Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN TECNOLOXÍAS DA INFORMACIÓN

Xeración dun Dataset baseado en Tráfico CoAP nun Entorno IoT

Estudante: Alberto Pardal Noya
Dirección: Víctor Manuel Carneiro Díaz

A Coruña, xuño de 2020.

A Manuel Pardal Noya

Agradecementos

Quero agradecer, en primeiro lugar, ao meu titor Víctor Carneiro Díaz, pola axuda emprestada na elaboración deste proxecto e o seu apoio en todo momento.

Tamén agradecer á miña familia todo o que fixeron e fan por min no día a día, todo o ánimo e o apoio que me deron durante todos estes anos para continuar estudando e mellorando como persoa.

A todos os meus amigos, os de toda a vida e mais os que coñecín na facultade. Grazas a todos por estar aí, por toda a axuda que me emprestaron e polos bos momentos pasados ao seu lado. Que sexan moitos mais. Aos compañeiros do grupo 2.3 orixinal, que ano tras ano pelexabamos por estar todos xuntos no mesmo grupo; a eles o meu máis sentido agradecemento e, sobre todo, expresar a sorte que tiven de atopar xente tan boa.

Tamén quero dar as gracias a todos os profesores que me ensinaron. Sobre todo aos profesores do IES San Clemente e aos da Facultade de Informática, pois eles ensináronme o meu oficio.

Por último, a toda a xente que, directa ou indirectamente, me axudou na elaboración deste proxecto; o que segue vai por eles.

Resumo

A IoT (Internet of Things) está cada vez máis presente no mundo. Ben sexa na nosa casa ou en procesos industriais, esta tecnoloxía aglutina unha gran cantidade de dispositivos que intercambian un volume de datos considerable; e que debemos protexer contra ameazas externas. Neste caso, unha monitorización manual do tráfico non é viable, será necesario expor outros mecanismos máis automáticos para poder levar a cabo esta tarefa de forma máis efectiva e eficiente. Para automatizar o proceso de análise e detección de ameazas en fluxos de datos de grande magnitude é posible usar a IA (Intelixencia Artificial), proporcionándolle os conxuntos de datos necesarios para que leve a cabo a aprendizaxe máquina; e desa forma sexa capaz de detectar, e incluso paliar, ditas anomalías. Neste proxecto, crearemos un conxunto de datos (DATASET) anotado de tráfico do protocolo CoAP (Constrained Application Protocol), utilizando software para a creación de paquetería de rede. Este protocolo, grazas á súa simplicidade e baixa sobrecarga, pode ser utilizado en dispositivos cun hardware mínimo; coma por exemplo os sensores dunha rede IoT. O que faremos neste traballo será, primeiramente, un estudo sobre as características do protocolo CoAP, para entender o seu funcionamento. Logo, unha vez escollidas as ferramentas a usar, deseñaremos unha implantación do entorno a virtualizar, e unha programación das mensaxes CoAP que se enviarán e recibirán entre as máquinas. Poremos a proba dito deseño e, unha vez estea funcionando correctamente, alteraremos a paquetería resultante empregando distintos tipos de ataques a redes IoT. Para rematar, realizaremos unha captura de tráfico final incluíndo todos os comportamentos, tanto normais como anómalos; e da que obteremos as conclusións oportunas.

Abstract

The IoT (Internet of Things) is more and more present in the world. Whether in our home or in industrial processes, this technology brings together a large number of devices that exchanges a considerable volume of data, and we must protect them against external threats. In this case, manual traffic monitoring is not possible, it will be necessary to find automatic mechanisms to carry out this task more effectively and efficiently. To automate the process of analysis and detection of threats in large data flows, it is possible to use AI (Artificial Intelligence), providing it with the necessary data to execute the machine learning, and in this way be able to detect and even mitigate said anomalies. In this project, we will create an annotated Dataset of CoAP (Constrained Application Protocol) protocol traffic, using software for creating network packets. This protocol, thanks to its simplicity and low overhead, can be used on devices with minimal hardware; such as the sensors of an IoT network. In this work,

firstly, we will make a study of the characteristics of the CoAP protocol, to understand its operation. Then, once we have chosen the tools to use, we will design an implementation of the environment to be virtualized and a programming of the CoAP messages that will be sent and received between the machines. We will test this design and, once it is working correctly, we will alter the resulting packets using different types of attacks on IoT networks. To finish, we will make a final traffic capture including all behaviors, both normal and altered, and from which we will obtain the appropriate conclusions.

Palabras chave:

- IoT
- CoAP
- Sensor
- Protocolos IoT
- Scapy
- Ataques IoT
- Ettercap

Keywords:

- IoT
- CoAP
- Sensor
- IoT protocols
- Scapy
- IoT attacks
- Ettercap

Índice Xeral

Índice de Figuras

Listings

Capítulo 1

Introdución

A Internet das Cousas (Internet of Things (IoT)) é considerada por moitos como unha das tecnoloxías máis importantes do século XXI [1]. A interconexión de obxectos cotiáns (obxectos de cociña, vehículos, termóstatos...) semella unha gran vantaxe e avance á hora de controlar e analizar o comportamento dos procesos levados a cabo por todos estes utensilios.

Este estado de conexión con obxectos e utensilios ábrenos as portas a algo máis grande, a un aumento da automatización das máquinas aos cambios que se poidan producir no seu ciclo de traballo. O fin é o cabo, o intercambio de datos que leven a cabo os dispositivos IoT poden realizarse cunha intervención humana mínima, baseada nunha programación a priori dos distintos comportamentos posibles. Esta situación pois, permite mirar inda máis lonxe; ao procesos de ámbito industrial e automatización de grandes fábricas ou superficies (Fabricación Intelixente, Cidades Intelixentes...).

Cada vez hai mais investigación en IoT e adáptanse mais máquinas e utensilios a esta tecnoloxía. Un círculo de avance e de investigación que comezou fai relativamente pouco, e que segue medrando e espertando moito interese polas potenciais vantaxes económicas, produtivas de control que ofrece.

Nestas redes IoT, o intercambio de información e fluxo de datos soe ser considerable, e os recursos de cómputo dos dispositivos moi limitado; ademais, hai que ter en conta como se dixo anteriormente, que se perde en grande medida o control nas comunicacións dos dispositivos, pois utilizarán unha comunicación automática e desasistida. Por tanto, necesitase de mecanismos e protocolos especializados para mellorar a eficiencia coa que se procesan e envían os datos. Nacen así os protocolos de comunicación IoT. Estes son moi variados e divídense fundamentalmente en domésticos ou industriais [2], inda que sobre isto falaremos mais no seguinte capítulo do traballo. Baste dicir que existen diferenzas entre eles, tanto en estrutura como en tratamento dos datos transportados, que provocan que uns protocolos estean máis orientados a certas situacións e escenarios ca outros (ex. maior ou menor automatización da

comunicación).

Un dos dispositivos máis básicos e importantes das redes IoT é o sensor. Polo xeral ten un hardware mínimo, pois non necesita capacidade de cómputo. Sen embargo, a evolución fai que cada vez se busque simplicidade e sinxeleza nestes dispositivos (sensores mais baratos, mais pequenos...), inda que aumentando na medida do posible a súa precisión. Por todo isto, e dado que os sensores teñen que comunicarse con outras máquinas, xurde a necesidade de protocolos de rede máis lixeiros que os comunmente usados, diminuíndo así os recursos necesarios para a comunicación entre cliente-sensor.

Estes protocolos lixeiros diferéncianse entre si en moitas formas, e inda que se fará unha pequena descrición deles, neste traballo centrarémonos no protocolo CoAP (*Constrained Application Protocol*). Este é unha versión lixeira do protocolo HTTP. CoAP foi deseñado pola IETF (Internet Engineering Task Force) para ser utilizado por dispositivos de recursos limitados e, desta maneira, permitirlles usar un subconxunto dos métodos HTTP [3]. As mensaxes son enviadas sobre o protocolo UDP, mentres HTTP usa TCP. Da mesma forma que HTTP, utiliza a mesma arquitectura cliente-servidor, proporcionando iteracións orientadas a recursos (ex. URLs e URIs), tamén fai uso do mesmo tipo de peticións (GET, POST,...) e dos mesmos códigos de resposta (201 Created, 404 Not Found,...). Malia toda a relación que ten o protocolo CoAP co HTTP, non debemos velo como un protocolo HTTP comprimido, senón optimizado para dispositivos limitados.

Como ven dixemos anteriormente, as redes IoT poden chegar a ter un trafico moi voluminoso. Coma todas as redes informáticas, non está exenta de interferencias de terceiros (ataques a redes de datos), nin de erros hardware e software, que alteran o bo funcionamento dos sistemas. A cantidade de alteracións (ben sexan orixinadas externamente como internas) sufridas por estas redes tamén é moi grande; e, debido á xa mencionada capacidade limitada dos dispositivos, é máis difícil implantar mecanismos para paliar os seus efectos. Por tanto, os métodos tradicionais que manteñen a seguridade e a estabilidade nunha rede normal de computadores, non son tan válidos e eficaces neste caso.

Para paliar esta inseguridade intrínseca das redes IoT é necesaria a creación de novos mecanismos que supervisen a rede e poidan actuar ante calquera ameaza ou erro que poida xurdir. Por este motivo, estanse poñendo en práctica novos métodos como o uso da IA (Intelixencia Artificial) para detectar automaticamente as distintas situacións onde se estean producindo interferencias de terceiros (ataques) ou comportamentos anómalos nunha rede ou nun dispositivo IoT; e, dependendo do tipo da anomalía que sexa, tomar as accións oportunas.

Esta IA aplicada a redes de datos ou, máis concretamente neste caso, a redes IoT; utiliza algoritmos de aprendizaxe máquina que toman como exemplo distintos *Datasets* de varios protocolos IoT para ver e analizar os distintos escenarios que se poidan producir, tanto os

normais coma os que comprometen a seguridade e o bo funcionamento dos dispositivos da rede. Por tanto, é necesaria a creación dun *Dataset* anotado, que exemplifique o uso do protocolo CoAP nunha situación normal e nun entorno corporativo virtualizado; no que se mostren tamén algúns ataques e anomalías para que a IA poida detectar estas situacións e aprendelas.

Neste proxecto trataremos de amosar o protocolo CoAP dende unha perspectiva analítica e funcional. Ao longo do proceso desentrañaremos as principais características e estrutura do protocolo, ademais de recrear un entorno corporativo virtualizado no cal se procederá a realizar probas e capturas de tráfico con fin de xerar un *Dataset*. Tamén, tal e como se dixo anteriormente, exemplificaremos situacións nas que o fluxo de datos CoAP se verá comprometido por terceiras partes e conseguir así un *Dataset* anotado completo para que unha IA poida aprender a detectar situacións similares.

1.1 Obxectivos

Os principais obxectivos e motivación deste traballo é a obtención dun *Dataset* anotado do protocolo CoAP. Mais concretamente:

- Estudo do protocolo CoAP co obxectivo de entender as súas características e funcións principais.
- Mellorar e incrementar as competencias do título mediante a adquisición de coñecementos sobre redes IoT e as súas ameazas, así como a aprendizaxe de uso das diferentes ferramentas e linguaxes escollidas para a elaboración deste traballo.
- Deseño e implantación simulada dun entorno real, e programación das mensaxes de petición e resposta CoAP; de cada unha das máquinas, tanto da que actúa como cliente como das que actúan como servidores.
- Experimentación e probas sobre a implantación do entorno simulado, realización dunha captura do tráfico resultante empregando as ferramentas axeitadas.
- Alteración do comportamento por defecto da paquetería CoAP para introducir diferentes anomalías ou ”ataques” contra o fluxo de datos.
- Estudo da captura completa final, obtención de conclusións e creación de gráficas mediante análise da paquetería resultante con software de estatística.

Para levar a cabo esta abstracción apoiarémonos nunha serie de ferramentas debidamente descritas no capítulo 3. A realización de ditos obxectivos culminará coa obtención dun *Dataset* anotado, que será unha captura de trafico da nosa rede IoT virtualizada, e na que poidamos

observar e analizar os fluxos de datos do protocolo CoAP e os cambios que produciron os ataques nos diálogos; en contraste cunha comunicación normal.

Capítulo 2

Estado do Arte

2.1 IoT e Sensores

Internet of Things, ou Internet das Cousas, é, basicamente, un conxunto de dispositivos de computación interconectados, máquinas, obxectos, (incluso persoas) que teñen identificadores únicos e a capacidade de transferir datos a través dunha rede; sen requirir, a priori, de interaccións humanas [4].

Un dos dispositivos fundamentais das redes IoT é o sensor. Polo xeral soen ter unha electrónica básica, dado que a súa utilidade radica en medir as distintas variables físicas ou químicas dun ambiente determinado [5]. O hardware básico dun sensor é [6]:

- **Microcontrolador**: É a unidade de procesamento do dispositivo. Normalmente estes procesadores operan cunha frecuencia de reloxo baixa e contan cunha arquitectura de entre 4 e 32 bits. Contan tamén con unha memoria de só lectura (ROM) e outra de acceso aleatorio (RAM), cunha capacidade na orde das decenas ou centenas, de quilobytes (KB). Poden operar nun modo activo ou nun de hibernación, co fin de reducir o consumo de enerxía.

- **Memoria**: Componse dun chip de almacenamento interno cunha capacidade maior que a memoria interna (RAM e ROM). A súa finalidade é almacenar datos temporais que poden provir de distintas fontes (datos do sensor ou da rede). A súa capacidade pode variar entre quilobites (KB) e xigabytes (GB).

Á hora de deseñar e construír un sensor, sempre se busca reducir. Reducir o tamaño para poder colocalo en lugares mais pequenos e estratéxicos, reducir o hardware para que sexa mais accesible en termos económicos e, sobre todo, que consuma menos enerxía... Non é un dispositivo no que se busque capacidade de cómputo, ten unha función básica e simple. Sen embargo neste caso, sinxeleza non é sinónimo de facilidade. Reducir sen perder fiabilidade e

precisión é complicado en termos de deseño hardware, sobre todo se se pretende que interactúe con outras máquinas. No relativo á comunicación e fluxos de datos, non se podería seguir empregando o mesmo tipo de protocolos; pois necesitaría de recursos adicionais para casos como fragmentación de paquetes grandes, nos que o gasto de recursos sería moi custoso para un dispositivo deste tipo.

Cada vez máis redes IoT acceden aos diferentes sensores utilizando redes TCP/IP [7], debido á compatibilidade coa inmensa maioría das máquinas existentes hoxe en día. Sen embargo, isto complica o deseño dos sensores volvéndoos mais complexos, dado que terán que implantar o modelo TCP/IP. Este problema trae consigo a necesidade de protocolos de comunicación máis simples e adaptados a este modelo. É aí onde se comezan a crear as redes LLN (Low Power and Lossy Networks) [8] e os protocolos lixeiros para dispositivos embebidos con potencia, memoria e capacidade de procesamento limitadas.

2.2 Protocolos nas Redes IoT

Como dixemos anteriormente, os sensores teñen implantado o modelo TCP/IP. Este derivou do modelo OSI, que se dividía en 7 capas de comunicación (véxase imaxe comparativa 2.1). O modelo TCP/IP é moito mais flexible e escalable [9] (o modelo OSI resultou meramente teórico), dividíndose en 4 capas cos seus respectivos protocolos. A comunicación entre capas segue o mesmo procedemento que o modelo OSI, os datos son encapsulados antes de pasar á capa seguinte (o protocolo usado na capa engade a súa cabeceira ao paquete).

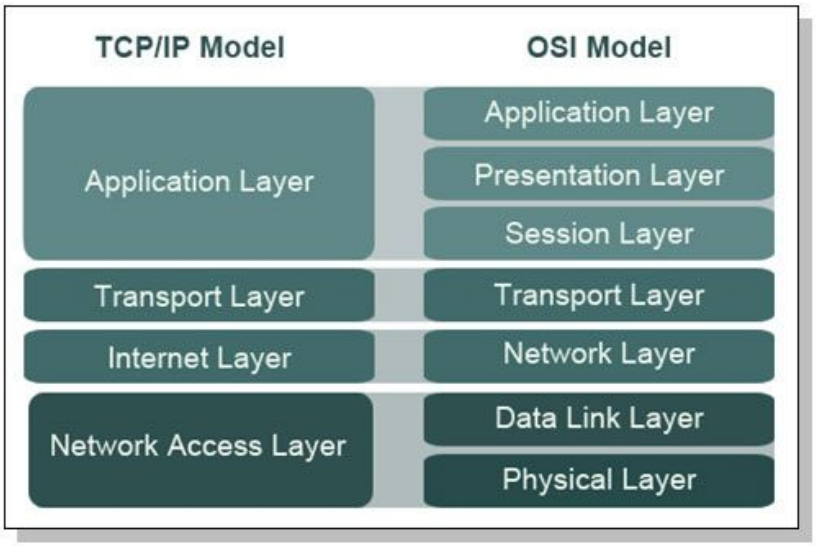


Figura 2.1: Modelo OSI vs Modelo TCP/IP

As capas do modelo TCP/IP son:

- **Capa Física ou de Enlace:** É a capa mais baixa do modelo TCP/IP. Está orientada a comunicacións punto a punto, polo que non entende de enrutamento. Normalmente os protocolos desta capa especifican as características físicas da rede na que se encontra o dispositivo. Tamén nesta capa se leva a cabo o control de acceso ao medio, ben sexa nunha rede cableada (LAN) mediante a dirección MAC; ou unha rede sen fíos (WLAN) usando algún método existente (ex. CSMA). Os protocolos mais usados desta capa en IoT son [10]:

  - IEEE 802.15.4 : Redefine a capa física e o control de acceso ao medio para redes inalámbricas con baixa complexidade, baixo consumo de enerxía e baixa taxa de bits.
  - IEEE 802.11ah : tamén chamado Low Power Wi-Fi. Estende a tecnoloxía Wi-Fi á banda dos 900 MHz permitindo unha conectividade de baixo consumo necesaria para a creación de grupos de dispositivos (ex. sensores) que cooperan para compartir sinais.
  - BLE : Bluetooth Low Energy, usa unha banda de transmisión corta con un consumo mínimo de enerxía (0.01 - 10 mW).
  - PCL: Power Line Communications. Utiliza os cables da rede eléctrica en uso para transportar datos.

- **Capa de Rede:** Unha rede IoT, en teoría, podería ter millóns de dispositivos conectados. Esta situación pon de manifesto a gran demanda que podería haber de direccións IP. O protocolo IPv4 non sería capaz de ofertar o volume hipotético necesario para estas redes IoT, sería necesario o uso de tecnoloxías como NAT que complicarían o funcionamento e o despregue das redes [11]. Por tanto a solución máis óptima sería IPv6, que nos permitiría asignar 3.4×10^{38} direccións IP, fronte as 2^{32} de IPv4. Mais esta non é única razón pola que usar IPv6, debemos contar con que, por definición, este protocolo integra IPSec para aumentar a seguridade e prove direccións de enlace local e global (Multicast IPv6).

  Certo é que pode haber casos ou redes onde utilizar IPv4 sexa o mais axeitado. Inda existen numerosas infraestruturas informáticas que non se actualizaron a IPv6 por diversos motivos (custe de implantación, falta de adaptación...). O comentado anteriormente sobre o número de dispositivos que unha rede IoT podería ter, é teórico. Por tanto debe ser tratado como tal e usar o protocolo que mais conveña para a nosa infraestrutura.

- **Capa de Transporte:** Dado que os escenarios IoT deben ser lixeiros e ter a menor carga computacional posible, a mellor opción sería usar o protocolo UDP para esta capa [12]. É certo que non ten moitas das características do seu antagonista, o protocolo TCP,

como por exemplo o control da conxestión e da retransmisión, a orde de paquetes... Sen embargo, ao non requirir que non haxa fragmentación de paquetes nin confirmación de chegada á máquina de destino dun paquete enviado dende un sensor, o protocolo UDP semella o mais óptimo para usar. Inda así, non hai un protocolo predefinido, nin pola industria nin pola comunidade académica, para a capa de rede; como os que hai para outras capas.

- **Capa de Aplicación:** Esta é a capa mais alta do modelo. Os protocolos desta capa deben de ter e conta todos os requirimentos e condicións derivados das capas máis baixas [13]. Por exemplo, HTTP e TCP non son protocolos axeitados debido ao altos gastos computacionais que provocan Por un lado, HTTP é un protocolo baseado en texto e moi detallado. Por outro lado TCP é un protocolo de transporte orientado á conexión e que require uns axustes e un mantemento das conexións abertas. Como podemos ver, non son gastos accesibles dende unha perspectiva computacional e comunicativa. Por esta razón, os protocolos desta capa deben ser deseñados tendo en conta todos os aspectos dos dispositivos e das capas máis baixas.

  Así mesmo, nesta capa itroduciranse os datos concretos do dispositivo a enviar, por tanto a maneira en que se representen eses datos tamén terá impacto nas capas inferiores á hora de realizar o envío. A fragmentación incrementa a probabilidade de que un paquete a entrega do paquete sexa errónea, por tanto é aconsellable enviar datos de pequeno tamaño para evitar a fragmentación e, por conseguinte, o erro na entrega.

  Os protocolos da capa de aplicación máis importantes en IoT son [14]:

  - *MQTT : Message Queuing Telemetry Transport.* É un protocolo do tipo publicación-subscrición lixeiro. Proporciona comunicacións M2M (máquina a máquina) facendo uso dun reducido ancho de banda e consumición de enerxía. Está baseado na pila do modelo TCP/IP e soporta QoS (calidade de servizo). Pode implantarse para que use SSL/TLS e autenticación por usuario e contrasinal ou certificado, e así aumentar a seguridade das comunicacións.
  - *AMQP : Advanced Message Queuing Protocol.* Protocolo estándar aberto de publicación-subscrición que soporta comunicacións M2M [2]. Deseñado para garantir a confiabilidade a a interoperabilidade, tamén soporta QoS. Depende dunha fiable e eficiente cola de mensaxes [15]. Está pensado para aplicacións corporativas e redes de baixa latencia. Non resulta o máis adecuado para redes IoT con dispositivos de recursos limitados.
  - *XMPP : Extensible Messaging and Presence Protocol.* É un protocolo aberto, escalable e descentralizado baseado en XML, orixinalmente deseñado para mensaxería instantánea [2]. Pode se configurado para usar sobre o modelo publicación-

subscrición ou solicitude-resposta. Non soporta QoS pero si TLS. Usa XML para codificar os datos nas comunicacións, o que trae consigo un incremento da carga computacional [15].

- *DDS : Data Distribution Service.* Protocolo baseado no modelo publicación-subscrición orientado a sistemas en tempo real. É un estándar aberto e descentralizado. As máquinas que usan este mecanismo comunícanse a través de mensaxes UDP multidifusión (*multicast*), inda que tamén se pode implantar sobre TCP. Soporta QoS e pode usar TLS e DTLS para a seguridade das comunicacións.
- *MODBUS :* É un protocolo de tipo solicitude-resposta introducido no ano 1979 para sistemas de control de procesos [16]. Utiliza o protocolo TCP para o intercambio de mensaxes entre cliente-servidor. A súa característica principal é a estrutura da mensaxe, xa que define a súa PDU (*Protocol Data Unit*) independentemente das do resto das capas inferiores [17]. Por tanto, a PDU de MODBUS está composta por un Código de Función (*Function Code*), seguida polo campo Data onde se introducen os datos a enviar. Soporta TLS e autenticación mediante certificado [18].
- *CoAP :* dado que este traballo se centra en dito protocolo, analizarémolo con mais detalle no seguinte punto 2.3.

2.3 Protocolo CoAP

Constrained Application Protocol é un protocolo de tipo solicitude-resposta deseñado polo IETF que ten como obxectivo proporcionar un servizo REST (Transferencia de Estado Representacional) para aplicacións orientadas a recursos en dispositivos e redes limitadas [15]. Este protocolo está baseado na web e no intercambio de recursos da maneira que o realiza o protocolo HTTP, pero deseñado especialmente para os requirimentos e limitacións dun entorno con capacidades limitadas. CoAP non busca crear un protocolo HTTP comprimido, senón que define un subconxunto de operacións REST similares as de HTTP pero optimizadas para aplicacións M2M (máquina a máquina) [19].

O modelo de interacción de CoAP garda parecidos co modelo cliente-servidor de HTTP. Unha solicitude (*request*) CoAP é equivalente a unha solicitude HTTP que é enviada por un cliente para solicitar unha acción (usando un *Method Code*) nun recurso (identificado por unha URI) nun servidor. O servidor envía a resposta cun Código de Resposta (*Response Code*), esta resposta incluirá a representación do recurso solicitado [20].

A diferenza de HTTP, CoAP utiliza o protocolo de transporte UDP e mensaxes asíncronas. Tamén se pode configurar para que use o modelo de publicación-subscrición, no que as mensaxes poden ser enviadas utilizando comunicación *multicast* [15]. CoAP tamén está deseñado

para manter uns gastos de recursos baixos e limitar a necesidade de fragmentación. A IANA (Internet Assigned Numbers Authority) asignou o porto 5683 ao servizo CoAP [21].

2.3.1 Capas Lóxicas do protocolo CoAP

O protocolo CoAP pode dividirse de forma lóxica en 2 capas [20], como vemos na imaxe 2.2. Unha capa é a que leva a xestión dos mensaxes e da paquetería UDP ben sexa unicast ou multicast. A outra capa que xestiona as iteracións solicitude-resposta usando Códigos de Método e Códigos de Resposta.

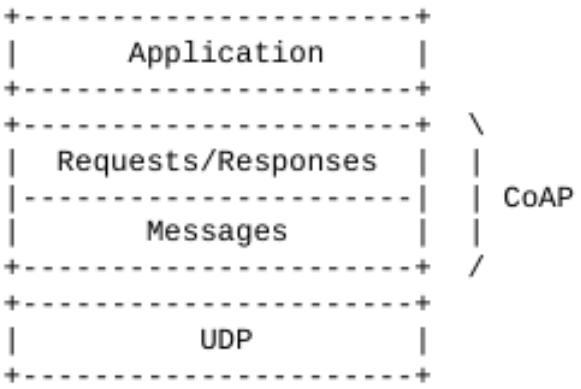


Figura 2.2: División en dúas capas lóxicas do protocolo CoAP

Capa de Mensaxería

Encárgase do intercambio de paquetería UDP entre as máquinas finais [22]. Cada mensaxe ten asignado un tipo, estes poden ser:

- *Confirmable (CON)*: proporciona fiabilidade e confirmación de recibimento. Este tipo de mensaxe obriga á máquina que o recibe a contestar cun acuse de recibo (mensaxe ACK) co mesmo identificador de mensaxe (*Message ID*) que o paquete CON. Se a máquina non é capaz de procesar a mensaxe tipo CON, enviará un RST (*Reset*) en vez dun ACK.

- *Non-confirmable (NON)*: empregado cando non é necesario a confirmación de recibo por parte da máquina final, por tanto non require de resposta ACK. Se non pode procesar a mensaxe NON a máquina enviará un RST.

- *Acknowledgement (ACK)*: tipo de mensaxe requirida como resposta nos mensaxes CON. Existe un tempo máximo de espera dunha máquina por unha mensaxe ACK, este tempo está definido na variable ACK_TIMEOUT (2 segundos por defecto).

- *Reset (RST)*: tipo de mensaxe que se envía cando unha máquina non é capaz de procesar o paquete recibido.

Se unha solicitude é de tipo NON, a resposta que debe devolver a máquina será igualmente de tipo NON. Sen embargo, pode darse o caso de que unha máquina que envíe un mensaxe CON e reciba como resposta, logo do ACK correspondente, unha mensaxe de tipo NON. Tamén pode pasar á inversa, que unha máquina envíe a mensaxe de tipo NON e que a resposta sexa de tipo CON [23].

Capa de Solicitude-Resposta

En CoAP, cada solicitude e resposta ten un Código de Método (*Method Code*) e un Código de Resposta (Código de Resposta) [24]. Os códigos de método son [25]:

- $0.01 \rightarrow GET$: Devolve unha representación da información correspondente ao recurso identificado pola dirección URI da solicitude. Se a solicitude se realizou con éxito, a resposta debe devolver un código 2.05 (*Content*) ou un 2.03 (*Valid*).
- $0.02 \rightarrow POST$: solicita á máquina destino que a representación contida na mensaxe sexa procesada. A forma na que se procese ou os cambios que poida realizar a petición dependen da programación do dispositivo final. Se como resultado da solicitude se creou un recurso deberase devolver na resposta un código 2.01 (*Created*) e debe incluír a URI do novo recurso creado. Se a solicitude soamente fixo cambios nalgún recurso deberá devolver un 2.04 (*Changed*) e se borrou un recurso, 2.02 (*Deleted*).
- $0.03 \rightarrow PUT$: solicita á máquina destino que o recurso identificado pola URI sexa actualizado ou creado segundo os datos que contén a petición. Os códigos de resposta que pode devolver este método son 2.01 (*Created*) ou 2.04 (*Changed*).
- $0.04 \rightarrow DELETE$: solicita que o recurso identificado pola URI sexa eliminado. O código que devolve unha operacion DELETE exitosa será 2.02 (*Deleted*).

Os códigos de resposta son [26]:

- $2.01 \rightarrow Created$
- $2.02 \rightarrow Deleted$
- $2.03 \rightarrow Valid$: indica que a representación dun recurso almacenado na memoria caché dun dispositivo aínda é válido (non desactualizado).
- $2.04 \rightarrow Changed$

- $2.05 \rightarrow Content$: indica que no *payload* (carga útil) da resposta ven incluída a representación do recurso solicitado.
- $4.00 \rightarrow BadRequest$
- $4.01 \rightarrow Unauthorized$
- $4.02 \rightarrow BadOption$
- $4.03 \rightarrow Forbidden$
- $4.04 \rightarrow NotFound$
- $4.05 \rightarrow MethodNotAllowed$
- $4.06 \rightarrow NotAcceptable$
- $4.12 \rightarrow PreconditionFailed$: indica que unha precondición incluída no paquete solicitude mediante as opcións *If-Match* ou *If-None-Match*, non se cumpre.
- $4.13 \rightarrow RequestEntityTooLarge$: indica que o servidor non é capaz de procesar solicitudes tan grandes.
- $4.15 \rightarrow UnsupportedContent - Format$
- $5.00 \rightarrow InternalServerError$
- $5.01 \rightarrow NotImplemented$: indica que o método requirido na solicitude non está implantado no servidor.
- $5.02 \rightarrow BadGateway$
- $5.03 \rightarrow ServiceUnavailable$
- $5.04 \rightarrow GatewayTimeout$
- $5.05 \rightarrow ProxyingNotSupported$

Como podemos ver, existe unha correspondencia moi grande dos métodos e dos códigos de resposta que se utilizan nunha petición e resposta CoAP.

En CoAP existen ademais as chamadas respostas *Piggybacked*. Estas consisten en que a representación do obxecto solicitado na petición vai directamente introducido na mensaxe ACK que envía o servidor como confirmación de recepción da petición enviada polo cliente [27]. Por tanto a petición que envía o cliente é de tipo CON, e no ACK que envía o servidor tamén ven introducida a resposta.

2.3.2 Estrutura dun Paquete CoAP

Un paquete CoAP consta de varias partes [28], coma podemos ver na imaxe 2.3. Cada unha ten a súa función, como explicamos a continuación.

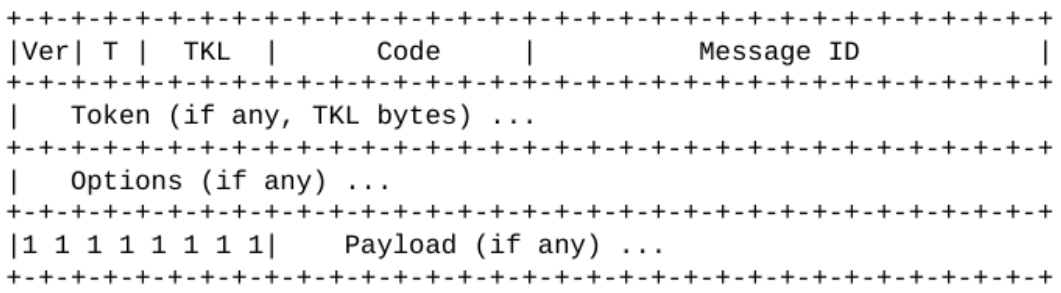


Figura 2.3: Estrutura paquete CoAP

Version

Este campo ten un tamaño de 2 bits e indica o número de versión do protocolo CoAP. Actualmente só existe a versión 1 de CoAP, por tanto este campo debe conter un ”01”.

Type

Ten un tamaño de 2 bits e indica o Código do Método CoAP que terá o paquete. Os códigos son os que vimos antes en 2.3.1 e correspóndense con: 0 para mensaxes CON, 1 para mensaxes NON, 2 para ACK e 3 para *Reset*.

Token Length

Este campo ten un tamaño de 4 bits. Indica o tamaño en bytes que ten o campo *Token* do paquete. Os tamaños de entre 9 e 15 bits están reservados e non deben ser usados.

Code

Ten un tamaño de 8 bits. Está dividido en dúas partes. Os primeiros 3 bits indícanos o tipo de mensaxe:

- 0 : solicitude (*request*)
- 2 : resposta normal (*response*), sen erros
- 4 : resposta errónea do cliente (*client error response*)
- 5 : resposta errónea do servidor (*server error response*)

Os 5 bits restantes son complementarios cos códigos anteriores e proporcionan información adicional. A agrupación dos 3 bits cos 5 adicionais correspóndese cos Códigos de Resposta vistos en (2.3.1, Capa de Solicitude-Resposta). Se collemos por exemplo o *2.01 Created*, podemos ver que os tres bits máis significativos (010, se descompoñemos o número 201 en binario) indícanos que é de tipo ”2” por tanto é de resposta. Os outros 5 bits soamente nos aclaran que tipo de resposta, neste caso un *Created*.

Como excepción teriamos o valor de código 0.00 que significaría Mensaxe Baleiro (*Empty Message*).

Message ID

Este campo ten un tamaño de 16 bits. Úsase para a detección de mensaxes duplicadas e para comprobar que os mensaxes de ACK e de RST se corresponden con respostas a mensaxes CON e NON. Esto cúmprese porque por definición as mensaxes ACK e RST deben de ter o valor *message ID* igual ao *message ID* da petición que as orixinou. Cada ID de mensaxe debe ser distinto e non poden repetirse. O primeiro *message ID* dun dispositivo ou conexión debe ser xerado aleatoriamente [29], inda que despois pode ir incrementándose á vez que se envían os paquetes.

Token

Ten un tamaño de entre 0 e 8 bytes, especificado polo campo *Token Lenght* anteriormente visto. Cada petición CoAP porta un *token* que a máquina destino debe incluír na súa resposta sen modificalo. Desta forma, podemos asociar a petición coa resposta de maneira inmediata [30]. O campo *Token* está pensado para diferenciar solicitudes simultáneas, cada *token* é único para unha petición-resposta entre dúas máquinas. Sen embargo, un cliente pode usar o mesmo *token* para calquera *request* sempre e cando use unha dirección de destino diferente. Un cliente que envíe respostas usando o protocolo UDP debe usar *tokens* xerados aleatoriamente como medida de protección fronte a ataques de *spoofing* (suplantación). O tamaño depende do entorno no que estean os dispositivos, pero se existe conectividade con Internet o tamaño mínimo recomendado é de 4 bytes (32 bits).

Options

CoAP define unha serie de opcións (*Options*) que poden ser incluídas na mensaxe para especificar un valor que logo se terá en conta á hora de procesar a petición. Estas opcións son [31]:

- *Uri-Host, Uri-Port, Uri-Path e Uri-Query*: estas opcións especifican o recurso obxectivo da petición ao servidor CoAP. A URI completa do recurso divídese nestas opcións, e

logo no servidor reconstrúese seguindo a sintaxe predefinida no protocolo [32]. Unha dirección dun recurso CoAP segue o formato:

$$coap://"Uri-Host":"Uri-Port"/"Uri-Path"?"Uri-Query"$$

  - *Uri-Host* especifica o nome ou a dirección IP da máquina que actúa como servidor do recurso.
  - *Uri-Port* especifica o número de porto do servidor do recurso.
  - *Uri-Path* especifica a localización absoluta do recurso dentro do servidor.
  - *Uri-Query* especifica un argumento parametrizado do recurso.

- *Proxy-Uri e Proxy-Scheme*: indican o uso dun *proxy* intermedio entre a máquina que actúa como cliente e o servidor.
- *Content-Format*: Indica o formato de representación do obxecto que vai incluído no *payload* (carga útil, datos) da mensaxe CoAP.
- *Accept*: esta opción serve para indicar que tipo de formato (*Content-Format*) é aceptable para o cliente.
- *Max-Age*: indica o tempo máximo que unha resposta pode estar en memoria caché (ex. dun *proxy*), antes de que sexa considerada desactualizada e inválida.
- *Location-Path e Location-Query*: indican a Uri dun obxecto que resulta dunha petición POST ou PUT ao crear un novo recurso.
- *If-Match e If-None-Match*: son opcións condicionais que se necesitan cumprir para que o servidor poida procesar a solicitude completa e enviar unha resposta.
- *Size1*: serve para indicar información do tamaño sobre representación dun recurso nunha petición CoAP (ex. o tamaño máximo de solicitude que o servidor pode procesar).

Payload

Carga Útil, neste campo está representado o obxecto que devolve o servidor da forma que especifica a opción *Content-Format* se a houbera. Antes do *payload* ben o denominado *Payload Marker*, que ten un tamaño de 1 byte e indica o final do campo Options e o comezo do *Payload*. A marca de *payload* en CoAP é 0xFF (11111111 en binario).

2.4 Ataques a Redes IoT

Dado que a tecnoloxía IoT pretende ser fundamental e utilitaria para o noso día a día e para os medios de produción, débense ter en conta os aspectos relativos á seguridade. Dentro da gran variedade de ataques intencionados a redes de información, podemos acoutar os máis importantes para as redes IoT. Estes dividiríanse en [33]:

- DOS (*Denial of Service*): Ataques por Denegación de Servizo, un clásico de calquera rede de datos. Este ataque ten como obxectivo inhabilitar o uso dun sistema, unha aplicación ou unha máquina, co fin de bloquear o servizo para o que está destinado. Máis orientado a IoT podemos distinguir 4 grupos fundamentais de ataques DOS:
  - *Jammers*: aplicados sobre redes inalámbricas fundamentalmente, a máquina atacante ocupa o canal de comunicación polo que quere transmitir o dispositivo (ex. sensor).
  - *Network Congestion*: O obxectivo principal deste ataque é conseguir atrasar o máximo posible a entrega de datos na rede. Isto pode conseguirse inundando a rede de paquetes aleatorios, publicando información sobre enrutamento falsa...
  - *Packet Dropping*: ten como obxectivo descartar paquetes do fluxo de datos entre dúas máquinas. Pódense descartar todos os paquetes ou deixar pasar algúns (*Selective Forwarding*). Se se descartan todos entón este tipo de ataque chamaríase *BlackHole*.
  - *Energy Consumption*: Como xa comentamos en numerosas ocasións anteriormente, un dos problemas fundamentais á hora de deseñar e construír redes de dispositivos IoT é o consumo enerxético. Este ataque intenta aumentar o consumo de recursos dos dispositivos vítima. Pode realizarse enviando ás máquinas obxectivo paquetes *Hello* que deberán contestar, dado que se usan para iniciar as conexións ou establecer relacións de veciñanza. Tamén se pode tentar un *Session Flooding*, enviando solicitudes de sesión ás vítimas ata que esgotan o seu número máximo de conexións.
- *Privacy Attacks*: o obxectivo principal destes ataques é tentar atopar información sobre persoas. IoT pode controlar diversos tipos de dispositivos que usa a xente no día a día, polo que poden revelar moita información sobre alguén. Estes pódense dividir en:
  - Orientados a Datos: Empregan técnicas como o *Eavesdropping* (escoitar de forma secreta [34]) espiando o fluxo de datos para sacar calquera posible información. Logo pódese acompañar con ataques de substitución, clonado e repetición (*Replay Attack*).

  - Orientados a Contexto: Escoitan o medio e logo levan a cabo ataques *Tampering*, nos que se modifica unha parte dos datos dos paquetes pertencentes ao diálogo cliente servidor para así conseguir credenciais, permisos ou datos cruciais [35].

- *Impersonation*: Ataques de suplantación, o atacante asume a identidade dun nodo físico da comunicación, ou varios. Poden ser de dous tipos, xeralmente:

  - Identificación de Capa Física: teñen como obxectivo suplantar un nodo obxectivo xerando paquetes ou sinais que conteñan datos ou características que permitan facerse coa identidade do dispositivo. Unha vez se fai efectiva a suplantación, o atacante terá acceso a todos os datos que se transmitan á máquina vítima e poderá manipulalos ou enviar o que considere.
  - *Sybil*: Os ataques de tipo *Sybil* tratan de contaminar un sistema distribuído creando un gran número de identidades independentes e usala para obter unha influencia na rede desproporcionada [36], podendo alterar rutas, modificar o contido de mensaxes...

Como podemos ver, as redes e dispositivos IoT están expostos a moitos e diversos ataques. É moi probable que o custo en recursos para a defensa contra eles moitas veces sexa inalcanzable para os dispositivos. Por tanto hai que prestar especial coidado no deseño da rede e na conectividade que lle proporcionemos aos dispositivos. Os protocolos normalmente implantan diversos mecanismos de detección destes ataques, por tanto convén aproveitalos.

Os modelos de ataques que trataremos de exemplificar neste traballo serán de tipo *Privacy Attacks* e DOS. Realizaremos un ataque que chamaremos de "Interceptación", que pertence ao tipo *Privacy Attacks*, no que escoitaremos o medio e realizaremos un intercambio de información fidedigna por outra errónea. Outro ataque tamén do tipo dos *Privacy Attacks* é o de "Duplicación", no que duplicaremos as respostas dun sensor ás solicitudes do cliente. No que concerne ao tipo de ataques *DOS*, implantaremos unha anomalía no fluxo de datos que chamaremos "Eliminación", e na que cortaremos toda a comunicación da máquina cliente cun dos servidores.

Capítulo 3

Fundamentos Tecnolóxicos

PARA conseguir levar a cabo todos os obxectivos propostos neste traballo, faremos agora un breve resumo e descrición das ferramentas que usaremos. Ao tratarse dun traballo de enxeñaría, é importante unha correcta valoración e elección dos instrumentos necesarios. Así mesmo explicaremos o por que de tal elección.

3.1 Python

Python é unha linguaxe de programación multiparadigma, isto é, permite Programación orientada a obxectos, programación estruturada, programación funcional... Tamén se trata dunha linguaxe interpretada, usa tipado dinámico (fortemente tipado) e é multiplataforma [37].

A elección desta linguaxe baséase na comodidade e no alto nivel de programación que ofrece. Permite a instalación dunha gran variedade de paquetes externos e, a día de hoxe, é utilizado no desenrolo de gran cantidade de aplicacións e funcionalidades. Tal e como veremos no seguinte punto, o uso de Python tamén é preferente neste traballo debido a que nos permite empregar a ferramenta Scapy para a creación e manexo de paquetes.

3.2 Scapy

É unha ferramenta de manipulación de paquetes de rede. Está escrita en Python e é capaz de construír distintas mensaxes, na inmensa maioría de protocolos existentes [38], da forma que se lle indique; e logo envialos á rede. Tamén é capaz de escoitar a rede e descodificar mensaxes para poder visualizar o seu contido e cambialo. Estas características permítennos crear ferramentas e manipular paquetes da forma que máis nos conveña. Incluso é posible crear os nosos propios protocolos de rede.

Por estes motivos e pola súa versatilidade, Scapy foi a ferramenta escollida para levar a

cabo o desenvolvemento deste traballo. O protocolo CoAP está xa definido en Scapy, polo que non teremos que realizar ningunha implantación nese sentido.

3.3 Virtual Box

Virtual Box é un software de virtualización deseñado por Oracle. Permite instalar sistemas operativos (sistemas invitados) dentro do sistema operativo da computadora (sistema anfitrión). Ten unha interface gráfica para a administración de todas as máquinas e recursos necesarios. Co fin de integrar as funcións do sistema anfitrión nos sistemas invitados, incluíndo carpetas compartidas, portapapeis compartido, aceleración de vídeo e un modo de ventás fluído; proporcionnase as denominadas *Guest Additions* que, unha vez instaladas na máquina virtual, engaden as funcionalidades anteriormente citadas ao sistema invitado [39].

Debido á súa sinxeleza e interface intuitiva, Virtual Box é a ferramenta escollida para realizar a virtualización do noso entorno de probas.

3.4 Netcat

Netcat é unha ferramenta de rede que nos permite, a través do intérprete de comandos, abrir portos TCP/UDP nun equipo [40]. Tamén é posible asociar unha shell a un porto e forzar conexións UDP/TCP. O seu funcionamento básico sería, primeiramente, crear un socket na máquina para conectarse a outra (ou para recibir conexións doutra); e logo enviar todo o que entre a través da entrada estándar polo socket aberto ou ben sacar por saída estándar todo o recibido dende o socket.

Esta ferramenta vainos permitir abrir o porto que nos queiramos e permanecer a escoita de paquetes (ex. paquetes CoAP). Por este motivo imos usala durante este traballo.

3.5 Ettercap

É un programa libre e de código aberto que proporciona funcionalidades para realizar ataques de MitM (Man-in-the-middle) nunha rede [41]. Pode ser usado tamén para análise de protocolos de rede e auditorías de seguridade. Está escrito en linguaxe C e pode ser executado dende a maioría dos sistemas operativos Unix e en Microsoft Windows. É capaz de interceptar tráfico nun segmento da rede, capturar contrasinais e levar a cabo unha escoita activa da maioría dos protocolos. Tamén é posible crear os denominados ”filtros”, para levar a cabo accións nos paquetes antes de ser reenviados.

Para a parte deste traballo na que se leven a cabo ataques contra o protocolo CoAP e a rede IoT virtualizada, esta ferramenta será a máis axeitada para o fin que buscamos (interceptar comunicacións e crear un *proxy* entre o cliente e o sensor).

3.6 Wireshark

Wireshark é un programa libre e de código aberto orientado a analizar paquetes de rede [42]. É usado para a detección de problemas na mensaxería, análise de rede e desenvolvemento de protocolos de comunicación. A súa funcionalidade básica é facer de *sniffer* da rede (similar a tcpdump), co engadido dunha interface gráfica que incorpora filtros e opcións de visualización de paquetes e fluxos de datos.

Por estes motivos é a ferramenta que empregaremos para visualizar e analizar os diálogos máquina-sensor que se leven a cabo usando o protocolo CoAP. Dado que ten a interface gráfica que permite visualizar os paquetes conforme van chegando ás máquinas, podemos ir comprobando o correcto funcionamento das probas durante a súa execución.

Capítulo 4

Metodoloxía

PARA a realización deste traballo faremos uso de dúas metodoloxías. Primeiramente, usaremos un modelo metodolóxico cun enfoque tradicional, buscando sobre todo o estudo en profundidade das ferramentas e protocolos a utilizar, así como o posterior deseño que cubra a totalidade dos obxectivos. Consta das seguintes fases:

- Fase do Estado do Arte e das tecnoloxías involucradas: levaremos a cabo unha busca de información sobre os diferentes protocolos das redes IoT. Centrarémonos despois no protocolo CoAP, do que faremos un estudo e análise das súas características, para logo poder basearnos nelas á hora de facer o deseño da nosa implantación, e poder utilizalas de xeito correcto.

- Fase de selección e exploración de ferramentas: realizaremos unha escolla entre as diferentes ferramentas que nos poidan servir para conseguir os obxectivos que definimos neste traballo. Para iso levaremos a cabo un pequeno estudo de cada unha delas, así como pequenas probas para avaliar o seu comportamento. Por último, seleccionaremos as máis adecuadas e faremos unha breve descrición das súas características principais.

- Fase de Análise e Deseño: unha vez feito o estudo do protocolo no que nos centraremos e mais a selección das ferramentas que usaremos, o seguinte paso será a realización dunha análise e adaptación das características do protocolo CoAP, e mais das ferramentas escollidas, aos nosos obxectivos. Acompañaremos esa parte dun deseño para a nosa posterior implantación, que cubrirá por completo todos os obxectivos propostos para este traballo.

Unha vez completadas as fases anteriores, deberemos dispoñer dunha visión clara sobre as tarefas que deberemos realizar. Para esta parte práctica do traballo, utilizaremos unha metodoloxía iterativa e incremental. Iterativa porque as fases realizaranse de maneira secuencial; e incremental debido a que en cada fase engadirase unha nova funcionalidade previamente definida. Podemos, pois, definir as seguintes fases:

- Fase de creación do entorno: virtualizaremos o entorno no que nos basearemos, descrito na fase de análise, empregando as ferramentas adecuadas para esta tarefa.

- Fase de implantación e desenvolvemento do entorno sen ataques: definiremos e implantaremos o comportamento que seguiran as máquinas do noso entorno. Programaremos o comportamento dos fluxos de datos da rede, tanto as mensaxes que se enviarán como as respostas que se obterán, baseándonos na fase de análise e no deseño proposto. Tamén definiremos as probas necesarias para dar por correcta a implantación desta fase.

- Fase de experimentación e probas do entorno sen ataques: adaptaremos a implantación antes realizada corrixindo calquera erro que se puidera producir. Empregaremos o conxunto de probas para validar o correcto funcionamento da implantación.

- Fase de implantación e desenvolvemento do entorno con ataques: unha vez temos o comportamento básico do entorno programado, podemos proceder a introducir as anomalías e definidas na fase de análise e deseño. Estas anomalías alterarán os fluxos de datos CoAP do tráfico resultante entre a máquina que actúa como cliente a un dos servidores (sensores). Así mesmo, definiremos as probas pertinentes que comprobarán a validez da nosa implantación.

- Fase de experimentación e probas do entorno con ataques e execución da simulación final: corrixiremos os erros que puidera haber na implantación descrita na anterior fase, e aplicaremos o conxunto de probas. Unha vez todas as anomalías se executen con éxito, poderemos proceder a realizar a captura do *Dataset* final.

- Fase de análise de resultados: utilizando o *Dataset* obtido na anterior fase, procederemos a obter as conclusións oportunas do traballo realizado, amosando tamén algunhas gráficas obtidas mediante análise estatístico do *Dataset*.

O uso destas dúas metodoloxías, así como a división en fases, constituirá a forma de realizar o traballo ao completo e cumprir os obxectivos de forma correcta.

4.1 Planificación e Avaliación de Custos

Agora veremos a distribución horaria feita para este proxecto, e realizaremos o correspondente presuposto. Tendo en conta as fases anteriores, e engadindo ao final o tempo dedicado á elaboración desta memoria e da súa presentación, a asignación quedaría do seguinte xeito:

- Estado do Arte e das Tecnoloxías Involucradas: *30 horas* de analista.

- Selección e Exploración de Ferramentas: *20 horas* de analista.

- Análise e Deseño: *50 horas* de analista.
- Creación do Entorno Virtualizado: *15 horas* de programador.
- Implantación e Desenvolvemento do Entorno Sen Ataques: *45 horas*de programador.
- Experimentación e Probas do Entorno Sen Ataques: *35 horas* de programador.
- Implantación e Desenvolvemento do Entorno Con Ataques: *40 horas* de programador.
- Experimentación e Probas do Entorno Con Ataques: *25 horas* de programador.
- Análise de Resultados: *15 horas* de analista.
- Elaboración de Documentación: *25 horas* de analista.

Para elaborar o presuposto en función da distribución horaria, sumaremos as horas asignadas a cada rol empresarial xa indicado, e estableceremos un prezo base de 30€ para o Analista e de 20€ para o Programador. Como resultado teremos que:

Rol	**Prezo Hora**	**Horas Totais**	**Prezo Total**
Analista	30€	140	4200€
Programador	20€	160	3200€
Total			7400€

Capítulo 5

Análise e Deseño

PARA levar a cabo a realización dos nosos obxectivos, deberemos realizar agora unha análise e un deseño do sistema, para así conseguilos tamén de forma correcta. Para isto primeiro debemos ver o entorno sobre o que faremos a nosa simulación.

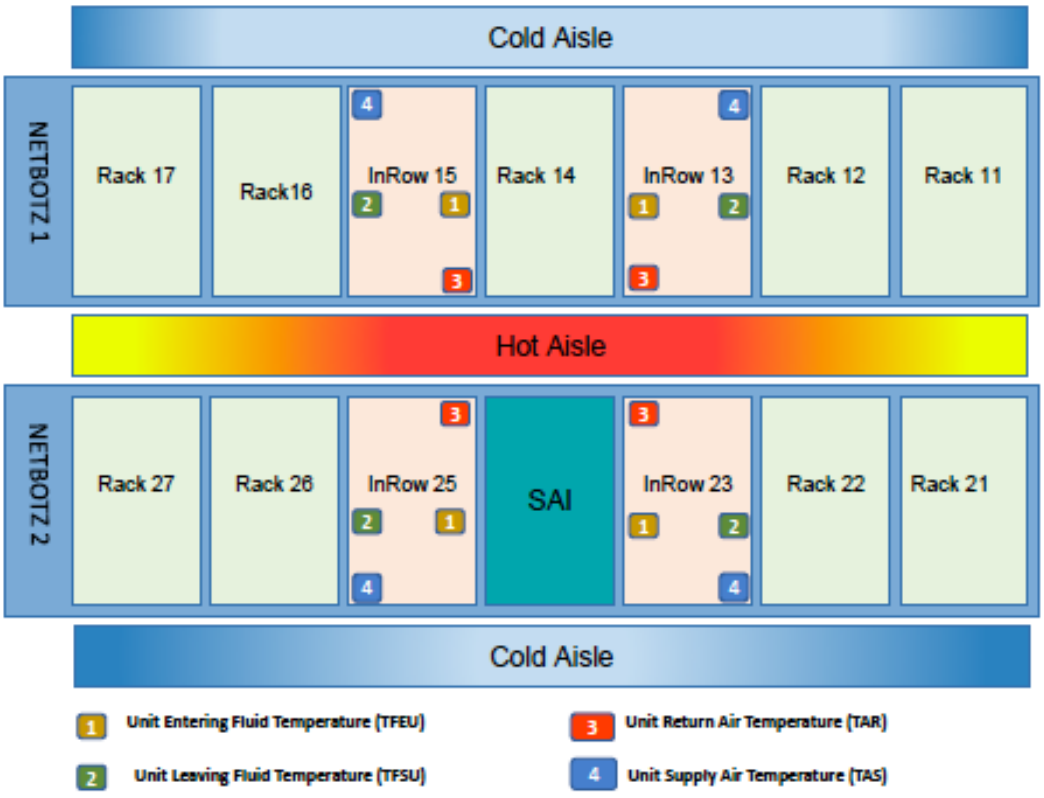


Figura 5.1: Distribución dos sensores e das Climatizadoras InRow no CPD

Como vemos na imaxe 5.1 , que amosa unha distribución dun CPD real (neste caso o

do CITIC da Universidade da Coruña) con corredor quente pechado (HACS), a parte na que nos centraremos son os sensores de temperatura, en concreto, nos sensores de temperatura que regulan as máquinas enfriadoras exteriores. Neste caso, os sensores están situados nas climatizadoras de tipo InRow posicionadas entre os Rack (armarios de compoñentes) do CPD. Cada InRow ten 4 sensores de temperatura que son:

- 1 : Unit Entering Fluid Temperature (TFEU, Temperatura do Fluído que Entra na Unidade), mide a temperatura do fluído de entrada á unidade InRow. O fluído normalmente sería auga provinte dunha enfriadora exterior.
- 2 : Unit Leaving Fluid Temperature (TFSU, Temperatura do Fluído que Sae da Unidade), mide a temperatura do fluído que sae da InRow, de volta á enfriadora exterior.
- 3 : Unit Return Air Temperature (TAR, Temperatura do Aire de Subministración da Unidade), mide a temperatura do aire que entra á unidade climatizadora InRow provinte do corredor quente pechado.
- 4 : Unit Supply Air Temperature (TAS, Temperatura do Aire de Retorno á Unidade), mide a temperatura do aire que sae da InRow cara o corredor frío.

Cada InRow ten o control sobre estes 4 sensores, a comunicación no protocolo CoAP terá lugar entre unha máquina que fará de Cliente e cada InRow. Por tanto cada sensor será un recurso que se diferenciará mediante a URI (ex. *www.inrow15.gal/1*, identificará ao sensor 1 da InRow15). Por tanto, á hora de realizar a virtualización, haberá unha máquina virtual por cada InRow, e como Cliente pode actuar a máquina anfitrión (máquina física).

Tamén contaremos con medicións de temperatura do CPD sobre o que nos baseamos, por tanto, os datos de resposta de cada sensor procederán destas medidas. O que deberemos de facer é adaptarnos ao formato no que están representadas de maneira predefinida, para que cada valor sexa o que corresponda a cada sensor no momento exacto da simulación (veremos a estratexia a seguir para isto máis adiante, na sección 5.1.1).

Este análise e deseño do noso sistema dividirémolo en dúas partes, unha que se centrará no funcionamento básico do entorno sen ataques e, posteriormente, desenrolaremos a segunda parte na que trataremos de realizar distintos tipos de ataques ao fluxo de datos CoAP.

5.1 Entorno Sen Ataques

Nesta sección definiremos o comportamento do noso entorno virtual básico, isto é, diálogo Cliente-InRow e a correcta estruturación das mensaxes e dos seus campos.

No noso entorno podemos facer unha definición das características que teñen que cumprir cada un dos roles que temos presentes:

- Cliente: Enviará mensaxes CoAP de solicitude ás InRow (Servidores). Estas mensaxes enviaranse utilizando o protocolo UDP e serán de tipo NON (*Non-Confirmable*), xa que son as máis apropiadas para sensores [22], e non requiren mensaxe ACK de confirmación (co cal a rede estará menos sobrecargada e os dispositivos consumirán menos recursos á hora de procesar e enviar estas mensaxes). En función de cada campo da mensaxe teremos a seguinte estrutura:

  - Version (*version*): o valor deste campo será 1 (01).
  - Tipo (*type*): dado que os mensaxes serán de tipo NON, o valor será 1 (01).
  - Código (*code*): As mensaxes que se envíen dende o Cliente solicitando un recurso do servidor serán do método GET, por tanto o valor do campo *code* será 1 (0.01).
  - Identificador de Mensaxe (*Message ID*): debido as medicións de temperatura do CPD que temos, debemos calcular o ID da mensaxe de forma que se incremente segundo se executa a simulación para que as medidas sexan as correctas seguindo un orde temporal. Na sección 5.1.1 detallaremos esta situación e a solución proposta.
  - Token: Cada campo *token* de cada mensaxe será único e xerado aleatoriamente. O tamaño deste *token* pode ser como máximo 8 bytes pero, dado que o tamaño mínimo recomendado é de 4 bytes, usaremos ese.
  - Opcións (*Options*): para o noso caso, as opcións que necesitamos son *Uri-Host* e *Uri-Path*, para poder identificar o recurso correctamente no servidor correspondente. Engadiremos ademais a opción *Accept* con valor 0, que indica que o formato de representación do recurso que acepta o cliente é texto plano (*text/plain*).

  Os portos de orixe e de destino serán os 5683 do protocolo UDP (no noso caso), definidos para CoAP.

- Servidor: No noso caso serán as InRow que simulamos. Dado que existen 4 InRow, cada unha será nomeada do mesmo xeito que vimos anteriormente na imaxe 5.1, isto é: www.inrow13.gal, www.inrow15.gal, www.inrow23.gal e www.inrow25.gal. Cada InRow ten 4 sensores, cuxo identificador de recurso ou *Uri-Path* en CoAP será 1, 2, 3 e 4 respectivamente, correspondéndose este co número de cada sensor. Esa nomenclatura, tanto de servidor como de recurso, foi escollida debido a súa simplicidade e correlación inmediata do que cada nome representa. No consecuente coa estrutura da mensaxe de resposta á solicitude do Cliente, esta quedaría da forma:

  - Version (*version*): a mesma que a da mensaxe orixinal (01).
  - Tipo (*type*): mesmo tipo que a mensaxe orixinal (tipo NON, 01).

- Código (*code*): O código de resposta é *2.05 Content*, que inclúe a representación do recurso solicitado.
- ID de Mensaxe (*Message ID*): é xerado aleatoriamente para cada mensaxe de resposta.
- *Token*: o mesmo que o da mensaxe de solicitude.
- Opcións (*Options*): as opcións *Uri-Host* e *Uri-Path* son as mesmas que as da solicitude; pero a opción *Accept* é cambiada pola opción *Content-Format*, que especifica o formato de representación do obxecto incluído no campo *Payload* da mensaxe (no noso caso o valor será 0, *text/plain*).
- Carga Útil (*Payload*): neste campo estaría incluída a representación do obxecto, no noso caso a temperatura que marca o sensor.

5.1.1 Message ID e Medicións de Temperaturas

Como xa comentamos anteriormente, temos un ficheiro con medicións reais do CPD que simulamos. Este ficheiro está escrito en linguaxe Python, e cada temperatura calcúlase usando unha función que deberemos importar para poder executala pasándolle os datos da forma:

```
1    temperature ( z, clid, ds)
```

Listing 5.1: Parámetros Función Temperatura

Dicir, antes de nada, que este cálculo está realizado sobre unha modelización levada a cabo polo Grupo de Investigación Telemática da UDC, e é súa a propiedade intelectual do algoritmo. Neste traballo, simplemente levamos a cabo a implantación de dito algoritmo de cálculo das temperaturas.

Estes parámetros significan:

- z : Número de iteración. Enténdese por iteración a execución de todas as solicitudes a todos os servidores (InRow) por parte do Cliente. Por tanto, cada iteración constará de 16 solicitudes (4 sensores en 4 InRow).
- clid: Identificador do cliente. É a concatenación do número da InRow e do número do sensor, por exemplo, o sensor 1 da InRow13 tería o clid=131 (13 por InRow13 e 1 polo sensor 1).
- ds: Día da semana. Dependendo do día da semana no que queiramos facer a simulación estableceremos este valor segundo corresponda de 1 a 7. Debido ao deseño e a estrutura das medidas no ficheiro de mostras, os días da semana non se corresponden coa súa numeración tradicional (0-Luns, 1-Martes,...) senón que será 0-Xoves, 1-Venres, 2-Sábado,...

O problema desta implantación é que non podemos facerlle cambios, senón que temos que adaptarnos e calcular os datos correctamente antes de poder pasarllos á función *temperature*. O "clid" podemos calculalo en función do *Uri-Host* e do *Uri-Path* da mensaxe, e o "ds" (dia da semana) podemos poñer o que queiramos simular ou incluso crear unha pequena función que o calcule dependendo do dia real no que executemos a simulación.

No que concerne a "z" (número de iteración), preséntasenos unha situación máis complexa. Debemos calcular cada iteración en función dun parámetro, e debe provir da máquina Cliente; para que así tamén funcione como marca de sincronismo en situacións que impliquen perda de paquetes (ataques ao fluxo de datos) e que imposibiliten levar a conta de maneira simple. Neste caso, empregaremos o ID da Mensaxe solicitude.

O que faremos no Cliente será crear un primeiro ID de Mensaxe aleatorio ao instanciar a clase ClienteCoAP e gardalo nunha variable. Isto farase chamando a unha pequena función que escollerá un número aleatorio do rango entre 0 e 65535. Logo, cada vez que se envíe unha mensaxe incrementaremos ese valor en 1.

En cada Servidor (InRow), gardaremos nunha variable o primeiro valor do ID de Mensaxe que veña do Cliente. Nas solicitudes posteriores calcularemos a iteración na que nos atopamos da seguinte maneira:

- Calcularemos a diferenza entre o ID da Mensaxe da solicitude que imos a procesar co ID da Mensaxe primeira, que temos gardado nunha variable.

- Dividiremos a diferenza anterior entre 16 (solicitudes totais dunha iteración, 4 por cada InRow) e colleremos a parte enteira do resultado.

Desta maneira sempre poderemos calcular o número exacto de iteración na que se atopa a solicitude recibida, inda que se perderan paquetes do fluxo de datos normal entre o Cliente e un dos Servidores. Hai que ter en conta que o primeiro paquete que se envíe a cada servidor non se pode perder (non se levará acabo un ataque dende o principio), porque senón xa non funcionaría este método de sincronismo.

5.1.2 Definición de Probas Entorno Sen Ataques

Para comprobar o correcto funcionamento deste desenrolo, unha vez feita a implantación do Cliente e dos servidores (InRow), utilizaremos a ferramenta Wireshark descrita anteriormente (3.6) para facer unha captura do tráfico da rede IoT. Logo visualizaremos os paquetes que se intercambiaron entre o Cliente e unha das InRow para poder verificar que son correctos; isto é, que se enviara unha mensaxe e se recibira unha resposta por cada sensor, e que cada unha das mensaxes estea construída segundo o deseño antes descrito.

5.2 Entorno Con Ataques

Unha vez temos a simulación da nosa rede IoT con fluxos do protocolo CoAP funcionando con normalidade, podemos pasar ao seguinte apartado no que procederemos á introducir unha serie de ataques comúns destas redes. Serán ataques sinxelos, que tratarán de exemplificar o comportamento destes, para conseguir que o *dataset* sexa completo e a IA poida aprender e identificar cando unha rede ou dispositivo IoT está sendo atacado.

Os ataques que propoñemos son 3 e poden definirse da seguinte maneira:

- Interceptación: É un ataque de tipo *Privacy Attacks* segundo a clasificación que describimos anteriormente na sección de Ataques a Redes IoT (2.4). Faremos unha escoita do medio e trataremos de interpoñernos entre o Cliente e unha das InRow mediante un ataque Mitm (*Man-in-the-middle*, ou ataque de Home no medio), para así capturar e reenviar todo o tráfico entre os dous dispositivos (facendo a función de *proxy*). Unha vez conseguido isto, realizaremos modificacións nos datos dos paquetes antes de que sexan reenviados dende a InRow ao Cliente, alterando así o correcto funcionamento da rede e das máquinas sobre as que o sensor teña efecto directamente. O que faremos será incrementar a temperatura dos sensores dunha InRow; o que pode provocar, por exemplo, que a propia InRow funcione máis rápido ao igual que a enfriadora exterior, aumentando así o consumo de enerxía do CPD, quizais provocando unha avaría no sistema de refrixeración...

- Duplicación: Este ataque tamén se pode clasificar de tipo *Privacy Attack*, se ben é certo que atenta contra a seguridade dunha comunicación, dado que ten aceso á mensaxe, pero o que fai é simplemente duplicar os paquetes enviados dende a InRow ao Cliente. A máquina Cliente (ou calquera máquina cliente nunha rede IoT) espera a chegada dunha resposta, non de dúas, o que pode provocar un comportamento non esperado ou impredicible.

- Eliminación: Este ataque, tamén chamado *Packet-Dropping*, pertence á categoría de ataques DOS (denegación de servizo), e nel o que faremos é eliminar todos os paquetes de resposta dunha InRow a solicitudes do Cliente.

Agora soamente queda facer un uso adecuado das ferramentas antes escollidas para levar a cabo de forma correcta estes ataques á nosa rede IoT.

5.2.1 Definición de Probas Entorno Con Ataques

Esta parte da implantación probarase realizando cada un dos ataques mentres existe fluxo de datos Cliente-InRow e, ademais, se leva a cabo unha captura de tráfico. Cada ataque

realizarase individualmente e será dirixido a unha InRow en concreto. Despois, simplemente teremos que comprobar nos paquetes se se produciron os cambios correspondentes no ataque de Intercepción, se se descartaron os paquetes no ataque de Eliminación e se se duplicaron os paquetes da comunicación no ataque de Duplicación.

5.2.2 Captura da Simulación Final e Creación do Dataset

A Simulación Final consistirá en utilizar toda a implantación feita nos anteriores apartados nunha soa captura. É dicir, realizaremos unha captura de tráfico na que amosaremos unha situación de normalidade durante unhas horas, logo comezaremos a introducir as nosas anomalías (ataques) dirixidos a unha InRow en concreto e separados no tempo.

A duración da simulación será de un día enteiro (24 horas), e a distribución horaria será a seguinte:

- Comezaremos a simular as 0:00, e deixaremos que se execute un comportamento normal ata as 8:00.
- De 8:00 ás 9:00 levaremos a cabo o ataque de ”Interceptación”. Logo pararemos o ataque e a simulación continuará de xeito normal.
- De 14:00 a 15:00 executaremos o ataque de ”Eliminación”. Unha vez finalizado o tempo para este ataque deixaremos que a simulación siga normalmente.
- De 20:00 a 21:00 realizaremos o ataque de ”Duplicación”. Este será o último ataque que exemplificaremos.
- Ás 23:59 pararemos a simulación e gardaremos a captura resultante.

Como vemos, a duración dos ataques será de unha hora. Cada iteración de solicitudes da máquina Cliente aos Servidores (InRow) estará espazada 5 minutos (300 segundos), de tal forma que se executarán 12 iteracións cada hora (12 $\times$ 5 minutos = 60 minutos). En total nas 24 horas haberá 288 iteracións (24 $\times$ 12 iteracións = 288 iteracións).

Unha vez rematada a simulación, gardaremos a captura resultante nun ficheiro. Este será o noso *Dataset*, e sobre o que crearemos o arquivo de etiquetado (anotado); que consistirá en executar un pequeno programa sobre a captura gardada. Este programa contará cunha serie de instrucións que analizarán cada paquete capturado e, en función das características da mensaxe que se estea a analizar, levaranse a cabo as anotacións pertinentes. Cada paquete terá unha liña no arquivo de etiquetado, que se cubrirá da seguinte maneira:

”Número de Trama”; ”Ip Orixe”; ”Porto UDP Destino”; ”Nome Recurso CoAP”; ”ID de Mensaxe CoAP”; ”Ataque ou Non Ataque (0 ou 1)”

O valor de cada campo pode definirse como segue:

- **Número de Trama**: é o número de paquete da captura.
- **IP Orixe**: Se o paquete ten PDU de capa IP, entón escribiremos a dirección IP de orixe.
- **Porto UDP Destino**: Se o paquete ten PDU de protocolo UPD, entón anotaremos o porto de orixe.
- **Nome Recurso CoAP**: Se o paquete ten PDU de protocolo CoAP, entón anotaremos as opcións *Uri-Host* e *Uri-Path* (que identifican o recurso solicitado e devolto como resposta), e mais o identificador de mensaxe (*Message ID*)
- **ID de Mensaxe CoAP**: Se o paquete ten PDU de protocolo CoAP, entón anotaremos o valor do campo *Message ID*.
- **Ataque ou Non Ataque**: Se a dirección MAC de destino ou de orixe coincide coa dirección MAC da máquina atacante, entón estableceremos o valor deste campo a "1"; en calquera outro caso será "0". Isto indícanos que dita mensaxe forma parte dunha anomalía do tráfico

O arquivo de etiquetado final será de tipo CSV (*Comma-Separated Values*), debido á súa simplicidade e gran compatibilidade.

Capítulo 6

Implantación e Probas do Entorno Sen Ataques

Logo de facer a análise e o deseño dos nosos obxectivos, procedemos agora a realizar a parte práctica do noso traballo. Utilizaremos as ferramentas seleccionadas vistas na sección 3, e trataremos de realiza o desenvolvemento completo das características vistas no capítulo anterior 5.

6.1 Virtualización do Entorno

Para esta parte empregaremos a ferramenta Virtual Box de Oracle®, amplamente coñecida e usada. Como ben dixemos no anterior capítulo de Análise e Deseño (5), haberá catro máquinas virtuais que simularán o comportamento de cada unha das InRow, e o rol da máquina Cliente será asumido pola máquina física.

Centrándonos nas máquinas virtuais, Cada unha terá un sistema operativo o máis reducido posible (sen interface gráfica) pero que sexa común e teña as funcionalidades propias dun sistema Linux (por comodidade e coñecementos previos de Linux, simplemente). Este sistema será un Ubuntu Server versión 20.04, instalado seguindo as opcións por defecto de Virtual Box, ao que se lle instalará o paquete Scapy para Python. A ferramenta Netcat xa a trae instalada de forma nativa.

As catro máquinas virtuais estarán todas conectadas, dado que establecemos a opción ”Adaptador sólo-anfitrión” na configuración do adaptador de rede de Virtual Box para cada unha das máquinas. Con esta configuración a máquina física tamén terá conectividade coas outras máquinas. Por defecto, neste modo de conexión de Virtual Box, a rede que usarán as máquinas é a 192.168.56.0 (inda que pode cambiarse), e a dirección IP asignada á máquina anfitrioa é a 192.168.56.1 . Por tanto as outras máquinas terán establecidas unhas direccións IPs da maneira que segue:

- InRow13: ”www.inrow13.gal”, 192.168.56.2
- InRow15: ”www.inrow15.gal”, 192.168.56.3
- InRow23: ”www.inrow23.gal”, 192.168.56.4
- InRow25: ”www.inrow25.gal”, 192.168.56.5

Unha vez configuradas as máquinas segundo estes parámetros de rede, podemos apagalas e poñernos coa codificación do Cliente e Servidores CoAP.

6.2 Codificación con Scapy dos Fluxos CoAP

Unha vez virtualizado o noso entorno, podemos pasar agora a implementar o comportamento da máquina Cliente e das InRow.

Para isto usaremos linguaxe Python, en concreto na súa versión 3.6.9, xunto coa librería Scapy para a manipulación de paquetes de rede. Comezaremos pois coa implantación do Cliente.

Primeiro de todo crearemos unha clase, inda que non sería necesario dado que python soporta programación estruturada e ademais soamente crearemos unha única instancia dela, pero queda todo máis ordenado e optimizado (encapsulación, facilita o entendemento e a reutilización de código). Dado que o obxectivo xeral desta clase será enviar mensaxes a cada unha das InRow e a cada un dos sensores, podemos crear dous arrays; un para os servidores (InRow) e as súas direccións IP, e outro para os sensores. Logo crearemos un bucle aniñado *for* de forma que, para cada servidor, se recorra o array de sensores completo. Véxase como quedaría:

```
1 class ClienteCoAP(Thread):
2 def __init__(self):
3     self.servidores = [("www.inrow13.gal","192.168.56.2"),
      ("www.inrow15.gal","192.168.56.3"), ("www.inrow23.gal","192.168.56.4"),
      ("www.inrow25.gal","192.168.56.5")]
4     self.servidoresIP = ["192.168.56.2", "192.168.56.3", "192.168.56.4",
      "192.168.56.5"]
5     self.sensores = ["1", "2", "3", "4"]
6
7         for servidor,ip in self.servidores:
8             for sensor in self.sensores:
```

Listing 6.1: Clase ClienteCoAP con construtor, arrays de servidores e sensores, e bucle anidado.

Desta forma para cada par "servidor,ip" , e para un dos seus sensores, poderemos executar o código necesario para enviar a mensaxe de solicitude.

Crearemos un método que se executará ao instanciar a clase, por tanto crearemos un construtor para a clase dende onde chamaremos a o método. Este método servirá para crear o paquete CoAP e introducirlle os datos necesarios, para iso teremos que basearnos no funcionamento de Scapy para a construción de paquetes. O funcionamento é moi simple, o que hai que facer é ir concatenando protocolos dende a capa máis baixa do modelo TCP/IP á máis alta, por exemplo un paquete dunha petición HTTP construiríase desta maneira: IP() / TCP() / HTTP() . Dentro de cada un dos parénteses que seguen ao nome do protocolo situaríanse os datos do paquete, en función dos campos que poida ter cada protocolo (ex. TCP(sport=80, dport=80)). Unha vez construída cada mensaxe, debemos usar algunha das distintas funcións para enviar o paquete á rede que nos proporciona Scapy.

No noso caso, o paquete será creado da forma que vemos:

```
1 for servidor,ip in self.servidores:
2   for sensor in self.sensores:
3     coapPackage = CoAP(ver=1, type=1, code=1, msg_id=self.create_msgID,
      token=self.create_token, options=[("Uri-Host", servidor),"Uri-Path",
      sensor), ("Accept", b"\x00")], paymark=b"\xff")
4     packet = (IP(dst=ip) / UDP(sport=5683, dport=5683) / coapPackage)
```

Listing 6.2: Creación do paquete CoAP dentro do Bucle Anidado en ClienteCoAP

Este código estaría dentro do bucle aniñado que vimos anteriormente, por iso aparecen as variables "servidor" e "sensor" que son as creadas polo bucle for ao recorrer os arrays. Para o ID da Mensaxe collemos a variable explicada en 5.1.1 para o caso do Cliente; e para o *Token*, chamamos a unha función creada dentro da clase ClienteCoAP e que nos devolve o valor deste campo creado de forma aleatoria. Esta función, devolve unha cadea de 4 números e letras, e a do *msg_id* devolvería un número de entre 0 e 65535 ($2^{16} - 1$).

Unha vez está construído o paquete, podemos envialo á rede utilizando a función *send* de Scapy. Esta función o que fai é enviar o paquete que especifiquemos pola interface de rede que queiramos. Nótese que esta función *send* é asíncrona, simplemente transmite unha mensaxe á rede; tamén existe unha función que, ademais de enviar o paquete, tamén espera pola resposta, esta é *sr1*). Pero, dado que CoAP utiliza por definición envío e recepción de paquetes de forma asíncrona, utilizaremos a función *send*. Entre cada unha das solicitudes establecemos un tempo de espera para o envío de 1 segundo, facemos isto para que non se envíen todas á vez e as capturas do tráfico sexan máis visuais e fáciles de seguir. Así e todo tamén sería unha boa práctica realizar esta acción nun entorno real, dado que así evitamos sobrecargar a rede en momentos puntuais como son os de toma das medidas dos sensores.

Logo de enviar cada mensaxe, a variable que garda a ID da Mensaxe no Cliente será in-

crementada en 1.

Esta sería pois toda a implantación que se realizará na máquina Cliente. Pasamos agora aos Servidores, para os cales levaremos a cabo unha implantación que logo replicaremos en cada un deles, pois o comportamento é similar en todos soamente cambiando os valores que pasaremos á función *temperature*.

Crearemos unha clase ServidorCoAP noutro ficheiro e na que definiremos o comportamento das InRow. Para comezar, o servidor ten que escoitar a rede para saber cando lle chega unha solicitude CoAP do Cliente. Para este caso utilizamos a función *sniff* de Scapy. O que fai é escoitar a rede dende a interface que lle indiquemos da máquina e, aplicando un filtro aos paquetes que chegan a esa interface, executa unha función se algún deses paquetes pasa o filtro. Se non poñemos ningún filtro executará a función que lle indiquemos todas as veces que entre un paquete pola interface. A sintaxe dos filtros non admite fórmulas complexas, de tal maneira que poñeremos *"udp port 5683"* para que a cada petición que lle chegue á máquina e sexa do protocolo UDP e porto 5683, se lle execute a función escollida. Esta función será *reply* que conterá o código necesario para enviar a mensaxe de resposta. Construiremos o paquete CoAP da mesma maneira que a vista anteriormente para o Cliente.

A función *reply*, antes de nada, realiza unha comprobación necesaria para saber se o paquete que vai procesar é unha solicitude CoAP, dado que os filtros da funciona *sniff* de Scapy non permiten unha sintaxe complexa (tal como dixemos antes). Esta comprobación consiste en saber se o paquete ten como dirección de destino o propio servidor a onde chega a solicitude, utilizando para iso a función que proporciona Scapy e chamada *getlayer* (que explicaremos a continuación); e comparando a propia dirección da InRow coa do campo destino da capa IP. Así mesmo, tamén comprobaremos se o paquete ten na súa estrutura a PDU (capa) correspondente ao protocolo CoAP usando a función de Scapy *haslayer*, que devolve un valor de tipo booleano dependendo de se a mensaxe conta coa PDU indicada ou non.

Dado que hai datos que temos que reutilizar da petición do cliente para crear a resposta deberemos extraelos da solicitude dalgunha maneira. Para isto Scapy ten unha funcionalidade chamada *getlayer* que extrae a capa do protocolo que lle indiquemos. Como vemos no seguinte código, podemos extraer a capa IP, UDP e CoAP de maneira moi sinxela:

```
1 ip = req.getlayer(IP)
2 udp = req.getlayer(UDP)
3 coap = req.getlayer(CoAP)
```

Listing 6.3: Función getlayer de Scapy

Desta forma agora podemos, por exemplo, saber a dirección ip de orixe da solicitude simplemente poñendo "ip.src". Isto podémolo aplicar ao *token* de CoAP, que vai a ser o mesmo que o da petición, así como á versión, ao tipo da mensaxe e ás opcións (só as dúas primeiras opcións). O ID da Mensaxe será creado de forma aleatoria, o código da resposta será 69 (*2.05 Content*)

indicando que no paquete se inclúe o recurso solicitado. A opcións *Uri-Host* e *Uri-Path* serán as mesmas que as da solicitude, pero a opción *Accept* será substituída por *Content-Format*, sinalando a codificación usada na representación do recurso (no noso caso *text/plain*).

```
1 if req.haslayer(CoAP) and req.getlayer(IP).dst == "192.168.56.X":
    #Substituír X polo valor correspondente da IP
2   respCoap = (CoAP(ver=coap.ver, type=coap.type, code=69,
      msg_id=self.create_msgID(), token=coap.token, options=[coap.options[0],
      coap.options[1], ("Content-Format", b"\x00")], paymark=coap.paymark) /
      Raw(load=self.calculate_replyText(coap)))
3   respIP = (IP(dst=ip.src) / UDP(sport=udp.dport, dport=udp.sport) /
      respCoap)
```

Listing 6.4: Creación paquete resposta en ServidorCoAP

Para engadir á resposta o valor de temperatura correcto, faremos uso da función *Raw* de Scapy; que o que fai é engadir os datos que lle pasamos no seu campo *load* ao *payload* do paquete. Para calcular a temperatura correcta realizamos a chamada á función *calculate_replyText* pasándolle como argumento a capa CoAP extraída anteriormente do paquete solicitude mediante a función *getlayer*.

Esta función *"calculate_replyText"* o que fai é calcular os tres valores necesarios para chamar a función *temperature* vista anteriormente (5.1.1). O valor de "z" calcularase seguindo as pautas indicadas para o Servidor no deseño visto en 5.1.1. O "clid" calcúlase coa *Uri-Host* e a *Uri-Path* das opcións de CoAP, de tal forma que para o recurso *www.inrow25.gal/3* o seu "clid" sexa 253. O valor de "ds" (día da semana) podemos establecelo ao día que queiramos simular, neste caso creamos unha función que comproba o día no que se executa o programa e establece o valor de "ds" a dito día. Unha vez executada a función *temperature*, esta devólvenos un valor de temperatura periódico, ao cal lle daremos un formato máis compacto para envialo como resposta. No noso caso enviaremos un valor con 2 díxitos enteiros e 4 díxitos decimais, que consideramos a suficiente precisión para esta situación.

Finalmente só queda enviar o paquete terminado de construír e con todos os datos necesarios. Utilizaremos a función *send*, ao igual que na clase ClienteCoAP, para mandar o paquete á rede coa dirección IP de destino do Cliente.

6.3 Probas da Implantación Sen Ataques

Unha vez programado o comportamento como vimos na sección anterior, é a hora de poñelo a proba. Para iso replicaremos o código do ficheiro servidorCoAP.py en cada unha das máquinas virtuais que creamos antes, así como unha copia do ficheiro *temperature.py* que contén o código para realizar o cálculo das medicións de temperatura. Neste paso tamén teremos que usar a utilidade Netcat descrita en 3.4 para abrir o porto CoAP (5683) en cada

unha das máquinas e que así permanezan á escoita e non devolvan mensaxes de ICMP Type 3 (*Destination Unreachable*) Code 3 (*Port Unreachable*). Para iso usaremos o comando "sudo netcat -kulp 5683 &", executado en segundo plano (&) e no que as opcións significan:

- -l: modo escoita e conexións entrantes.
- -k: manter o modo escoita indefinidamente, o comportamento por defecto de Netcat sen esta opción sería parar a escoita no momento en que chegue un paquete ao porto indicado.
- -u: usar os portos de datagramas, UDP en vez dos portos TCP.
- -p: para especificar o porto a usar.

Con ese comando Netcat executado en cada unha das máquinas (incluída o Cliente), podemos executar os arquivos clienteCoAP.py e servidorCoAP.py en cadansúa máquina mentres o programa Wireshark se executa na máquina Cliente.

Comezaremos por executar primeiro os arquivos servidorCoAP.py nas máquinas que simulan as InRow, a execución quedará á espera de solicitudes CoAP. No Cliente, unha vez arrancado Wireshark, executamos o arquivo clienteCoAP.py e vemos como van aparecendo as tramas na captura que estamos a realizar. Unha vez que se produciron diálogos con todas as máquinas podemos parar a execución (primeiro na máquina Cliente e logo nos Servidores).

O que debemos comprobar agora é que se enviaran 4 solicitudes a cada unha das InRow e que estas responderan a todas elas. Logo abriremos unha solicitude e a súa correspondente resposta e verificaremos que a información de cada un dos campos está ben formada e que se corresponde co implantado. Por exemplo, Podemos ver na imaxe 6.1 a construción final da solicitude logo de facer a captura, os campos todos coinciden co programado (versión, tipo, código de solicitude)... O *Token Length* vemos que é 4 bytes, pero o campo *Token* ten 8 caracteres, isto débese a que os 4 díxitos ou letras do *token* real construído polo programa python está codificado en Hexadecimal seguindo a táboa ASCII. En Hexadecimal cada díxito represéntase con 4 bits, e 4 bits + 4 bits son igual a 1 byte; esa é a explicación de que por cada carácter do *Token* real haxa dous díxitos no campo *Token* do paquete. O resto das opcións tamén están ben construídas, tanto o *Uri-Host* coma o *Uri-Path* están ben formados, así como a opción *Accept* (*text/plain*).

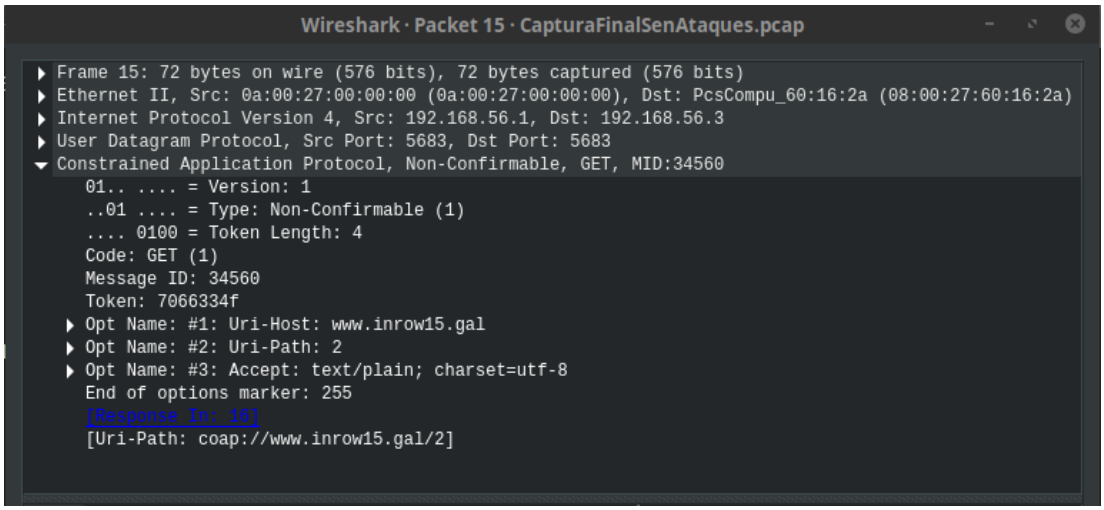


Figura 6.1: Solicitude CoAP en entorno Sen Ataques

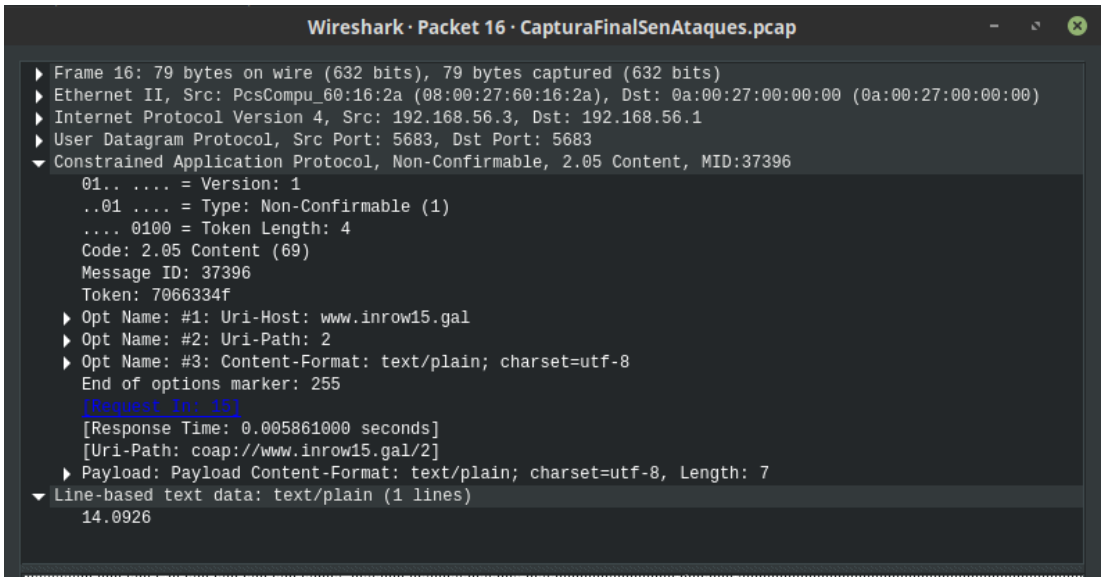


Figura 6.2: Resposta CoAP en entorno Sen Ataques

Na resposta (imaxe 6.2) podemos destacar que a versión e o tipo de mensaxe son correctos e similares aos da petición de solicitude. O código da resposta é *2.05 Content*, o que é correcto. A Id de Mensaxe é aleatoria e o *Token* é o mesmo que o da mensaxe de solicitude. Nas opcións vemos que se substituíu *Accept* por *Content-Format*, as outras opcións tamén están ben construídas (as mesmas que as da solicitude). Finalmente temos a temperatura que devolveu o sensor (facendo uso da función *temperature*), introducida no campo *Payload* e cunha lonxitude de 7 caracteres codificados en texto plano. Neste caso a temperatura devolta é de

14.0926, que sería medida en grados centígrados.

Capítulo 7

Implantación e Probas do Entorno Con Ataques

Unha vez posto en funcionamento o sistema e probado que funciona de xeito correcto, é a hora de introducir as anomalías ao fluxo de datos descritas na sección 5.2.

Unha cousa que teñen en común todos os ataques que nos propoñemos realizar é que necesitan ter acceso á comunicación que se leve a cabo entre o Servidor (InRow) vítima e a máquina Cliente. Máis que ter acceso, o que realmente necesitamos é que a nosa máquina Atacante sexa intermediaria (*proxy*) na comunicación entre os dous dispositivos. Isto obríganos a levar a cabo un ataque de MitM (Man-in-the-Middle) entre as dúas partes da comunicación para conseguir crear un *proxy* entre a InRow vítima e a máquina Cliente. Realizaremos este ataque empregando a ferramenta Ettercap (descrita en 3.5) e a técnica de *ARP Spoofing*.

Na memoria caché ARP gárdanse as asignacións $DireccinIP \rightarrow DireccinMAC$ das máquinas que coñece un dispositivo. Un ataque baseado en *ARP Spoofing* (tamén coñecido como *ARP Poisoning*) consiste en que a máquina Atacante "envelene" (*poison*) a memoria caché ARP de unha ou varias máquinas da rede, facéndolles crer que a dirección MAC da máquina Atacante está asociada á dirección IP da máquina vítima. Se unha máquina Atacante envía mensaxes ARP dicindo que a máquina con dirección IP 192.168.X.X está na súa dirección MAC, entón todas as máquinas que reciban as mensaxes ARP actualizarán a súa caché ARP aos novos datos. Cando queiran enviar unha mensaxe a esa IP, buscarán a dirección MAC na súa caché ARP e enviarán dita mensaxe con esa dirección MAC falsa. Nunha rede conmutada (utiliza dispositivos *switch*), utilízase a dirección MAC para realizar a localización do porto ao que está conectada unha máquina; por tanto, unha dirección MAC errónea levaranos a un porto diferente ao que está conectado o dispositivo real ó que se quería enviar a mensaxe. Neste novo porto é onde estará a máquina Atacante, e así esta recibe todas as mensaxes que teñen como destino a máquina suplantada.

Se realizamos o "envelenamento" a caché ARP para dúas máquinas en concreto, a máquina

Atacante será pois intermediaria das comunicacións levadas a cabo entre as dúas máquinas vítimas. Isto é pois o que nos propoñemos facer, sendo unha das máquinas o Cliente e outra unha das InRow. Crearemos unha nova máquina que será a Atacante e estará na mesma rede que as demais, con dirección IP 192.168.56.10. Esta máquina será un KaliLinux 2020.1, e ten xa instaladas todas as ferramentas que precisamos.

Unha vez temos claro o método que usaremos, debemos enfrontarnos a unha complicación do programa Scapy. Anteriormente cando falabamos de envelenar a caché ARP dunha máquina, referiámonos á caché ARP do sistema; a que ten un sistema operativo por defecto. Sen embargo Scapy non utiliza esa caché; por moitas direccións IP e MAC que haxa na memoria, esta non vai ser usada por Scapy. Se queremos que Scapy use unha caché ARP deberemos implantala pola nosa conta, ben usando a clase *netcache*, ou ben creando un dicionario Python con estrutura "Dirección IP":"Dirección MAC". A clase *netcache* fai basicamente iso, crear un dicionario Python para as direccións IP e MAC, pero dado que non automatiza ningunha operación, crearemos nosoutros un dicionario normal.

7.1 Implantación de Caché ARP Scapy en ClienteCoAP e ServidorCoAP

O dicionario que imos implantar realizará a función de caché ARP para Scapy, seguindo a estrutura antes citada de "Dirección IP":"Dirección MAC". Este dicionario soamente será consultado á hora de enviar unha mensaxe CoAP por parte tanto do Cliente como dos Servidores (InRow), co cal hai que telo actualizado. Deixar claro antes de nada que este dicionario é soamente de consulta, non levaremos a cabo a resposta a solicitudes ARP nin publicacións periódicas de dirección MAC; é soamente un listado de direccións MAC que usará o programa Scapy durante a súa execución. É totalmente independente da caché ARP do sistema, que seguirá funcionando da mesma maneira ca sempre.

Para manter actualizado o noso dicionario, deberemos escoitar a rede á que está conectada a máquina. Na implantación dos servidores xa utilizamos a clase *sniff* de Scapy, pero agora verémonos na obriga de utilizala tamén na implantación do Cliente.

A clase ClienteCoAP ten como funcionalidade principal enviar mensaxes do protocolo CoAP. Se queremos que a máquina Cliente tamén escoite a redes, deberemos crear outra clase e facer que estas dúas traballen en concorrencia. Para iso, ambas as dúas clases (tanto a ClienteCoAP como a nova) deberán herdar da superclase *Thread* de Python. Esta clase crea un fío de execución de cada clase sobre a que se estenda.

Crearemos pois unha nova clase no ficheiro clienteCoAP.py que se chamará *Sniffer*, e faremos que esta e mais a clase ClienteCoAP herde da superclase *Thread*. No construtor de cadansúa clase deberán engadir a sentenza "super.__init__()", que completa o mecanismo de

herdanza ao permitirnos usar os métodos e utilidades da superclase na subclase.

Na nova clase *Sniffer*, crearemos a función *run*, herdada da superclase *Thread*, que conterá o código que se executará cada vez que se instancie a clase. Nesta función introduciremos a utilidade *sniff* de Scapy, que nos permitirá escoitar a rede. A esta función *sniff* aplicarémoslle un filtro sobre o tráfico de entrada, neste caso soamente queremos escoitar o tráfico do protocolo ARP; por tanto o filtro que lle poñeremos será *filter="arp"*. Cada vez que un paquete cumpra as condicións do filtro, executaremos a función *actualizaCache*, que comprobará se realmente é un paquete ARP, se ten código de operación 2 (o que significa que é unha mensaxe de *ARP Reply*, e contén a dirección MAC xunto coa dirección IP dunha máquina en concreto); e se é así, actualizaremos o noso dicionario da maneira que corresponda. A función quedaría da seguinte maneira:

```
1 def actualizaCache(self,packet):
2   if packet.haslayer(ARP):
3     if packet[ARP].op == 2:
4       dict = {packet[ARP].psrc: packet[ARP].hwsrc}
5       dictCache.update(dict)
```

Listing 7.1: Clase Sniffer de clienteCoAP.py

A función *update* do módulo *dict* de Python, actualiza unha entrada se existe, e se non existe crea unha nova. A variable *dictCache* é a que contén o dicionario, e vai ser usada polos dous *threads*; pola clase *Sniffer* para actualizalo e pola clase *ClienteCoAP* para lelo. Por esta razón non establecemos ningunha marca de sincronismo (nin *locks* nin semáforos), xa que un *thread* vai escribir e outro ler. Si ben é verdade que se poden producir falsas lecturas, pero é a implantación máis simple e próxima que podemos facer dunha memoria caché ARP.

Esta implantación trae cambios á hora de enviar os paquetes, xa que primeiramente teremos que buscar a dirección MAC en función da dirección IP no dicionario que fai de caché ARP. Logo, unha vez temos gardada a MAC nunha variable local, comprobaremos se é realmente unha dirección MAC ou se, pola contra, a función *get* do módulo *dict* de Python nos devolveu un None. Esta función *get* devolve o valor asociado a un identificador nun dicionario (no noso caso o identificador é a dirección IP, e o valor asociado a dirección MAC), e se non atopa ningunha coincidencia entón devolve None.

Se o valor que devolve o *get* é unha dirección MAC, entón engadiremos ao paquete CoAP que iamos enviar (coas capas IP() / UDP() / CoAP()) a capa Ehter; que engade ao paquete unha capa do protocolo *Ethernet II*, quedando da seguinte maneira Ether() / IP() / UDP() / CoAP(). Nesta nova capa Ether podemos engadir o campo *dst* e nel especificar a dirección MAC de destino, no noso caso poñeremos a dirección MAC que almacenamos no dicionario. Unha vez feito isto poderemos enviar o paquete á rede, pero non poderemos utilizar a función *send* de Scapy; esa función non é válida para enviar paquetes con especificación de capa 2 do modelo

OSI (capa de Acceso ao Medio). Para iso teremos que utilizar a función *sendp*, que se usa da mesma forma que a anterior.

Se pola contra, o valor que devolve *get* é un None, entón seguiremos usando o método de envío que tiñamos anteriormente coa función *send* de Scapy. A implantación pois quedaría así:

```
macDest = dictCache.get(ip)
  if macDest is not None:
    packetl2 = (Ether(dst=macDest) / packet) # packet é o paquete completo
    (IP() / UDP() / CoAP())
    sendp(packetl2, iface="vboxnet0")
  else:
    send(packet, iface="vboxnet0")
```

Listing 7.2: Envío de paquetes CoAP en ClienteCoAP

O método principal da clase ClienteCoAP que antes executabamos ao instanciar a clase, agora cambiarémoslle o nome a *run*, así xa se executará por defecto dado que se herda da superclase *Thread*.

Na implantación do servidor, son menos os cambios que necesitamos facer para adaptarnos ao dicionario que actúa como caché ARP. Dado que xa usabamos o método *sniff* para escoitar a rede e saber cando chegaba unha solicitude CoAP, agora simplemente podemos cambiar a función que executabamos anteriormente (*reply*) por unha nova que realice tamén a comprobación de *ARP Reply*. Dado que a sintaxe do campo *filter* da función *sniff* non pode ser complexa (como vimos dicindo), deberemos entón eliminar dito campo e así procesar todas as mensaxes de entrada. Por tanto, para todas as mensaxes de entrada executaremos a nova función *process* que, primeiro de todo comprobará se é un *ARP Reply* (código de operación = 2) e se é así actualizará o dicionario de direccións IP : MAC. Se non é un paquete ARP, entón comprobará, do mesmo xeito que antes, se ten capa CoAP e se a dirección IP de destino é a mesma que a da máquina correspondente. Se estas dúas condicións coinciden, executarase a función *reply* definida na implantación 6.2; soamente cambiándolle o xeito de enviar os paquetes que será o mesmo que o definido en 7.2 para o ClienteCoAP.

7.2 Execución de MitM entre Cliente e unha das InRow

Unha vez temos os cambios explicados anteriormente introducidos na implantación do Cliente e dos Servidores, podemos tentar agora realizar un MitM que, como ben dixemos antes, vai ser necesario para cada un dos diferentes ataques a redes IoT que probaremos. Para isto, deberemos iniciar a nosa máquina Atacante (KaliLinux) e abrir o programa *ettercap-*

graphical. Usaremos a versión gráfica de Ettercap porque é moi cómoda e intuitiva, inda que se podería usar o programa mediante a execución de comandos.

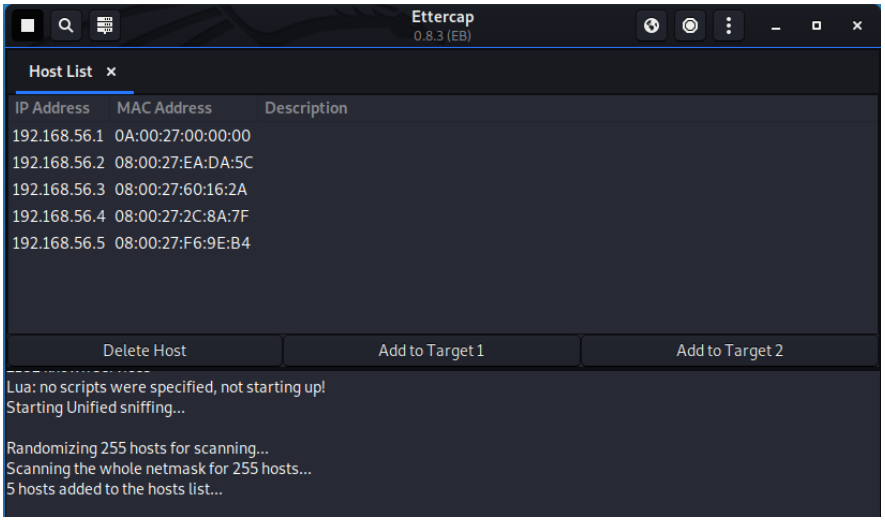


Figura 7.1: Escaneado da rede e listado de equipos na ferramenta Ettercap

Unha vez aberto o programa, seleccionaremos a interface de rede sobre a que traballaremos (no noso caso, a máquina KaliLinux usará o "eth1"); e levaremos a cabo un escaneo dos equipos que hai na rede, quedando como resultado o que vemos na imaxe 7.1 . Unha vez feito isto, escolleremos as nosas vítimas, o servidor (InRow) que queiramos e mais a máquina Cliente. Engadiremos estas dúas aos Obxectivos (*Targets*), o servidor a "Target 1" (seleccionando a súa dirección IP e logo pulsando o botón "Add to Target 1") e o Cliente a "Target 2" (seleccionando a súa dirección IP e logo pulsando o botón "Add to Target 2"). Logo iremos arriba á dereita e no menú de ataques seleccionaremos a opción *ARP poisoning*, e xa se comezaría a executar o ataque MitM. Agora todas as comunicacións que teñan lugar entre as dúas máquinas pasarán antes pola máquina Atacante, e logo serán reenviadas aos seus respectivos destinos para que todo sexa o máis opaco posible.

7.3 Ataque de Interceptación

Como ven dixemos no capítulo de Análise e Deseño referido ao entorno con ataques (5.2), realizaremos un ataque MitM do xeito que vimos en 7.2, para logo realizar un cambio na temperatura das respostas enviadas polo servidor vítima á máquina Cliente. Entón, unha vez o programa Ettercap reciba a resposta do servidor coa correspondente temperatura, haberá que cambiala por unha da nosa escolla antes de que o paquete sexa reenviado cara o Cliente. Esta tarefa pode ser levada a cabo utilizando unha utilidade da propia ferramenta Ettercap chamada *Filter*. Os *filter* ou filtros son uns scripts con unhas sentenzas escritas nun documento en

linguaxe *C*, e que se executarán para todos os paquetes que pasen a través de Ettercap. Nestes filtros non se pode realizar a típica programación estruturada e imperativa en *C*, senón que a ferramenta *Etterfilter* (pertencente ao propio paquete de Ettercap) proporciona só unhas poucas ordes que permite executar [43].

No noso caso, o script quedará da seguinte maneira:

```
if (ip.proto == UDP && ip.dst == '192.168.56.1' && udp.dst == 5683) {
   pcre_regex(DATA.data, "\d{2}\.\d{4}$", "13.5555");
   msg("Zapped Temperature");
}
```

Listing 7.3: Filtro Ettercap para ataque de Interceptación

Nel, definimos primeiramente que para cada paquete se comprobe a dirección IP de destino e o porto UDP de destino. Se a dirección IP de destino é a do Cliente e o porto UPD de destino é o porto CoAP (5683), entón aplicaremos a dito paquete unha instrución de *Etterfilter* (xa que non se poden executar outras) chamada *pcre_regex*. Ela aplica unha expresión regular aos datos do paquete (DATA.data), e se esta ten algunha coincidencia, intercámbiaa polo valor especificado a continuación (neste caso 13.5555).

A expresión regular busca os valores que teñan 2 díxitos enteiros, un símbolo "." e 4 díxitos decimais; e ademais que estea ao final dos datos do paquete ("$"). Este é o lugar inequívoco que ocupará o valor da temperatura na nosa implantación. Escollemos o valor 13.555 porque é relativamente baixo, se ben non tanto para os sensores de entrada de fluído e saída de aire, si que o é bastante para os de saída de fluído e entrada de aire. Isto provocará unha alteración na curva de temperaturas do CPD (que pasará a ser unha recta), así como unha redución do funcionamento da enfriadora exterior e un pequeno aumento da temperatura da sala. Podería realizarse o proceso inverso e aumentando o consumo enerxético se poñemos unha temperatura máis grande do normal. Unha vez realizado o cambio de temperaturas, o filtro mostrará unha mensaxe de confirmación (instrución *msg*) na consola de Ettercap.

A execución óptima deste cambio de temperatura consistiría en manter a curva. Subir ou baixar os grados devoltos polo servidor en unha ou varias unidades, pero tendo en conta a temperatura introducida no paquete e non facer unha substitución simplemente. O problema radica en que esta utilidade *Etterfilter* non permite esa implantación, sen embargo para exemplificar un ataque deste tipo é suficiente para nós.

Para que Ettercap poida facer uso dun filtro, é necesario compilar antes o arquivo en linguaxe C co contido mostrado en 7.3. Isto faise co comando *etterfilter*, pasándolle como primeiro parámetro o arquivo e logo, coa opción *"-o"*, podemos especificar o nome do ficheiro de saída. Para usar un filtro na consola gráfica de Ettercap, antes ou durante a execución dun ataque, deberemos ir ao menú e logo seleccionar a opción *Load a Filter* do apartado *Filters*. Abrirásenos un explorador de arquivos onde teremos que seleccionar o arquivo de saída do

comando *Etterfilter* executado anteriormente. Unha vez feito isto, o filtro xa estaría en uso.

7.4 Ataque de Eliminación

Este ataque executarémolo de maneira moi parecida ao anterior. Levaremos a cabo tamén un *MitM* do xeito que vimos en 7.2 e, utilizando un filtro compilado coa ferramenta *Etterfilter*, interromperemos a conectividade entre a máquina Cliente e un dos servidores (InRow). O filtro será moi sinxelo, simplemente teremos unha sentenza *if*, que comprobará que o protocolo da mensaxe sexa o UDP, que o porto destino sexa o 5683 (porto CoAP) e que a dirección IP de orixe sexa a da máquina Cliente. Se se cumpren estas condicións, faremos uso da función *drop()* proporcionada por *Etterfilter*, que descarta o paquete a procesar sen reenvialo ao seu destino. Unha vez efectuado o descarte do paquete, mostraremos unha mensaxe de confirmación na consola de Ettercap.

O código do filtro de Ettercap quedaría da seguinte maneira:

```
1 if (ip.proto == UDP && ip.src == '192.168.56.1' && udp.dst == 5683) {
2    drop();
3    msg("Drop Package\n");
4 }
```

Listing 7.4: Filtro Ettercap para ataque de Eliminación

7.5 Ataque de Duplicación

Neste caso, deberemos facer unha implantación do ataque algo mais elaborada. Non podemos utilizar o método de *MitM* e mais filtro de Ettercap, xa que non existe ningunha funcionalidade de *Etterfilter* para esta tarefa. Deste xeito, o que faremos será crear un pequeno programa de Scapy e acompañalo cun ataque *MitM* similar ao descrito en 7.2.

O código que teremos que executar consistirá nunha función *sniff* de Scapy similar á que xa implantamos anteriormente, sobre a que lle aplicaremos un filtro para os mensaxes do protocolo UDP e do porto 5683 (CoAP). Por cada unha das mensaxes que cumpra o filtro, executaremos a función *duplicate*. Esta función comproba se a dirección IP orixe é a do servidor e se a IP destino é a do Cliente, se é así, extrae a PDU do protocolo CoAP do paquete coa función *getlayer* de Scapy. Isto faise para comparar o valor do campo *options* do paquete CoAP co valor do mesmo campo do paquete anterior, que está almacenado nunha variable. É dicir, o proceso é o que segue:

- Ao comezar a execución do programa creamos unha variable chamada *packetOption* e inicializámola a un valor calquera (ex. *nonexistent*).

- Cando chega un paquete CoAP que cumpre as condicións antes sinaladas (direción IP orixe e destino), extraemos a PDU do protocolo CoAP do paquete coa función *getlayer* de Scapy.

- Comparamos a opción do identificador do sensor do campo *options* (que, por definición, sempre vai ser a identificada por [1][1] do array de opcións do campo *options*), co valor que toma a variable *packetOption*.

- Se este valor non coincide, entón enviarase o paquete usando a función *sendp* de Scapy; pero se a comparación das opcións si coincide, entón non se produce o envío.

- Finalmente actualízase o valor da variable *packetOption* ao valor do mesmo campo do paquete actual.

A ferramenta Ettercap reenviou o paquete que estamos procesando, e estando en execución este programa, enviaríase á máquina Cliente outra mensaxe igual á que xa recibiu. Toda esta implantación é necesaria para previr bucles infinitos, moi similares ás *tormentas de broadcast* que poden suceder nunha rede conmutada (*switch*). Ao enviar o mesmo paquete de volta á rede, a función *sniff* de Scapy volverá a escoitalo e a procesalo, entrando así en bucle e inundando a rede. Do xeito descrito anteriormente, inda que poida parecer pouco ortodoxo, evitamos esta situación; quedando a implantación da función *duplicate* da seguinte maneira:

```
1 def duplicate(self, packet):
2     ip = packet.getlayer(IP)
3     if ip.src == "192.168.56.X" and ip.dst == "192.168.56.1": #substituír X
      pola IP correcta da InRow
4         coap = packet.getlayer(CoAP)
5         if coap.options[1][1] != self.packetOption:
6             sendp(packet, iface=self.interface)
7         self.packetOption = coap.options[1][1]
```

Listing 7.5: Programa Scapy para reenviar paquetes no ataque de Duplicación

7.6 Probas da Implantación Con Ataques

Levaremos a cabo agora as probas sobre as implantacións descritas anteriormente de cada un dos ataques. Unha vez temos todas as máquinas funcionando (4 máquinas Ubuntu e a máquina Atacante KaliLinux), executando o comando Netcat para abrir o porto 5683 (CoAP) e o ficheiro servidorCoAP.py; entón podemos proceder a executar as probas.

7.6.1 Proba Ataque de Interceptación

Para este ataque, na máquina Atacante, arrancaremos a ferramenta Ettercap, cargaremos o filtro correspondente que vimos en 7.3 e iniciaremos Wireshark para visualizar os fluxos de paquetes que cheguen á máquina. Na máquina Cliente executaremos o ficheiro clienteCoAP.py e tamén iniciaremos a ferramenta Wireshark. Nos servidores, simplemente executaremos cadanseu ficheiro servidorCoAP.py.

Os obxectivos (*Targets*) a configurar na ferramenta Ettercap da máquina Atacante serán, o primeiro a Inrow23 (inda que podiamos escoller calquera outra) e o segundo a máquina Cliente. Iniciamos o *ARP Poisoning*, e visualizamos o Wireshark para ver se efectivamente se produce a substitución do valor das temperaturas para as mensaxes de resposta procedentes da Inrow23.

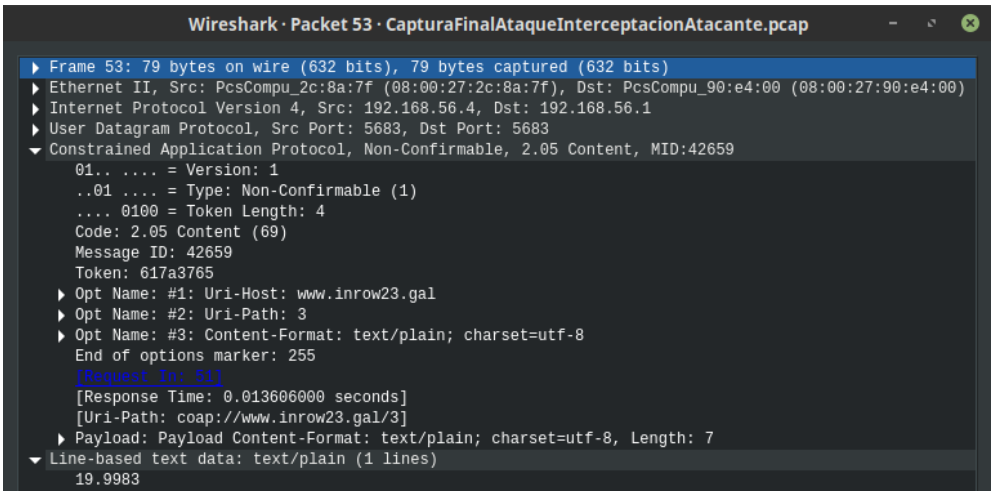


Figura 7.2: Ataque de ”Interceptación”. Paquete CoAP que provén da InRow23 a súa chegada á máquina Atacante.

Na imaxe 7.2, podemos ver o paquete de resposta que chega de ”192.168.56.4” (InRow23), e que transporta unha temperatura de 19.9983 ºC. Este é a mensaxe que recibe Ettercap xusto antes de realizar o intercambio de temperaturas e reenviar o paquete. Podemos ver que o campo ”Dst” da PDU *Ethernet II* pertencente á capa 2 do modelo OSI, ten a dirección MAC ”08:00:27:90:E4:00”, que é exactamente a dirección MAC da máquina Atacante.

Na imaxe 7.3, vemos exactamente o mesmo paquete pero logo de ser procesado por Ettercap. A temperatura foi cambiada á que establecemos por defecto durante a creación do filtro. A dirección MAC da máquina Atacante está agora no campo ”Src” da PDU de *Ethernet II*, e non no campo ”Dst”, que está agora ocupado pola dirección MAC da máquina Cliente. Foi, pois, realizada con éxito esta primeira alteración da paquetería.

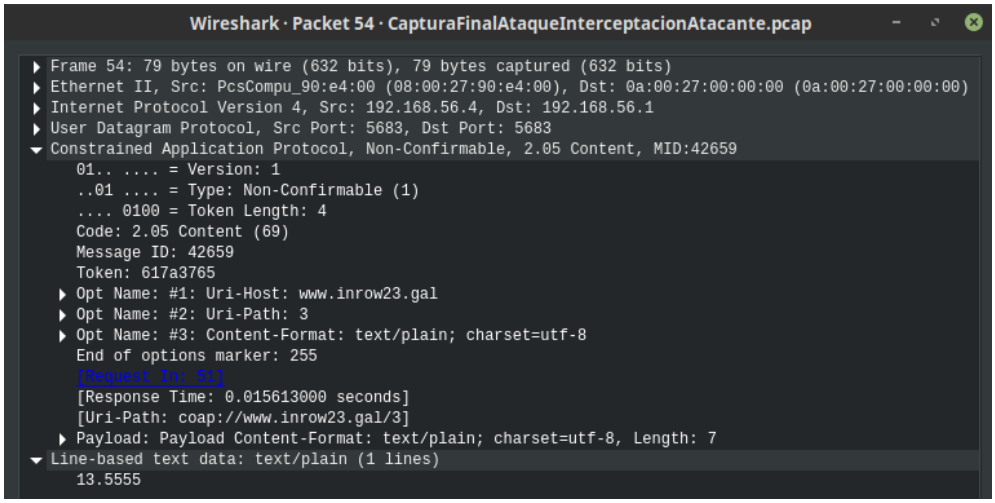


Figura 7.3: Ataque de ”Interceptación”. Paquete CoAP saínte da máquina Atacante camiño á máquina Cliente.

7.6.2 Proba Ataque de Eliminación

Neste caso, moi similar ao caso anterior, executaremos a ferramenta Ettercap na máquina Atacante e cargaremos o filtro definido para este ataque (7.4). Nos servidores executaremos os ficheiros Python correspondentes, así como na máquina Cliente onde ademais teremos o Wireshark facendo unha captura do tráfico.

A InRow obxectivo para este ataque será a InRow15, que ten a dirección IP ”192.168.56.3”. Por tanto o *Target 1* de Ettercap será dita IP e o *Target 2* a IP da máquina Cliente. Executamos o ataque *ARP Poisoning* e visualizamos a captura de tráfico que nos amosa a ferramenta Wireshark na máquina Cliente.

129	54.345874	192.168.56.1	192.168.56.2	CoAP	72 NON, MID:42877, GET, TKN:75 68 79 78, coap://www.inrow13.gal/4
130	54.354192	192.168.56.2	192.168.56.1	CoAP	79 NON, MID:59674, 2.05 Content, TKN:75 68 79 78, coap://www.inrow13.gal/4 (text/plain)
131	55.407696	192.168.56.1	192.168.56.3	CoAP	72 NON, MID:42878, GET, TKN:6f 64 43 56, coap://www.inrow15.gal/1
132	56.464727	192.168.56.1	192.168.56.3	CoAP	72 NON, MID:42879, GET, TKN:62 6b 4b 77, coap://www.inrow15.gal/2
133	57.538244	192.168.56.1	192.168.56.3	CoAP	72 NON, MID:42880, GET, TKN:4f 76 41 6d, coap://www.inrow15.gal/3
134	57.560316	PcsCompu_90:e4:00	0a:00:27:00:00:00	ARP	60 192.168.56.3 is at 08:00:27:90:e4:00
135	58.620465	192.168.56.1	192.168.56.3	CoAP	72 NON, MID:42881, GET, TKN:76 59 79 67, coap://www.inrow15.gal/4
136	59.665370	192.168.56.1	192.168.56.4	CoAP	72 NON, MID:42882, GET, TKN:52 69 6c 43, coap://www.inrow23.gal/1
137	59.674041	192.168.56.4	192.168.56.1	CoAP	79 NON, MID:2021, 2.05 Content, TKN:52 69 6c 43, coap://www.inrow23.gal/1 (text/plain)

Figura 7.4: Ataque de ”Eliminación”. Mensaxes de solicitude da máquina Cliente sen resposta.

Na figura 7.4 podemos ver como as mensaxes de solicitude da máquina Cliente cara a InRow15 non teñen resposta. Por tanto, o ataque está a realizarse con éxito.

7.6.3 Proba Ataque de Duplicación

Para este ataque de ”Duplicación”, na máquina Atacante levaremos a cabo un MitM do xeito explicado en 7.2 e que terá como obxectivos, *Target 1* a InRow25 cuxa dirección IP é a ”192.168.56.5”, e *Target 2* a máquina Cliente. Nos servidores executaremos os ficheiros Python correspondentes, así como na máquina Cliente onde ademais teremos o Wireshark facendo unha captura do tráfico.

Unha vez estea todo o anterior funcionando, iniciaremos o ataque *ARP Poisoning* na ferramenta Ettercap da máquina Atacante e, ademais, executaremos o programa que contén a función vista en 7.5 e que leva a cabo o reenvío duplicado das mensaxes.

```
116 53.038447   192.168.56.1   192.168.56.5   CoAP   72 NON, MID:38296, GET, TKN:5a 38 34 5a, coap://www.inrow25.gal/1
117 53.055801   192.168.56.5   192.168.56.1   CoAP   79 NON, MID:60996, 2.05 Content, TKN:5a 38 34 5a, coap://www.inrow25.gal/1 (text/plain)
118 53.055898   192.168.56.5   192.168.56.1   CoAP   79 NON, MID:60996, 2.05 Content, TKN:5a 38 34 5a, coap://www.inrow25.gal/1 (text/plain)
119 54.118867   192.168.56.1   192.168.56.5   CoAP   72 NON, MID:38297, GET, TKN:6e 52 50 78, coap://www.inrow25.gal/2
120 54.137247   192.168.56.5   192.168.56.1   CoAP   79 NON, MID:45575, 2.05 Content, TKN:6e 52 50 78, coap://www.inrow25.gal/2 (text/plain)
121 54.137403   192.168.56.5   192.168.56.1   CoAP   79 NON, MID:45575, 2.05 Content, TKN:6e 52 50 78, coap://www.inrow25.gal/2 (text/plain)
122 55.190354   192.168.56.1   192.168.56.5   CoAP   72 NON, MID:38298, GET, TKN:6e 6c 66 5a, coap://www.inrow25.gal/3
123 55.207289   192.168.56.5   192.168.56.1   CoAP   79 NON, MID:5529, 2.05 Content, TKN:6e 6c 66 5a, coap://www.inrow25.gal/3 (text/plain)
124 55.212814   192.168.56.5   192.168.56.1   CoAP   79 NON, MID:5529, 2.05 Content, TKN:6e 6c 66 5a, coap://www.inrow25.gal/3 (text/plain)
125 56.274891   192.168.56.1   192.168.56.5   CoAP   72 NON, MID:38299, GET, TKN:75 51 38 53, coap://www.inrow25.gal/4
126 56.294932   192.168.56.5   192.168.56.1   CoAP   79 NON, MID:41422, 2.05 Content, TKN:75 51 38 53, coap://www.inrow25.gal/4 (text/plain)
127 56.295112   192.168.56.5   192.168.56.1   CoAP   79 NON, MID:41422, 2.05 Content, TKN:75 51 38 53, coap://www.inrow25.gal/4 (text/plain)
```

Figura 7.5: Ataque de ”Duplicación”. Mensaxes de resposta da InRow25 duplicados.

Na imaxe 7.5, podemos ver como todas as respostas que proveñen da dirección ”192.168.56.5” (InRow25) son recibidas pola máquina Cliente de xeito duplicado; o que confirma a correcta execución desta alteración no tráfico.

7.7 Execución da Simulación Final e Creación do Dataset

Unha vez que o noso entorno ao completo estea funcionando e executándose segundo todo o definido e deseñado, así como que todas as alteracións da paquetería (ataques) se poidan levar a cabo correctamente; entón podemos proceder a realizar a Simulación Final e a Captura do Dataset, que é o cumio deste traballo. Para isto, seguiremos o guión descrito en 5.2.2, tendo en conta as horas de comezo e final de cada paso. O servidor (InRow) que imos atacar será InRow15 (inda que puido escollerse calquera dos outros), cuxa dirección IP é a ”192.168.56.3”.

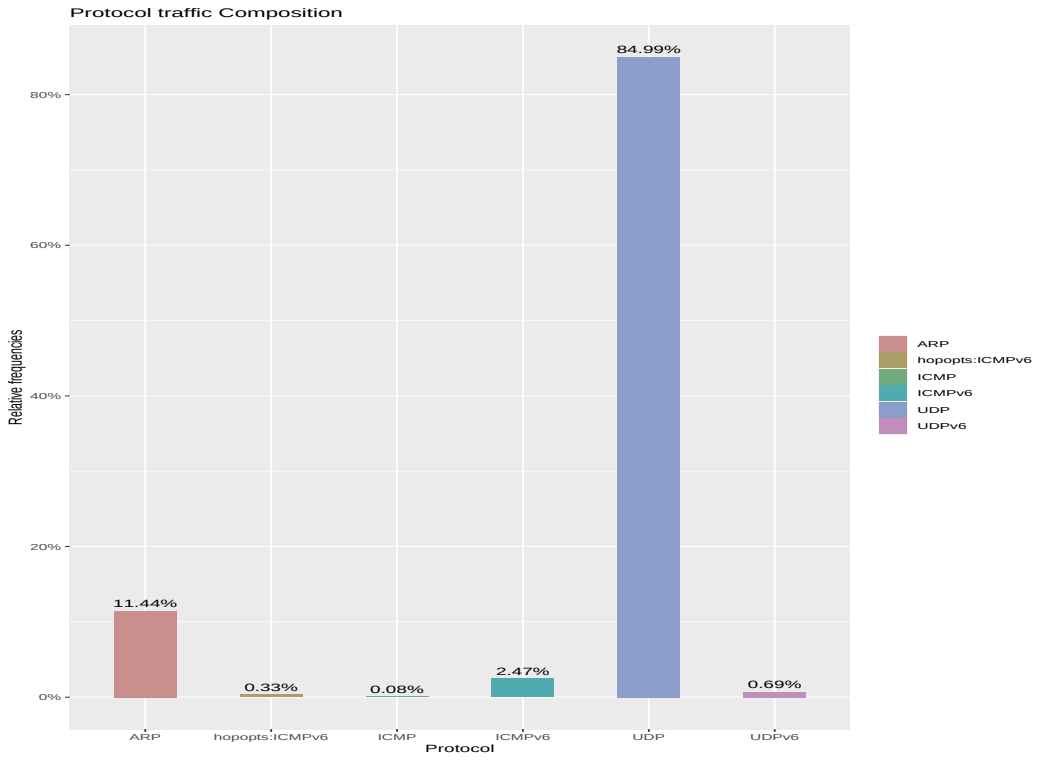
Chegados a este punto, e con todas as situacións que se exemplificarán durante a simulación probadas (tanto as de fluxos normais como alterados), non deberiamos ter ningún problema á hora de levar a cabo esta tarefa. Ao final da execución da simulación, gardaremos esta nun ficheiro con extensión ”.pcap”, que pode ser lido e aberto pola maioría de software analizador de paquetes. Tamén crearemos o arquivo de etiquetado da captura e gardarémolo, tal e como dixemos, con extensión ”.csv”.

Capítulo 8

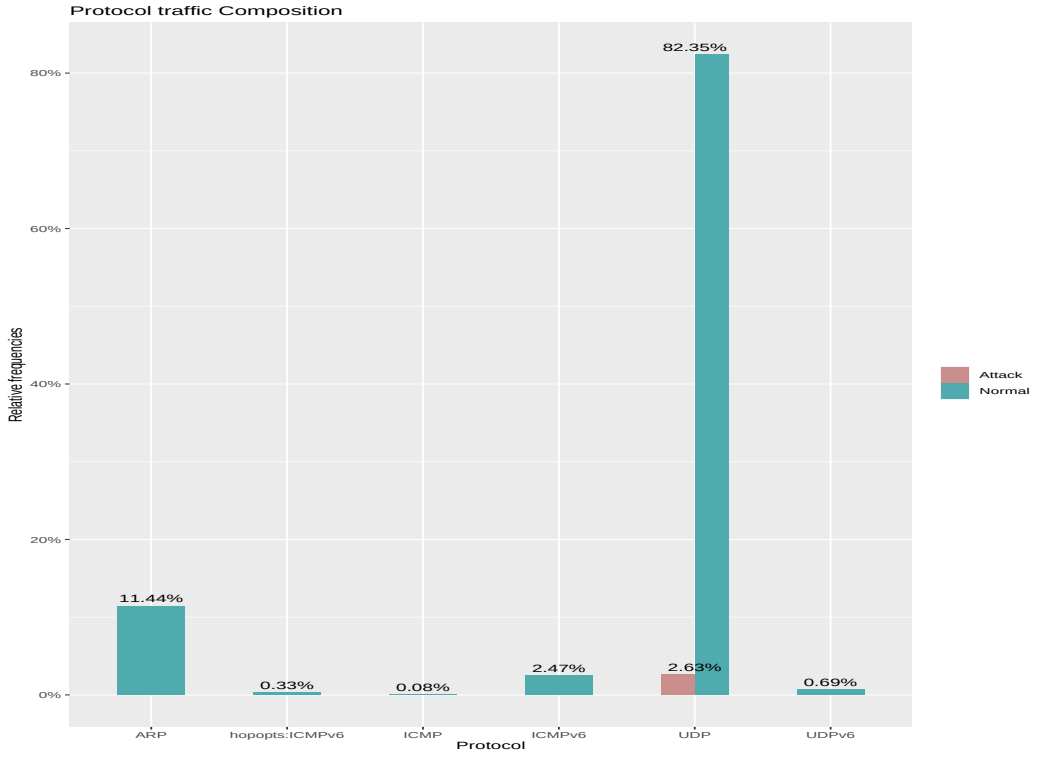
Análise Gráfico do Dataset

Logo de crear o *Dataset* anotado de maneira satisfactoria, podemos pasar agora a realizar unha análise estatística do seu contido. Para esta tarefa empregaremos a ferramenta *RStudio*, así coma unhas secuencias de instrucións de ”R” agrupadas en arquivos. Estes conxuntos de instrucións darán como resultado uns gráficos sobre os que realizaremos os comentarios oportunos. Dicir tamén que os scripts en código ”R” foron desenvoltos polo Grupo de Investigación Telemática da UDC, e usados para análises de *datasets* nas súas investigacións. Por tanto, o que fixemos neste traballo foi reutilizar ditos conxuntos de instrucións e aplicalos ao noso *Dataset* resultante.

Unha vez importado o noso *Dataset* e separados os seus campos, puidemos extraer a información máis relevante e representala en gráficas para poder visualizar máis facilmente o resultado da análise. Comezaremos amosando as gráficas de porcentaxe de protocolos (véxase figura 8.1). Na gráfica 8.1a podemos observar a porcentaxe dos distintos protocolos que foron capturados durante a simulación final e que están incluídos no *Dataset*. A grande maioría é tráfico UDP, seguido de ARP; tal e como se esperaba. Hai que ter en conta que no noso entorno simulado soamente tiñamos programados fluxos de datos do protocolo CoAP, por iso se produce esta alta porcentaxe de paquetes UDP. Tamén se produciron as solicitudes ARP necesarias para levar a cabo as comunicacións entre dispositivos, de aí a que ocupe o segundo lugar como protocolo máis común do *Dataset*. O resto do tráfico débese a comportamentos espontáneos das máquinas, gran parte del debido á máquina Cliente xa que é unha máquina física cun sistema operativo máis complexo que o dos servidores. Na figura 8.1b vemos as mesmas porcentaxes que na gráfica anterior, soamente que neste caso fixemos a diferenza entre os paquetes que pertencen a ataques e os pertencentes a comunicación normal. Como vemos, soamente hai un 2.63% de paquetes UDP utilizados nos ataques, fronte a un 82.35% de paquetes UDP de comunicación normal. Esta proporción correspóndese coa esperada, xa que simplemente se producen 3 horas de ataques a unha máquina (máquina vítima), fronte a 24 horas de comportamento normal en 3 máquinas e 21 na máquina vítima.

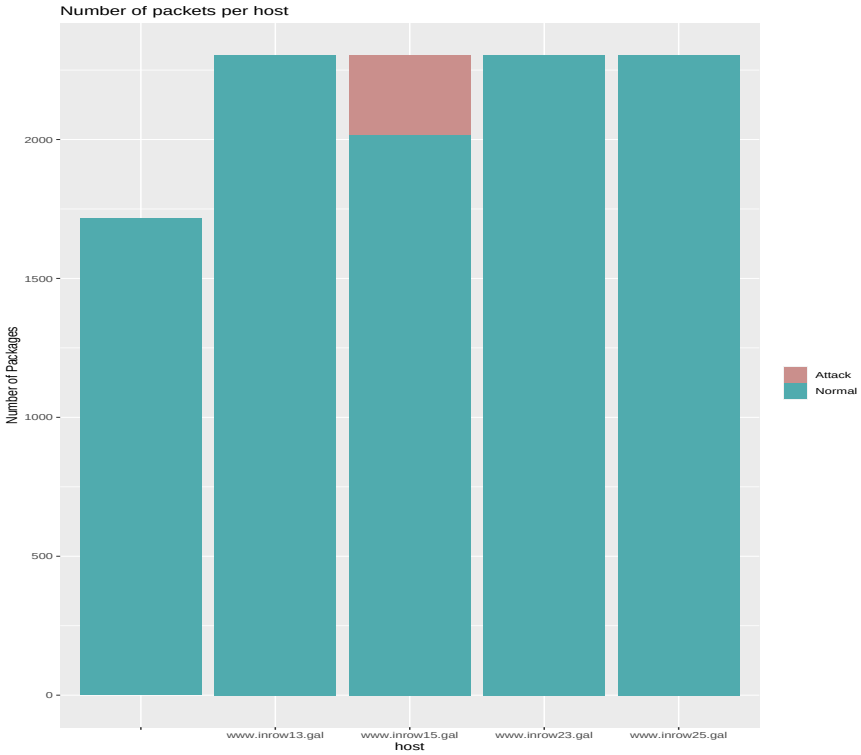


(a) Porcentaxe de protocolos capturados durante a simulación.

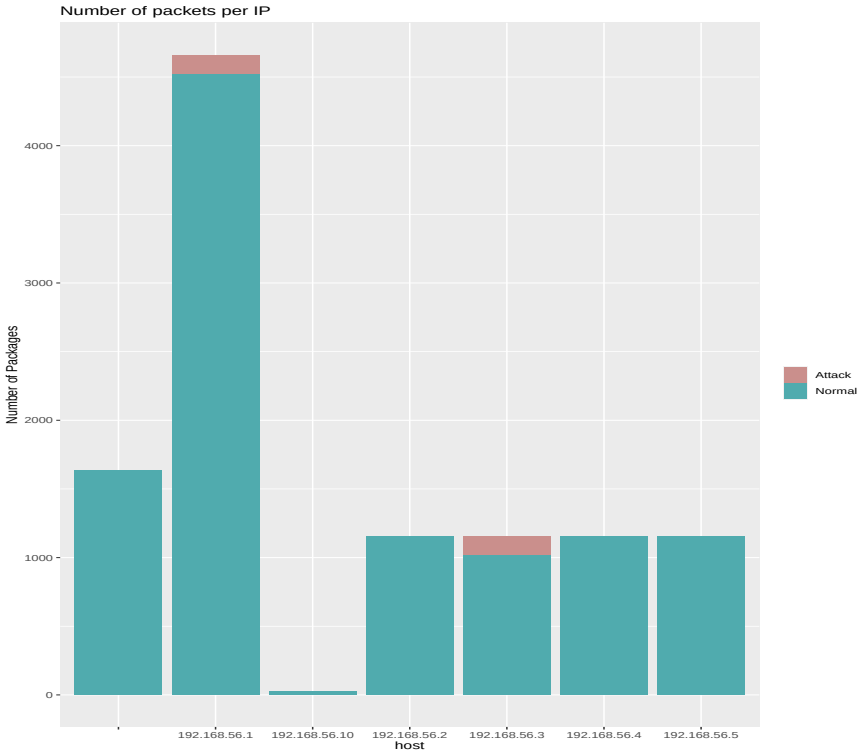


(b) Porcentaxe de paquetes pertencentes a ataques no total da simulación.

Figura 8.1: Gráficos dos diferentes protocolos do Dataset



(a) Número de paquetes CoAP enviados e recibidos por cada URI.



(b) Número de paquetes enviados por cada dirección IP.

Figura 8.2: Gráficos da cantidade de paquetes

Pasamos agora as gráficas que mostran a cantidade de paquetes enviados e recibidos en función de cada máquina (véxase figura 8.2). Na figura 8.2a, podemos ver a cantidade de paquetes enviados e recibidos por cada servidor (InRow). Entre eles diferéncianse tamén os que son de ataques, todos do servidor *www.inrow15.gal*; o que é correcto. Non se amosan os paquetes da máquina Cliente debido a que non ten identificador URI, simplemente é unha máquina solicitando recursos periodicamente aos servidores (non confundir coa primeira columna da gráfica).

Na figura 8.2b podemos ver os paquetes enviados por cada dirección IP que aparece no *Dataset*. A diferenza da anterior gráfica, agora si se mostran columnas para a dirección IP da máquina Cliente e para a máquina Atacante. Pode parecer estraño que a columna correspondente á dirección IP da máquina Atacante (”192.168.56.10”) non mostre ningún envío de paquetes pertencentes a ataques, como si o fan as columnas da InRow15 (”192.168.56.3”) e da máquina Cliente (”192.168.56.1”). Sen embargo isto é certo, para o protocolo IP. O *Proxy UDP* que creamos na máquina Atacante entre a máquina Cliente e a InRow15, establécese debido a envelenamento da caché ARP e das direccións MAC que ela contén. Estas dirección MAC pertencen á capa 2 do modelo OSI (Capa de Enlace de Datos) e o protocolo IP pertence á capa 3 (Capa de Rede). Por tanto, a nivel de capa 3, ningún paquete con orixe a máquina Cliente e destino a InRow15, ou viceversa, pasou pola máquina Atacante durante a simulación final. De aí que se amose a IP da máquina Atacante con tan poucos paquetes enviados. Nesta gráfica tamén podemos ver que a cantidade de paquetes enviados pola máquina Cliente é moito maior que os de calquera servidor, aproximadamente a suma dos paquetes enviados polos catro servidores tería como resultado os paquetes enviados do Cliente; o cal entendemos que é correcto.

Nas dúas gráficas da figura 8.2 aparece unha primeira columna sen lenda ao pé. Isto débese á existencia de paquetes que non teñen orixe ou destino coñecido, ou que non pertencen ao protocolo IPv4. Existe tráfico de solicitudes IPv6, así como os paquetes de solicitude ARP dirixidos á dirección de *Broadcast*,...

Capítulo 9

Conclusións

Como conclusión deste traballo, podemos dicir que o protocolo CoAP presenta unhas características e unhas capacidades de configuración que, sen dúbida, o converterán nun dos protocolos IoT máis usados nos próximos anos. Hai que ter en conta que soamente nos baseamos na configuración de CoAP sobre UDP e en modo asíncrono, todo o relativo a esta configuración está redactado no RFC7252 (sobre o que nos apoiamos enormemente neste traballo). Sen embargo, existe a configuración de CoAP sobre TCP e utilizando TLS e outras opcións de seguridade, opcións que de feito abranguen outro manual completo chamado RFC8323. Este RFC amosa toda a configuración de CoAP sobre TCP, TLS e *WebSockets*, para o seu uso en comunicacións confiables; non é unha versión nova de CoAP (segue sendo a versión 1). Resulta evidente, pois, que a capacidade de expansión e adaptabilidade de CoAP son un grande atractivo para as novas redes e implantacións IoT.

En relación coa elaboración deste traballo, resultou un problema atopar capturas de tráfico do protocolo CoAP que, aparentemente, son bastante escasas. Como consecuencia, tivemos que consultar unha cantidade grande de documentación (libros, artigos...), da que conseguimos extraer a información necesaria para poder deseñar en detalle o comportamento do protocolo. En canto ás ferramentas utilizadas, existía unha grande variedade de opcións; na súa escolla baseámonos, na maioría dos casos, polas máis coñecidas e que mellor se adaptaran aos nosos obxectivos.

No que concerne á programación do deseño, resaltar que a ferramenta Scapy conta cunha gran versatilidade e potencia. A capacidade de poder traballar directamente cos campos dos paquetes de case calquera protocolo de rede, así como todos os módulos e funcións para escoitar a rede e extraer información dos paquetes, resulta verdadeiramente útil e práctico. Si ben é certo que na implantación dos distintos tipos de ataques tivemos que resolver o problema derivado da caché ARP, o que nos supuxo un incremento de código e funcións necesarias; pero unha vez adaptados os arquivos Python, non se produciu ningunha alteración do comportamento.

Sobre os ataques á rede IoT, especificar de novo que o que amosamos neste traballo, non se corresponde exactamente con ataques. Utilizamos esa nomenclatura por resumir e darnos a entender. Un ataque, realmente, implica un proceso previo de análise e recompilación de información sobre o entorno (dispositivos e rede). Unha vez feito isto, a persoa atacante podería decidir se proceder a realizar alteracións no tráfico ou nos dispositivos. Na nosa simulación, exemplificamos alteracións e anomalías sobre o tráfico de rede similares ás que se poden producir nun ataque propiamente dito; para facer que así a IA poida aprender a diferenza entre un comportamento normal e un ataque real, que é o obxectivo da creación do noso Dataset.

A simulación final, tal e como dixemos, foi creada seguindo un guión e un horario previamente definido. Unha das cousas que se poderían mellorar sobre a súa realización sería incrementar a automatización do proceso. No noso caso, xustificamos esta execución manual da captura e dos ataques realizados argumentando que sería unha execución máis supervisada, e con maior control do que está a suceder. Sen embargo, unha vez exemplificado e probado cada comportamento, tanto normal como anómalo, antes de realizar a captura final sería posible, por exemplo, executar todos os ataques empregando o modo comando e non a interface gráfica da ferramenta Ettercap. Deste xeito, poderiamos programar cada execución do ataque en función do horario a seguir, empregando a ferramenta de planificación de Linux chamada *cron*.

Para rematar, indicar que se realizaron e cumpriron correctamente todos os obxectivos expostos ao inicio deste proxecto. Con isto, conseguimos obter un Dataset anotado do protocolo CoAP, preparado para ser utilizado durante a execución da aprendizaxe máquina dunha IA. Como traballo futuro, podería implantarse un comportamento similar ao descrito nesta memoria, pero empregando conexións TCP e seguridade TLS. Deste xeito, tamén se podería estudar o impacto computacional que tería esta configuración en dispositivos con hardware limitado e, da mesma forma, poder crear un Dataset anotado de tráfico CoAP sobre TCP; algo aínda máis escaso, se cabe, que o exemplificado e probado neste proxecto.

Relación de Acrónimos

IoT *Internet of Things*

RAM *Random Access Memory*

ISO *International Organization for Standardization*

LLN *Low Power and Lossy Network*

HTTP *Hypertext Transfer Protocol*

CoAP *Constrained Application Protocol*

LLN *Low Power and Lossy Networks*

IETF *Internet Engineering Task Force*

UDP *User Datagram Protocol*

TCP *Transmission Control Protocol*

MAC *Media Access Control*

CSMA *Carrier Sense Multiple Access*

TLS *Transport Layer Security*

DTLS *Datagram Transport Layer Security*

REST *REpresentational State Transfer*

CPD *Centro de Procesamento de Datos*

HACS *Hot Aisle Containment System*

PDU *Protocol Data Unit*

MitM *Man-in-the-Middle*

Glosario

Sensor Dispositivo capaz de detectar magnitudes físicas ou químicas (temperatura, humidade...), e transformalas en variables eléctricas.

Modelo OSI É un conxunto de estándares ISO relativo á interconexión dos sistemas de comunicacións publicado en 1980. Está dividido en 7 capas.

Modelo TCP/IP Versión do Modelo OSI que soamente conta con 4 capas (Aplicación, Transporte, Rede e Enlace de datos). É o modelo mais usado, pois o Modelo OSI é puramente teórico.

Chip É unha pastilla pequena de material semicondutor, dalgúns milímetros cadrados de área, sobre a que se fabrican circuítos electrónicos e que está protexida dentro dun encapsulado de plástico ou cerámica.

Proxy É unha máquina que fai de intermediaria entre as peticións realizadas por un cliente a un servidor. Nesta máquina intermediaria pódense levar a cabo diferentes funcionalidades como control de acceso, rexistro do tráfico, restrición de determinados tipos de tráfico, mellora do rendemento, anonimato da comunicación, caché web...

Linguaxe Interpretada Linguaxe na que se executan as súas instrucións sen unha previa compilación do programa a linguaxe máquina.

Tipado Dinámico Unha linguaxe de programación é dinámicamente tipado se unha variable pode tomar valores de distinto tipo.

Shell Intérprete de comandos.

Man-in-the-Middle denomínase ataque do ”home no medio” ao ataque onde o atacante retransmite en segredo, e posiblemente altera a comunicación, entre dúas partes que cren estar comunicándose directamente entre si mediante unha conexión privada.

Sniffer Programa destinado á captura de datos dunha rede informática.

Bibliografía

[1] Oracle®, "¿qué es iot?" 2020. [En liña]. Dispoñible en: https://www.oracle.com/es/internet-of-things/what-is-iot.html

[2] I. P. Sáez, "Iot: protocolos de comunicación, ataques y recomendaciones," Febreiro 2019. [En liña]. Dispoñible en: https://www.incibe-cert.es/blog/iot-protocolos-comunicacion-ataques-y-recomendaciones

[3] C. C. Mónica Martí, Carlos Garcia-Rubio, "Performance evaluation of coap and mqtt_sn in an iot environment," *MDPI*, vol. 31, no. 49, p. 12, Novembro 2019.

[4] M. Rouse, "Internet de las cosas (iot)," Xaneiro 2017. [En liña]. Dispoñible en: https://searchdatacenter.techtarget.com/es/definicion/Internet-de-las-cosas-IoT

[5] Wikipedia®, "Sensor," Xuño 2019. [En liña]. Dispoñible en: https://gl.wikipedia.org/wiki/Sensor

[6] V. M. M. A. F. J. A. O. V. C. Dubravko Ćulibrk, Dejan Vukobratovic, "Processing power and memory," in *Sensing Technologies For Precision Irrigation.* Springer, 2014, ch. 1, p. 3.

[7] M. A. M. I. G. A. J. C. J. M. A. Andrés Mejías, Reyes S. Herrera, "Easy handling of sensors and actuators over tcp/ip networks by open source hardware/software," *MDPI*, vol. 17, no. 94, p. 23, Xaneiro 2017.

[8] J. Vasseur, *Terms Used in Routing for Low-Power and Lossy Networks*, IETF, Xaneiro 2014.

[9] M. P. L. V. Simone Cirani, Gianluigi Ferrari, ""traditional" internet review," in *Internet of Things: Architectures, Protocols and Standards.* WILEY, 2019, ch. 2, p. 9.

[10] ——, "Physical/link layer," in *Internet of Things: Architectures, Protocols and Standards.* WILEY, 2019, ch. 2, p. 28.

[11] ——, "Network layer," in *Internet of Things: Architectures, Protocols and Standards.* WILEY, 2019, ch. 2, p. 33.

[12] ——, "Transport layer," in *Internet of Things: Architectures, Protocols and Standards.* WILEY, 2019, ch. 2, p. 34.

[13] ——, "Application layer," in *Internet of Things: Architectures, Protocols and Standards.* WILEY, 2019, ch. 2, pp. 34–36.

[14] P. Singh, "Application protocols for iot," Maio 2016. [En liña]. Dispoñible en: https://iotbytes.wordpress.com/application-protocols-for-iot/

[15] D. A.-z. Muneer Bani Yassein, Mohammed Q. Shatnawi, "Application layer protocols for the internet of things: A survey," in *Conference: IEEE International Conference on Internet of Things and Pervasive Systems*, Setembro 2016, pp. 1–4.

[16] M. H. R. Hussein T. Mouftah, Melike Erol-Kantarci, "Modbus," in *Transportation and Power Grid in Smart Cities.* WILEY, 2019, ch. 2.3.1, pp. 27–28.

[17] *MODBUS Application Protocol Specifications*, Modbus Organization, Abril 2012.

[18] *MODBUS/TCP Security*, Modbus Organization, Xullo 2018.

[19] K. H.-C. B. Z. Shelby, ARM, *The Constrained Application Protocol (CoAP).* Universitaet Bremen TZI: IETF, Xuño 2014, vol. RFC 7252, rfc 1, p. 5.

[20] ——, *The Constrained Application Protocol (CoAP).* Universitaet Bremen TZI: IETF, Xuño 2014, vol. RFC 7252, ch. 2, p. 10.

[21] ——, *The Constrained Application Protocol (CoAP).* Universitaet Bremen TZI: IETF, Xuño 2014, vol. RFC 7252, ch. 12.6, p. 95.

[22] ——, *The Constrained Application Protocol (CoAP).* Universitaet Bremen TZI: IETF, Xuño 2014, vol. RFC 7252, ch. 2.1, p. 11.

[23] ——, *The Constrained Application Protocol (CoAP).* Universitaet Bremen TZI: IETF, Xuño 2014, vol. RFC 7252, ch. 5.2.3, p. 34.

[24] ——, *The Constrained Application Protocol (CoAP).* Universitaet Bremen TZI: IETF, Xuño 2014, vol. RFC 7252, ch. 2.2, p. 12.

[25] ——, *The Constrained Application Protocol (CoAP).* Universitaet Bremen TZI: IETF, Xuño 2014, vol. RFC 7252, ch. 5.8, p. 47.

[26] ——, *The Constrained Application Protocol (CoAP).* Universitaet Bremen TZI: IETF, Xuño 2014, vol. RFC 7252, ch. 12.1.2, p. 88.

[27] ——, *The Constrained Application Protocol (CoAP).* Universitaet Bremen TZI: IETF, Xuño 2014, vol. RFC 7252, ch. 5.2.1, p. 33.

[28] ——, *The Constrained Application Protocol (CoAP).* Universitaet Bremen TZI: IETF, Xuño 2014, vol. RFC 7252, ch. 3, p. 15.

[29] ——, *The Constrained Application Protocol (CoAP).* Universitaet Bremen TZI: IETF, Xuño 2014, vol. RFC 7252, ch. 4.4, p. 24.

[30] ——, *The Constrained Application Protocol (CoAP).* Universitaet Bremen TZI: IETF, Xuño 2014, vol. RFC 7252, ch. 5.3.1, pp. 34–35.

[31] ——, *The Constrained Application Protocol (CoAP).* Universitaet Bremen TZI: IETF, Xuño 2014, vol. RFC 7252, ch. 5.10, pp. 52–59.

[32] ——, *The Constrained Application Protocol (CoAP).* Universitaet Bremen TZI: IETF, Xuño 2014, vol. RFC 7252, ch. 6.5, p. 62.

[33] M. B. Otmane El Mouaatamid, Mohammed Lahmer, "Internet of things security: Layered classification of attacks and possible countermeasures," *Revue e-Ti*, no. 9, pp. 1–14, 2016.

[34] Wikipedia®, "Eavesdropping," Agosto 2019. [En liña]. Dispoñible en: https://es.wikipedia.org/wiki/Eavesdropping

[35] OWASP, "Web parameter tampering," Febreiro 2020. [En liña]. Dispoñible en: https://owasp.org/www-community/attacks/Web_Parameter_Tampering

[36] Wikipedia®, "Ataque sybil," Xaneiro 2020. [En liña]. Dispoñible en: https://es.wikipedia.org/wiki/Ataque_Sybil

[37] ——, "Python," Maio 2020. [En liña]. Dispoñible en: https://es.wikipedia.org/wiki/Python

[38] P. Biondi, "Introduction about scapy," 2020. [En liña]. Dispoñible en: https://scapy.readthedocs.io/en/latest/introduction.html

[39] Wikipedia®, "Virtualbox," Xuño 2020. [En liña]. Dispoñible en: https://en.wikipedia.org/wiki/VirtualBox

[40] ——, "Netcat," Agosto 2019. [En liña]. Dispoñible en: https://es.wikipedia.org/wiki/Netcat

[41] ——, "Ettercap," Marzo 2020. [En liña]. Dispoñible en: https://en.wikipedia.org/wiki/Ettercap_(software)

[42] ——, "Wireshark," Maio 2020. [En liña]. Dispoñible en: https://en.wikipedia.org/wiki/Wireshark

[43] M. V. Alberto Ornaghi, "etterfilter(8)." [En liña]. Dispoñible en: https://linux.die.net/man/8/etterfilter