationMean	SessionBytesMean	SleepTimeMean	SleepTimeVariance
	0.014486074199999999	2213	301.2606232165	315430.2305065531

Lorsque le sensor émet beaucoup de trames sur 30 minutes, les sessions du dongle sont minoritaires et trompent peu l'algorithme. Le cas inverse, la précision de l'algorithme baisse drastiquement (cf. Analyse des résultats).

3.4 Explication du code

3.4.1 Installation et principales commandes

Installation des bibliothèques python requises :

```
pip install -r requirements.txt
```

Principales commandes :

```
python main.py buildDB all
python main.py buildDB directory Captures/your_new_captures
python main.py buildDB files Captures/new_capture_1 Captures/new_capture_2
python main.py mergeDB -dest DB/
python main.py learn
python main.py guess -path "Captures/your_capture.pcap"
```

3.4.2 Structure du Code

```
.
├── Builder
│   ├── __init__.py
│   ├── builder.py
│   ├── divide_pcap.py
│   ├── helpers_db.py
│   ├── helpers_extractor.py
│   ├── helpers_pcap.py
│   └── sensors_tags.txt
├── Captures
├── DB
├── Guesser
│   ├── guess.py
│   └── helpers_guess.py
├── Learner
│   ├── TrainingDB
│   │   └── trainingDB.csv
│   ├── helpers_learn.py
│   └── learn.py
├── README.md
├── capture_script.sh
├── main.py
└── requirements.txt
```

Architecture du code Python

Le dossier Builder/ contient les scripts python pour construire des bases de données à partir des captures PCAP.

Le fichier *Builder/sensor_tags.txt* contient les ID Zigbee long, ID réduits et noms des appareils Zigbee rencontrés sur le réseau de l'exocube. Ce fichier est chargé dans un dictionnaire pour annoter les sessions des captures.

Le dossier *Captures/* contient les captures réalisées. Par défaut, les scripts Builder/ viennent chercher les fichiers PCAP dans Captures/.

Le dossier *DB/* contient les databases constuites par Builder/builder.py au format CSV.

Le dossier *Learner/* contient les trainingDB au format CSV et les scripts python qui apprennent et font des prédictions. Les script Learner/ font appels aux scripts Builder.

Le dossier *Guesser/* contient les scripts python qui font des prédictions à partir de captures. Les scripts Guesser/ font appel aux scripts Learner/ pour réentraîner les algorithmes et faire des prédictions immédiatement après.

Le fichier *main.py* contient un parser (argparse). Il lit les commandes mentionnées plus haut et lance les scripts correspondants dans les dossiers *Learner/ Builder/ Guesser/*.

Les scripts python helpers_*.py contiennent des fonctions auxiliaires appelées par les scripts principaux. Le fichier *capture_script.sh* contient un exemple de script bash pour lancer une capture sur l'interface loopback de la Raspberry Pi (48h de captures, 30 minutes par capture).

3.4.3 Capture avec Dumpcap

Pour entrainer notre algorithme, il faut constituer une grande base de données à partir de captures longues, réalisés à différents moments de la journée.

Problèmes : **Les captures avec Wireshark**

- Wireshark limite le temps de capture, en s'éteignant lui-même après 1 heure d'activité.
- Les fonctionnalités de filtrage sont limitées.
- L'interface graphique de Wireshark est mal optimisée pour la Raspberry Pi.

Solution : **Utiliser Dumpcap ou Tshark**

Dumpcap est un module qui permet de lancer des captures Wireshark directement depuis l'invité de commandes. Il existe de nombreux paramètres, et les fonctionnalités sont bien documentées. Voici la commande que nous avons lancée pour réaliser nos captures : udo dumpcap -i lo -b duration:3600 -a duration:24*3600 -w "/home/pi/file$_c$*apture/mytrace.pcap*" $-$ *q*
-i lo est la commande pour capture sur l'interface loopback de la machine
-b duration :3600 commande pour des captures de 1h chacune
-a duration :24*3600 commande pour stopper les captures au bout de 24h
-w renseigne le path du dossier où sont enregistrées les captures
-q limite la verbosité du programme
Par défaut, Dumpcap exporte les captures au format PCAP.

3.4.4 Extraction de données avec Scapy et construction d'une database avec Pandas

Voici le code qui construit une DB au format CSV à partir d'un fichier PCAP:

```
1  def build(path, dest):
2
3      name = path.split(".")[0].split("/")[-1].split("\\")[-1]
4
5      packets = loadPcap(path)
6      httpPackets = filterHTTPPackets(packets)
7      tagsDictionnary = createTagsDictionnary()
8      packetsData = packetsDataExtractor(httpPackets, tagsDictionnary)
9
10     packetsData2 = mergeUpDownSessions(packetsData)
11     sessionsData = sessionsDataExtractor(packetsData2)
12
13     db = buildDataBase(sessionsData)
14     db.to_csv(f"{dest}{name}-iot-db.csv", index=False)
15
16     return True
```

Fonctions importées de *Builder/helpers_pcap.py* :

- loadPcap(path) : utilise la fonction scapy.rdpcap() pour renvoyer un objet scapy contenant les paquets de la capture. Cet objet est itérable (on peut parcourir les paquets à l'aide d'une boucle for).
- filterHTTPPackets() : itère sur un objet scapy et trie les paquets qui n'ont pas de couche IP. Renvoie tous les paquets IP qui ont pour port source ou destination le port 80.
- createTagsDictionnary() : ouvre le fichier *sensors_tags.txt* et renvoie le dictionnaire suivant : key = value , key=(sensor_name, sensor_short_ID), value=[sensor_Zigbee_ID]
- findTags() : prend en paramètre une chaine de charactère et le dictionnaire contenant les tags des sensors. S'il reconnait l'ID du sensor dans la chaine de charactère, il renvoie la clef du dictionnaire correspondante. Sinon il renvoie None.
- mergeUpDownSessions() : prend en paramètre un dataframe construit par packetsDataExtractor() et modifie les id de sessions de la façon suivante : Client > Serveur et Serveur < Client deviennent en Client<>Serveur

Fonctions importées de *Builder/helpers_extractor.py* :

- packetsDataExtractor() : itère sur un objet scapy et renvoie un pandas.DataFrame. Chaque ligne du DataFrame correspond à un paquet de l'objet scapy. Le sensorID est affecté au paquet si un tag est reconnu dans son content (donc s'il s'agit d'un POST HTTP) ; pour cela, on fait appel à findTags() définie plus haut.

	Session	Bytes	Time	PortSrc	PortDest	SensorID
0	TCP 127.0.0.1:44444 > 127.0.0.1:80	60	2022-05-27 13:23:11.8902:00	44444	80	nan
1	TCP 127.0.0.1:44444 > 127.0.0.1:80	52	2022-05-27 13:23:11.8904:00	44444	80	nan
2	TCP 127.0.0.1:44444 > 127.0.0.1:80	355	2022-05-27 13:23:11.8905:00	44444	80	nan
3	TCP 127.0.0.1:44444 > 127.0.0.1:80	243	2022-05-27 13:23:11.8907:00	44444	80	3.0
4	TCP 127.0.0.1:44444 > 127.0.0.1:80	52	2022-05-27 13:23:11.9256:00	44444	80	nan
...	...	...	...	...	...	...
8227	TCP 127.0.0.1:80 > 127.0.0.1:46220	60	2022-05-27 13:53:08.6854:00	80	46220	nan
8228	TCP 127.0.0.1:80 > 127.0.0.1:46220	52	2022-05-27 13:53:08.6857:00	80	46220	nan
8229	TCP 127.0.0.1:80 > 127.0.0.1:46220	52	2022-05-27 13:53:08.6885:00	80	46220	nan
8230	TCP 127.0.0.1:80 > 127.0.0.1:46220	992	2022-05-27 13:53:08.7242:00	80	46220	nan
8231	TCP 127.0.0.1:80 > 127.0.0.1:46220	52	2022-05-27 13:53:08.7276:00	80	46220	nan

- sessionsDataExtractor() : itère sur un dataframe retourné par packetsDataExtractor(). Le script regroupe les lignes (paquets) qui ont le même SessionID. Il crée de nouvelles données pour chaque session, et renvoie un sessionsDataFrame (chaque ligne représente une session).

	Session	Bytes	NumberOfPackets	StartTime	EndTime	DonglePort	SensorID
0	TCP 127.0.0.1:44444 <> 127.0.0.1:80	2074	12	2022-05-27 13:23:11.890259743+02:00	2022-05-27 13:23:11.928690910+02:00	44444	3.0
1	TCP 127.0.0.1:44448 <> 127.0.0.1:80	2074	12	2022-05-27 13:23:15.385618448+02:00	2022-05-27 13:23:15.430405378+02:00	44448	3.0
2	TCP 127.0.0.1:44450 <> 127.0.0.1:80	2074	12	2022-05-27 13:23:18.879756451+02:00	2022-05-27 13:23:18.907689810+02:00	44450	3.0
3	TCP 127.0.0.1:44452 <> 127.0.0.1:80	2272	12	2022-05-27 13:23:20.744548798+02:00	2022-05-27 13:23:20.758941650+02:00	44452	3.0
4	TCP 127.0.0.1:44454 <> 127.0.0.1:80	2074	12	2022-05-27 13:23:22.382482767+02:00	2022-05-27 13:23:22.426654816+02:00	44454	3.0
..	...	...	...	...	...	...	...
681	TCP 127.0.0.1:46210 <> 127.0.0.1:80	2120	12	2022-05-27 13:53:01.293882132+02:00	2022-05-27 13:53:01.311942339+02:00	46210	7.0
682	TCP 127.0.0.1:46214 <> 127.0.0.1:80	2074	12	2022-05-27 13:53:02.281463623+02:00	2022-05-27 13:53:02.305305958+02:00	46214	3.0
683	TCP 127.0.0.1:46216 <> 127.0.0.1:80	2277	12	2022-05-27 13:53:02.489340544+02:00	2022-05-27 13:53:02.498610973+02:00	46216	6.0
684	TCP 127.0.0.1:46218 <> 127.0.0.1:80	2074	12	2022-05-27 13:53:05.781076431+02:00	2022-05-27 13:53:05.823468447+02:00	46218	3.0
685	TCP 127.0.0.1:46220 <> 127.0.0.1:80	2074	12	2022-05-27 13:53:08.685385227+02:00	2022-05-27 13:53:08.727691650+02:00	46220	3.0

[686 rows x 7 columns]

  - StartTime : Timestamp du premier paquet de la session.
  - EndTime : Timestamp du dernier paquet de la session.
  - Bytes : Somme des tailles des paquets de la session.
  - DonglePort : port ouvert par le dongle (le numéro de port différent de 80).
  - SensorID : SensorID repéré dans le contenu du paquet POST HTPP de la session.

Fonction importée de *helpers_db.py* :

- buildDataBase() : construit la database qui sera envoyée à l'algorithme pour apprentissage. Prend en paramètre un dataframe retourné par sessionsDataExtractor(). Regroupe les sessions par SensorID. Renvoie une ligne par sensorID détecté dans la capture :

	SensorID	SessionDurationMean	SessionBytesMean	SleepTimeMean	SleepTimeVariance
0	3.0	0.038450	2088	3.249177	0.814016
1	2.0	0.020449	2259	24.193174	687.146528
2	6.0	0.021005	2273	62.948588	75.581953
3	4.0	0.014486	2213	301.260623	315430.230507
4	1.0	0.020890	2273	1800.000000	0.000000
5	7.0	0.018958	2188	77.135435	17346.853534
6	5.0	0.041063	2191	0.243532	0.000000

  - SessionDurationMean : le temps de chaque session est calculé à partir des StartTime et EndTime. On calcule ensuite la moyenne de toutes les sessions émises par un même SensorID.
  - SessionBytesMean : taille moyenne des sessions émises par un même SensorID.
  - SleepTimeMean et SleepTimeVariance : le SleepTime est le temps que met le sensor avant de renvoyer une session. Il se calcule de la façon suivant : StartTime(n+1) – Endtime(n) , n l'indice de la session.

3.4.5 Fonction Builder.build.merge()

Fusionne les databases construites à partir des captures en un unique fichier CSV. Fait appel aux méthode de traitement des fichiers CSV de la bilbiothèque pandas. Constitue la trainingDB.py.

3.4.6 Modèles d'Apprentissage :

Le code présenté ci-dessous vise à construire un algorithme d'identification des objets à partir d'une capture Wireshark. On rappelle que l'on se base sur les sessions et que l'on en extrait des caractéristiques.

Apprentissage :

On utilise pour prédire nos données les algorithmes d'apprentissage XGBoost et RandomForest.
L'utilisation de ces deux algorithmes semble pertinente dans la mesure où nous recherchons des algorithmes de classification de faible complexité. De plus, l'utilisation du RandomForest et du XGBoost semble courante dans le machine learning pour la reconnaissance des objets (cf. état de l'art, partie algorithmes).
Le code se situe dans Le dossier Learner, et est constitué d'une fonction générale et d'un helper. Le helper est constitué de différentes fonctions d'utilité distincte. La fonction générale, située dans le fichier.py, agence les différentes fonctions construites dans le helper. Nous nous concentreronssurtout sur le helper.
Le helper est agencé de la manière suivante : construction des bases de données, normalisation des données, une partie applicative XGBoost, une autre partie applicative RandomForest et enfin, les résultats des algorithmes.

Les Bibliothèques

On utilise les bibliothèques suivantes :

- Pandas pour manipuler nos bases de données
- Sklearn pour l'apprentissage statistique :
  - Le module RandomForestClassifier pour faire appel à l'algorithme de classification du même nom et aux fonctions du module.

- Les modules StandardScaler et LabelEncoder pour standardiser nos données train et test
- La fonction train_test_split pour créer nos différentes bases de données nécessaires à l'apprentissage.
- Les fonctions accuracy_score et confusion_matrix pour afficher respectivement le taux de prédiction et la matrice de confusion
- XGBoost pour faire appel à l'algorithme de classification du même nom et aux fonctions du module.

Les bases de données

Pour apprendre, il nous faut une base de données d'apprentissage et une base de données de test, issues de notre base de données d'origine.
Pour cela, il faut d'abord convertir notre ficher en une base de données exploitable. La conversion de notre base de données se fait avec la fonction import_csv qui prend en entrée le chemin pour accéder au fichier.

```
1 def import_csv(path):
2     return(pd.read_csv(path))
```

Ensuite, il s'agit ensuite de créer nos bases train and test. On sépare nos identifiants (target) de nos caractéristiques (features) et on crée, à l'aide de la fonction train_test_split les bases de données pour l'apprentissage. Elle prend en entrée la base de données caractéristiques, la base de données identifiants, une taille pour la base de données apprentissage et renvoie nos bases de données pour l'apprentissage, x_train et y_train, et nos bases de données pour le test, x_test et y_test. On peut décider, à l'aide de trainPercentage, de la taille des bases de données de test. Notre décision a été de l'ordre de 20% de la taille de la base de données originale.

```
1 def split(df, trainPercentage):
2     features = df.loc[:, df.columns != 'SensorID']
3     target = df['SensorID']
4     x_train, x_test, y_train, y_test = train_test_split(
5         features, target, train_size=trainPercentage, random_state=0)
6     return(x_train, y_train, x_test, y_test)
```

L'encodage

La normalisation et la standardisation des données est nécessaire au fonctionnement des algorithmes. La fonction features_encoder, via le module StandardScaler, a pour objectif de rapprocher la distribution des données le plus possible d'une loi normale centrée réduite $\mathcal{N}(0,1)$, sans quoi les estimations pourraient potentiellement être moins bonnes. Elle retourne les bases de données x_train_encoded et x_test_encoded qui seront ainsi utilisées par les algorithmes d'apprentissage. La fonction label_encoder, via le module LabelEncoder, nous permet de standardiser nos valeurs. Si on possède comme labels plusieurs chaînes de caractères, les fonctions LabelEncoder.fit et LabelEncoder.transform() associent à chaque chaîne de caractère un nombre. Laisser des chaînes de caractères comme label ne pose donc a priori pas de problème. C'est aussi le cas pour StandardScaler, mais il est plus rare qu'une des caractéristiques soit une chaîne de caractère.

L'algorithme XGBoost

On souhaite désormais mettre en place l'algorithme XGBoost et estimer ses performances. On fait d'abord appel au module XGBClassifier de la bibliothèque XGBoost. La fonction XGBClassifier.fit (model.fit) prend en entrée les bases de données d'apprentissage et c'est là que se concrétise l'apprentissage. La fonction XGBClassifier.predict (model.predict) est la fonction de prédiction. Elle prend en entrée la base de données de features normalisée (sans label, et après encodage) et a pour vecteur image y_predict. C'est le vecteur des labels prédits par la fonction de prédiction. Le vecteur y_predict est à comparer avec y_test, vecteur contenant les véritables labels associés aux données de x_test_encoded.

```
def model_xgboost(x_train_encoded, y_train_encoded, x_test_encoded, guess=False,
x_guess_encoded=None):
    model = XGBClassifier()
    model.fit(x_train_encoded, y_train_encoded)
    y_predict = model.predict(x_test_encoded)
    if guess:
        y_guess = model.predict(x_guess_encoded)
        proba_guess = model.predict_proba(x_guess_encoded)
    else:
        y_guess = None
        proba_guess = None

    return(y_predict, y_guess, proba_guess)
```

L'algorithme RandomForest

De la même manière, on souhaite estimer les performances du RandomForest. On fait appel au module RandomForestClassifier. Ce dernier prend de nombreuses entrées, en particulier celles que l'on voit dans le code (les autres sont paramétrées par défaut), n le nombre d'arbres que l'on souhaite avoir dans notre forêt, criterion la fonction utilisée pour la mesure d'impureté, et random_state, le caractère aléatoire du bootstrapping (cf. état de l'art). De la même manière que précédemment, la sortie des y_predict, le vecteur image de la fonction de prédiction par x_test_encoded.

```
def model_randomforest(x_train_encoded, y_train_encoded, x_test_encoded,
guess=False, x_guess_encoded=None):
    model = RandomForestClassifier(
        n_estimators=100, criterion='entropy', random_state=0)
    model.fit(x_train_encoded, y_train_encoded)
    y_predict = model.predict(x_test_encoded)
    if guess:
        y_guess = model.predict(x_guess_encoded)
        proba_guess = model.predict_proba(x_guess_encoded)
    else:
        y_guess = None
        proba_guess = None

    return(y_predict, y_guess, proba_guess)
```

Les résultats

On souhaite naturellement avoir la précision des algorithmes, c'est-à-dire à quel point l'algorithme considéré a prédit de manière exacte les objets auxquels appartiennent ces caractéristiques. La fonction accuracy_score affiche cette précision. On pourrait également considérer qu'il est correct de détecter des classes d'objet, mais c'est une considération que l'on ne prend pas en compte ici. Un paramètre qui nous semble intéressant est la matrice de confusion. Cette matrice contient sur sa diagonale le nombre d'éléments exactement prédits pour chaque capteur, mais en cas d'erreur, elle nous indique avec quel autre capteur la confusion a été faite. La fonction confusion_matrix affiche la matrice de confusion à partir du vecteur des observations et du vecteur des prédictions. À partir de cette matrice, on peut réfléchir à d'autres features à implémenter pour réduire la confusion au maximum.

```
def result(y_test_encoded, y_predict):
    R = accuracy_score(y_test_encoded, y_predict)
    mat = confusion_matrix(y_test_encoded, y_predict)
    return(f"Precis␣a␣{int(100*R)}%", mat)
```

La fonction learn et le Guesser

La fonction learn est adaptée à l'apprentissage mais également au module guess que l'on verra par la suite. Elle agence, dans l'ordre présenté ci-dessus, les fonctions du helper. Elle prend en entrée le modèle choisi, RandomForest ou XGBoost (il y a une partie guess qui n'est pas activée..), et renvoie les valeurs résultats (le score et la matrice de confusion).

Une fois l'algorithme entraîné, on souhaite prédire, à partir d'une base de données quelconque l'objet à l'origine de ces paquets. Le Guesser remplit cette fonction. Comme pour le Learner, le ficher Guesser est constitué d'une fonction générale guess et d'un helper.
Le helper est constitué d'une fonction build_guess_db dont l'objectif est de construire une base de données constituée de features à partir d'une capture avec un seul objet sur le réseau (plus le dongle). Cette base de données, qui sert essentiellement à vérifier si notre algorithme effectue de bonnes prédictions, est donc dépourvue de l'identifiant du capteur. La fonction renvoie en sortie x_guess, la base de données de features qu'on souhaite associer à l'objet d'origine par notre algorithme de prédiction.

La fonction guess utilise la partie dite guess de la fonction learn. La fonction learn, lorsqu'elle prend en entrée une base de données guess, La fonction learn renverra y_guess, la prédiction associée à la base de données guess et proba_guess. La sortie proba_guess indique à quel point l'algorithme choisi est certain de sa prédiction. La fonction guess prend en entrée une capture, utilise le helper pour construire la base de données adéquate et utilise la fonction learn pour chacun des deux algorithmes. En sortie, l'on obtient les prédictions du XGBoost et du RandomForest, ainsi que leur certitude sur la prédiction.

```
1  def guess(path):
2
3      tagsDictionnary = createTagsDictionnary()
4
5      x_guess = build_guess_db(path)
6      _, _, prediction_xgboost, proba_xgboost = learn(
7          'xgboost', guess=True, x_guess=x_guess)
8      _, _, prediction_randomforest, proba_randomforest = learn(
9          'xgboost', guess=True, x_guess=x_guess)
10
11 #voir le code fourni pour txt_xgboost et txt_randomforest
12 txt_xgboost = "Prediction␣de␣XGBoost"
13
14 txt_randomforest = "Prediction␣de␣RandomForest"
15
16  return (txt_xgboost, txt_randomforest)
```

On rappelle que la prédiction n'est réalisable que dans le cas de la présence d'un seul objet sur le réseau lors de la capture.

3.5 Résultats

Nous avons entraîné nos algorithmes à reconnaitre les objets Zigbee émetteurs d'un certain nombre de sessions TCP/IP sur 30 minutes.

Pour que les algorithmes s'entraînent sur les vraies sessions émises par les objets Zigbee, nous avons construit nos bases de données d'entrainement dans des conditions bien spécifiques et contraignantes : pour labelliser les sessions, le script build.py lit le contenu des paquets pour trouver l'ID Zigbee unique de l'objet émetteur.

Dans ces conditions d'entrainement, avec un taux apprentissage-test de 80%-20% sur les données capturées, la précision des algorithmes RandomForestClassifier et XGBoost est proche de 100%.

```
XGBoost :                          RandomForestClassifier :
Precis a 100%                      Precis a 100%
matrice de confusion :             matrice de confusion :
[[42  0  0  0  0  0  0  0]        [[42  0  0  0  0  0  0  0]
 [ 0 57  0  0  0  0  0  0]         [ 0 57  0  0  0  0  0  0]
 [ 0  0 63  0  0  0  0  0]         [ 0  0 63  0  0  0  0  0]
 [ 0  0  0 43  0  0  0  0]         [ 0  0  0 43  0  0  0  0]
 [ 0  0  0  0 34  0  0  0]         [ 0  0  0  0 34  0  0  0]
 [ 0  0  0  0  0 63  0  0]         [ 0  0  0  0  0 63  0  0]
 [ 0  0  0  0  0  0 44  0]         [ 0  0  0  0  0  0 44  0]
 [ 0  0  0  0  0  0  0 25]]        [ 0  0  0  0  0  0  0 25]]
```

En conditions réelles, les scripts n'ont pas accès au contenu des paquets. En raison du formatage induit par le dongle Conbee, la lecture des headers TCP n'est pas suffisante pour savoir si une session a été émise par tel ou tel objet. C'est pourquoi nous nous plaçons dans des conditions de test simplifiées : seul un objet Zigbee est connecté au réseau (en plus du dongle, qui émet également ses propres sessions).
Dans ces conditions, les script qui construisent les bases de données de test ne font pas la différence entre les sessions émises par le dongle et les sessions émises par le sensor Zigbee. Les données sont plus ou moins brouillées par les paquets du dongle. La précision des deux algorithmes chute drastiquement :

```
XGBoost :                          RandomForestClassifier :
Precis a 48%                       Precis a 48%
matrice de confusion :             matrice de confusion :
[[ 0  0  0  0  0 42  0  0]        [[ 0  3  0  0  0 39  0  0]
 [ 0 54  0  0  0  3  0  0]         [ 0 53  0  0  0  4  0  0]
 [ 0  0 63  0  0  0  0  0]         [ 0  0 63  0  0  0  0  0]
 [ 0  1  0  0  0 42  0  0]         [ 0  3  0  0  0 40  0  0]
 [ 0  2  0  0  0 32  0  0]         [ 0  4  0  0  0 30  0  0]
 [ 0  0  0  0  0 63  0  0]         [ 0  0  0  0  0 63  0  0]
 [ 0  2  0  0  0 42  0  0]         [ 0  4  0  0  0 40  0  0]
 [ 0 25  0  0  0  0  0  0]]        [ 0 25  0  0  0  0  0  0]]
```

On constate que certains objets sont toujours bien reconnus par les algorithmes, comme les objets 2 (smoke detector) et 3 (Prise connectée Aqara). Ce sont des objets qui émettent suffisamment de sessions sur 30 minutes pour que les données ne soient pas perturbées par les sessions du dongle. D'autres comme les objets 1 (capteur de porte SonOff) et 8 (capteur de pression Aquara) n'émettent qu'une frame toutes les 30 minutes en moyenne, ce qui explique qu'ils ne soient pas reconnus. Pour améliorer les résultats, nous avons entrainé nos algorithmes directement sur les sessions perturbées. On constate une large hausse de la précision sur les objets auparavant brouillés par dongle :

```
XGBoost :
Precis a 81%
matrice de confusion :
[[ 3  0  0  2  0  0  3  0]
 [ 0 21  0  0  0  0  0  0]
 [ 0  0 13  0  0  0  0  0]
 [ 1  0  0  2  0  0  4  0]
 [ 1  0  0  0  6  0  0  0]
 [ 0  0  0  0  0 13  1  0]
 [ 1  0  0  1  0  0  3  1]
 [ 0  0  0  0  0  0  0  7]]
```

```
RandomForestClassifier :
Precis a 78%
matrice de confusion :
[[ 3  0  0  2  0  0  3  0]
 [ 0 21  0  0  0  0  0  0]
 [ 0  0 13  0  0  0  0  0]
 [ 3  0  0  1  0  0  3  0]
 [ 0  0  0  0  7  0  0  0]
 [ 0  0  0  0  0 13  1  0]
 [ 3  0  0  2  0  0  0  1]
 [ 0  0  0  0  0  0  0  7]]
```

4 Limites et Améliorations

A la vue des résultats de nos algorithmes, on peut conclure que les informations tirées des header TCP sont suffisantes pour reconnaitre avec une très bonne précision les objets Zigbee.
Cependant, pour que cela soit réalisable en conditions de test, il faut pouvoir distinguer les sessions selon qu'elles ont été émises par tel ou tel appareil (qu'il revient ensuite de deviner). Cependant, le formatage des sessions par le dongle Conbee rend impossible cette opération si l'on s'interdit (d'un point de vue méthodologique) de regarder le contenu des paquets.

Une solution serait de trouver un dongle qui affecte à l'appareillage un port bien défini pour chaque objet (et non pas un port aléatoire pour chaque session comme le fait le Conbee).
Une autre solution serait d'utiliser des logiciels comme Burpsuit pour décoder les paquets encodés avec une clef SSL, pour trouver l'identifiant Zigbee de l'émetteur. Dans cette configuration, il serait également possible de faire de la reconnaissance sur le contenu des paquets (repérage de balises comme temperature , open , fire ...)

5 Annexe

5.1 État de l'art

5.1.1 Tableau de comparaison entre LoRa et Zigbee + Illustrations

Caractéristiques	Lora	Zigbee
Bandes de Fréquences	863 à 870 MHz, 902 à 928 MHz, 779 à 787 MHz	868 MHz, 915 MHz, 2450 MHz
Distance de couverture	2-5 Km (zones urbaines), 15 Km (zones suburbaines)	10 à 100 mètres
Consommation d'énergie	inférieur par rapport à zigbee	faible
Technique de modulation	Modulation LoRa (modulation CSS), FSK ou GFSK	Modulation BPSK, OQPSK. Utilise également la technique DSSS pour convertir les bits en puces.
Débit de données	0,3 à 22 Kbps (modulation LoRa) et 100 Kbps (en utilisant GFSK)	20 kbps (bande 868 MHz) , 40Kbps (bande 915 MHz) , 250 kbps (bande 2450 MHz)
Architecture de réseau (cf. Annexe)	Se compose d'une passerelle LoRa, de serveurs et de dispositifs finaux	Constitué d'un coordinateur, de routeurs et de dispositifs terminaux
Couche physique	Utilise le schéma de modulation mentionné ci-dessus et intègre des capacités de correction d'erreurs, ajoute un préambule à des fins de synchronisation, utilise le CRC de l'en-tête PHY ainsi que le CRC de la trame entière	Il existe deux couches physiques, à savoir 868/915 Mhz (utilise la MDPB, mise en forme des impulsions en cosinus surélevé) et 2450 MHz (utilise la MDPO, mise en forme des impulsions en demi-sinus).
Applications	utilisé comme réseau étendu	utilisé comme LR-WPAN, c'est-à-dire un réseau personnel sans fil à faible débit
Standard/Alliance	IEEE 802.15.4g, Alliance LoRa	IEEE 802.15.4 (définit PHY et MAC), Zigbee Alliance (définit les couches réseau, sécurité et application)

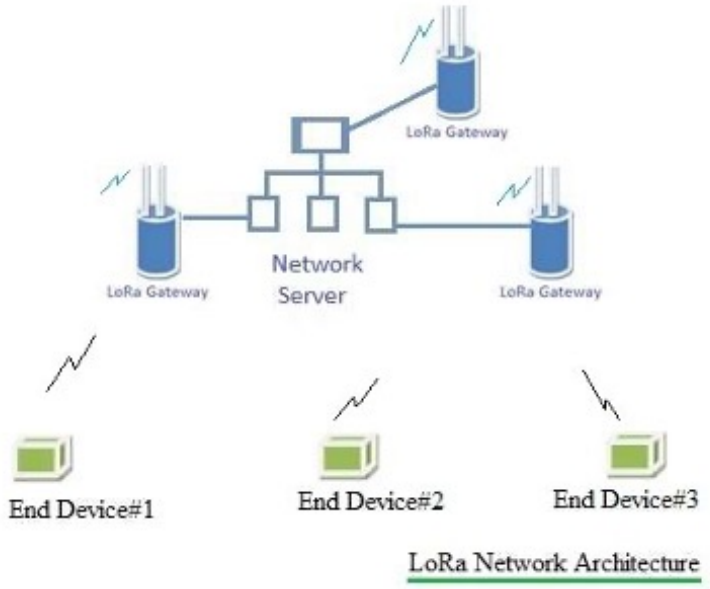

Figure 1 : Architecture Réseau LoRa

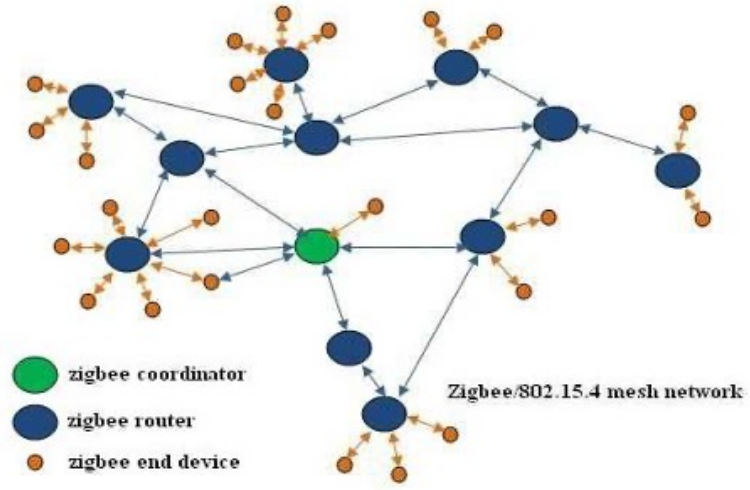

Figure 2 : Architecture Réseau Zigbee