UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

TRABAJO FIN DE GRADO

DESARROLLO DE UN MÓDULO DE SEGURIDAD PARA NODOS DE AGREGACIÓN EMPLEADOS EN INTERNET DE LAS COSAS

Irene Nicolás Jiménez
Tutor: Lluís Gifre Renom
Ponente: Jorge Enrique López de Vergara Méndez

JUNIO 2019

DESARROLLO DE UN MÓDULO DE SEGURIDAD PARA NODOS DE AGREGACIÓN EMPLEADOS EN INTERNET DE LAS COSAS

AUTOR: Irene Nicolás Jiménez
TUTOR: Lluís Gifre Renom
PONENTE: Jorge Enrique López de Vergara Méndez

Computación y Redes de altas prestaciones
Dpto. Tecnología Electrónica y de las Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio 2019

Resumen (castellano)

El uso de Internet se ha ido extendiendo en los últimos años a todo tipo de empresas y hogares de los usuarios. Por este motivo, se ha convertido en una necesidad el tener un control sobre los datos que son transportados por la red ya que pueden contener información de carácter personal y/o confidencial que en manos ajenas puede dañar seriamente la integridad de los usuarios y de los negocios.

Con la aparición de nuevos procesos de automatización y control, surge un nuevo caso de uso de Internet que consiste en utilizarlo para conectar diferentes máquinas ente sí para que puedan comunicarse, dando lugar a lo que se conoce como Internet de las Cosas (Internet of Things, IoT). Un caso de uso particular es la conexión de pequeños aparatos electrónicos de recolección de datos y actuación en respuesta a los datos recogidos. Los datos pueden ser desde datos del entorno hasta datos médicos personales recogidos de una manera automática y sistematizada facilitando innumerables tareas.

La motivación de este Trabajo Final de Grado surge de la necesidad de proporcionar una capa de seguridad ante un escenario en el que nodos de agregación, empleados para unificar y agregar datos de distintos sensores cercanos, pueden ser víctimas de un ataque de Denegación de Servicio (Denial of Service, DoS). El principal objetivo es diseñar un módulo que sea capaz de detectar un ataque de este tipo y actuar en consecuencia para así evitar que un nodo comprometido pueda perjudicar al resto del sistema. De esta manera si un ataque de DoS es detectado, el sistema podría reconfigurar su cortafuegos (firewall) para poder paralizar el ataque y mantener a salvo los equipos con sus respectivas informaciones.

Para poder realizar este proyecto se ha decidido implementar el sistema empleando tecnologías de código abierto que permiten capturar el tráfico recibido, en concreto la herramienta TShark. Esto se ha enlazado con un algoritmo de detección de eventos frecuentes en el análisis del tráfico denominado Heavy Hitters y una interfaz de programación para el módulo de seguridad de red IPTables en Linux que nos va a permitir reconfigurar las reglas de entradas de paquetes. Una vez terminado el módulo se realizarán diferentes pruebas para poder validar el correcto funcionamiento del sistema.

Palabras clave (castellano)

Cortafuegos, Heavy Hitters, DoS, IoT.

Abstract (English)

The use of the Internet has been growing so much lately that it has managed to enter companies and user's homes. This is why a requirement to be in control of all data transported in the network has become a must and also because it may contain personal and/or confidential information. This information in someone else's hands can seriously harm users' integrity or even business integrity.

With new automation and control processes appearing, a new use of the Internet which consists of using it to link different machines up so that they can communicate among themselves. This is what we call Internet of Things (IoT). A particular case is the connection of little electronic devices which are capable of gathering data and acting according to this data. It can be from environmental data to personal medical information and it is gathered in an automatic and systematized way making multiple tasks easier.

The motivation of this end-of-degree project arises due to the need to supply a security layer in a scenario where aggregation nodes are used to unify and aggregate data from close sensors, they can be attacked by a Denial of Service (DoS) attack. That is why the main goal is to design a module capable of detecting an attack of this kind and acting to avoid a compromised node from affecting the entire system. So, if a DoS attack is detected, the system is going to be able to reconfigure the firewall to stop the attack and also manage to keep the equipment safe where all the data and information has been stored.

In order to carry out this project, it has been decided to implement the system by using open source technologies that allow us to capture the traffic received, more precisely, a terminal oriented to Wireshark named TShark. It has been connected to a frequent error detection algorithm in the traffic test called Heavy Hitters and also to a programming interface to our network safety module IPTables in Linux. This is going to allow us to reconfigure the input rules from the firewall. Once finished the whole module, we are going to run some tests to be able to validate the correct system operation.

Keywords (inglés)

Firewall, IPTables, Heavy Hitters, DoS, IoT.

Agradecimientos

Con la entrega de este proyecto llega el momento de decir adiós a una de las experiencias más importantes de mi vida, a unos años duros y llenos de esfuerzos y sacrificios, pero también de buenos momentos y de personas sin las que llegar hasta aquí no hubiera sido posible.

En primer lugar, a mis padres y mi hermano, gracias por aguantarme esos días de nervios, de cambios de humor y de altibajos. Por creer en mi aun cuando ni yo misma lo hacía y no permitirme nunca tirar la toalla. Por enseñarme que en la vida el que algo quiere algo le cuesta, como bien dice mi abuela. Por siempre estar a mi lado en los días buenos y también en los malos, lo que hace que este camino sea un poco más ameno. Gracias por vuestro amor y vuestro apoyo incondicional.

A todas las personas que he conocido a lo largo de estos años, que sin duda se han llegado a convertir en familia. Tiempos en los que hemos compartido más horas de biblioteca y laboratorios que tiempo en casa. Me gustaría agradecer en especial a dos de esas personas, María y Mamen, dos personitas que han estado a mi lado y con las que he compartido esta montaña rusa de carrera y a las que hoy puedo decir que son parte de mi familia.

A los de siempre, porque sin ellos, sin ese rato de desconectar, de desahogarnos, aunque sea media hora no hubiera sido lo mismo. Porque esos momentos de risas y ánimos son impagables.

Por último, me gustaría darle las gracias a mi tutor, Lluís Gifre. Gracias por darme la oportunidad de realizar este trabajo, de explorar nuevos campos y sobre todo por toda la ayuda, la paciencia y el apoyo recibido. Gracias por todo.

Como dice la canción, "We can do anything if we put our minds to it"

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

1.1 Motivación

Internet de las Cosas (Internet of Things, IoT) consiste en la interconexión entre diferentes dispositivos informáticos a través de la red de Internet. Estos aparatos pueden ser desde televisiones, teléfonos móviles y persianas hasta incluso riegos inteligentes, termostatos o vehículos. [21] Además de esta interconexión cada vez más notable y utilizada por los usuarios, también hay que mencionar el fenómeno de la digitalización que ha ido incrementando en la mayoría de los sectores como son la banca, finanzas, medios de comunicación, telecomunicaciones o sanidad, entre otros.

Debido al aumento del uso de IoT y de la digitalización, el número de dispositivos en IoT han ido creciendo notablemente y esto supone un aumento considerable en los datos que se recopilan, analizan y gestionan. Además, la computación al borde de la red (Edge Computing) permite realizar este procesamiento de datos cerca del sitio donde los datos son recolectados, en vez de tener que subirlos a una nube y procesarlos y analizarlos allí. Esto supone una gran ventaja en cuanto a eficacia y rapidez a la hora de tener que procesar los datos que se almacenan y poder usarlos según las necesidades de los usuarios y sistemas interconectados. [22]

Una vez se contemplan estos escenarios, aunque son muy útiles para proporcionar servicios útiles y eficaces a los usuarios, se debe tener mucha precaución con los datos que se recolectan, pues estos nodos de procesado y almacenamiento, si no se diseñan adecuadamente, pueden sufrir brechas de seguridad y los datos se pueden ver comprometidos. Por ello es necesario que los dispositivos y nodos de agregación estén protegidos ante ataques que impidan el correcto funcionamiento del sistema o que se hagan con los datos que estos recopilan. Puede tratarse de un ataque de Denegación de Servicio, que inunda un equipo de red con numerosas peticiones haciéndole agotar sus recursos, impedir la recopilación de medidas, inyectar medidas fraudulentas o incluso comprometer la información que ya se ha recopilado. Como los daños pueden afectar directamente a las personas y a las empresas, los nodos de agregación deben ser convenientemente asegurados.

En la Figura 1, se observa el gran incremento del uso del Internet de las Cosas organizado por sectores y su previsión hasta el año 2025. En ella se observan sectores como el móvil, el médico o el del consumidor, en los que se tratan datos confidenciales de empresas, personales o incluso gestionando aparatos que afecten directamente a vidas humanas.

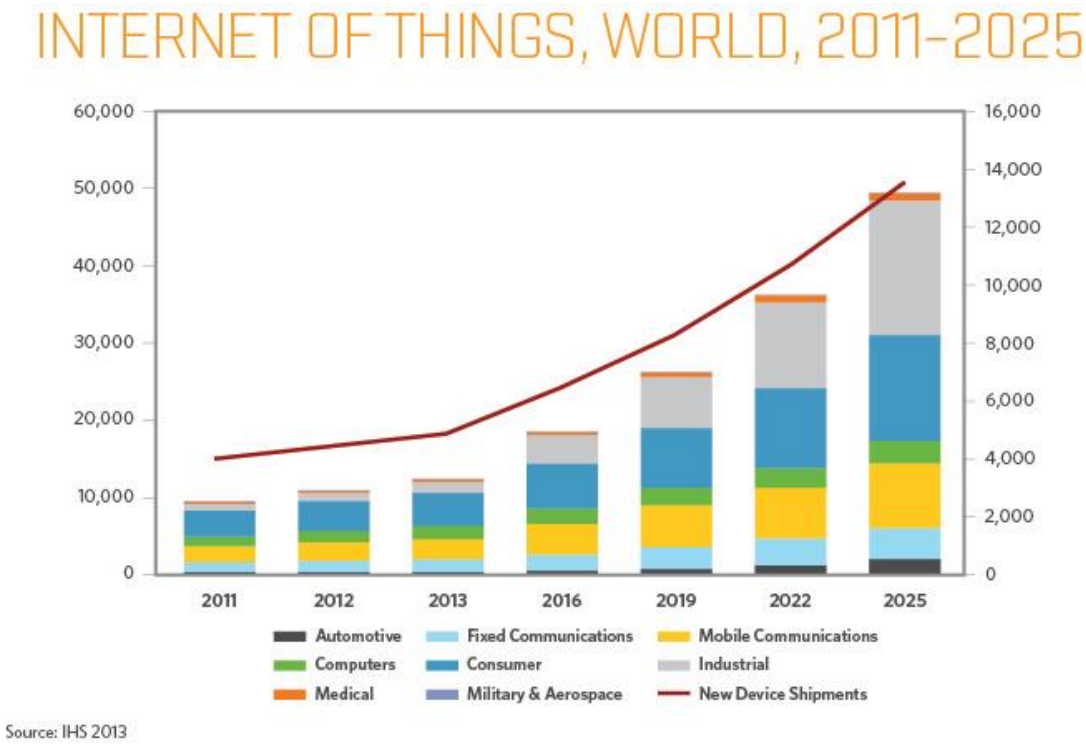

Figura 1 – Evolución Internet de las Cosas agrupado por sectores [1]

Es por todo lo mencionado, que la principal motivación de este proyecto es diseñar y desarrollar un módulo que se pueda incorporar a los nodos de agregación que se desplieguen en un entorno de computación al borde de la red de y que ofrezca interfaces de programación de aplicaciones que permitan extenderlo con nuevas funcionalidades. En particular, nos centraremos en desarrollar la funcionalidad de detección y bloqueo de ataques de DoS.

1.2 Objetivos

El objetivo de este Trabajo Fin de Grado es el diseño y desarrollo de un módulo que se pueda integrar en los nodos de agregación de Edge Computing y detecte ataques DoS. Además, si estos ataques son detectados, se podrá reconfigurar un cortafuegos, por ejemplo, basado en IPTables, para poder prevenir la continuación de dichos ataques.

Para llevar a cabo los objetivos mencionados, se han definido los siguientes subobjetivos:

- Investigar y obtener información sobre el uso y funcionamiento de las reglas IPTables usadas para la configuración del cortafuegos.

- Adquirir conocimientos sobre el desarrollo de herramientas de detección de ataques junto con el manejo de paquetes en lenguaje Python para así poder analizar las direcciones IP y puertos, tanto de destino como de origen.

- Diseñar y preparar un entorno de trabajo para la simulación del proyecto en una máquina virtual (Lubuntu) y en un miniordenador Raspberry Pi.

- Plantear un escenario experimental, en este caso, mediante la adición de una capa de seguridad para un bróker MQTT (Mosquitto) que permita aplicar el sistema desarrollado para una red IoT y así comprobar su utilidad para una posterior aplicación real.

- Reflexión sobre el trabajo futuro en el ámbito estudiado que sea capaz de garantizar la mayor seguridad posible.

- Redactar una memoria en la que se documente todos los trabajos que se han llevado a cabo, así como los conocimientos adquiridos durante la realización de este proyecto.

1.3 Fases de realización del Proyecto

Para poder llevar a cabo con éxito el proyecto planteado inicialmente, se ha realizado un diagrama de Gantt adjuntado a continuación en el que se han dividido las horas de desarrollo del trabajo completo, junto con una tabla explicando las tareas llevadas a cabo.

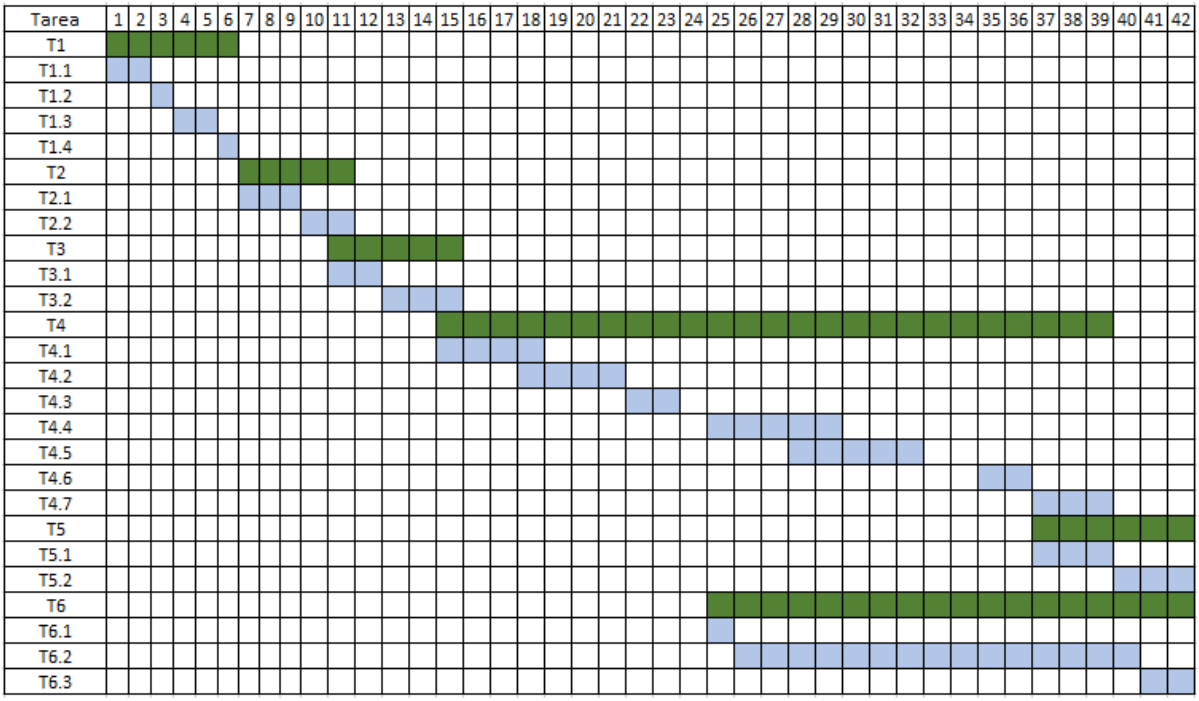

Figura 2 – Diagrama de Gantt

Tarea	**Actividad**	**Duración (semanas)**
T1	Monitorización	6
T1.1	Investigación sobre reglas IPTables	2
T1.2	Investigación sobre Python	1
T1.3	Investigación Algoritmo Heavy Hitters	2
T1.4	Despliegue Entorno Máquina Virtual	1
T2	Estudio Cortafuegos basado en reglas IPTables	5
T2.1	Estudio e Investigación Funciones	3
T2.2	Implementación primeras fases módulo Reglas	2

T3	Estudio Algoritmo Heavy Hitters	5
T3.1	Realización primeras pruebas Heavy Hitters	2
T3.2	Implementación Reglas + Heavy Hitters	3
T4	Implementación del Sistema	25
T4.1	Implementación Módulo Heavy Hitters	4
T4.2	Implementación Módulo Reglas IPTables	4
T4.3	Investigación Capturador de Tráfico TShark	2
T4.4	Implementación Módulo TShark	5
T4.5	Implementación Hilos	5
T4.6	Implementación Mútex	2
T4.7	Unión de todos los Módulos del Sistema	3
T5	Pruebas	6
T5.1	Estudio e Implementación Bróker Mosquitto (MQTT)	3
T5.2	Pruebas, Correcciones y Ampliaciones	3
T6	Presentación y Memoria	18
T6.1	Realización Tabla de Contenidos	1
T6.2	Realización de la Memoria	15
T6.3	Preparación para la Presentación	2

Tabla 1 – Tabla de Contenidos

1.4 Organización de la memoria

La memoria consta de los siguientes capítulos:

- En el capítulo 2, titulado *Estado del Arte*, se describen las tecnologías más relevantes que se van a emplear para el desarrollo del proyecto y sus respectivos estados actuales.

- El capítulo 3, titulado *Diseño y Desarrollo*, trata sobre los requisitos funcionales y no funcionales que deberá cumplir el sistema y se dividirá en requisitos. Además, se detallan los pasos que se han realizado para llevar a cabo el Trabajo y cómo se van a realizar para poder cumplir con los requisitos previos.

- El capítulo 4, titulado *Integración, pruebas y resultados*, recoge todos los resultados después de realizar pruebas tras ir completando el diseño del sistema.

- Finalmente, en el capítulo 5, *Conclusiones y trabajo futuro*, se presentan las conclusiones derivadas del trabajo realizado, así como posibles ideas para poder continuar desarrollando esta línea de investigación.

2 Estado del arte

En este apartado del proyecto se presentan las tecnologías empleadas en el mismo, en particular, Internet, Internet de las Cosas, dispositivos de seguridad de red (cortafuegos) haciendo uso de las reglas IPTables y gestores de mensajería para IoT.

De esta manera, se dará una visión extendida sobre el contexto en el que se va a trabajar y las tecnologías disponibles para ello.

2.1 Internet

Internet es una red de computadoras que interconecta millones de dispositivos informáticos a través de todo el mundo usando, principalmente, la pila de protocolos estándar TCP/IP. Estos dispositivos se denominan equipo de usuarios (hosts) o sistemas terminales, que se conectan con servidores para subir, bajar información o intercambiar datos y, son, desde ordenadores, teléfonos móviles, hasta videoconsolas, cámaras webs, vehículos autónomos o sistemas de riego. Estos dispositivos se conectan entre ellos mediante dispositivos de red que reciben paquetes de datos en sus puertos de entrada y, si es necesario, los procesan y los retransmiten a otros dispositivos vecinos a través de sus puertos de salida. Los más utilizados son los conmutadores de paquetes (switches en inglés) en la capa de enlace y los enrutadores (routers en inglés) en la capa de red. La principal función de un conmutador es recibir las tramas de la capa de enlace entrantes y reenviarlas hacia la salida. Además, los conmutadores realizan la función del filtrado y reenvío de los paquetes para decidir qué hacer con ellos al recibirlos. Los routers son dispositivos de conmutación que almacenan y reenvían paquetes usando direcciones de la capa de red que al contrario que los conmutadores de paquetes, no los reenvía usando la dirección MAC. [2]

Internet tiene su origen en la década de los sesenta en la DARPA (Defense Advanced Research Projects Agency), donde surgió la necesidad de conectar los computadores usados en sus investigaciones. Sin embargo, no fue hasta los setenta, cuando gracias a ARPANet (Advanced Research Projects Agency Network) se despliega una red de comunicaciones de alta velocidad que permite consultar resultados de investigaciones de otros laboratorios sin tener que viajar para informarse en persona, además de poder publicar los resultados de dichas investigaciones.

Desde entonces, Internet ha estado en continuo crecimiento y gracias a todos los avances que ha sufrido, su eficacia ha sido mejorada, y en 2019 se han alcanzado los 4.388 millones de usuarios. [3]

2.2 Internet de las Cosas

Internet de las Cosas (IoT) es la combinación de dispositivos electrónicos, software y la conectividad de red, que permite que estos dispositivos reúnan e intercambien datos entre ellos.

La expresión “Internet de las Cosas” fue introducida por Kevin Ashton, pionero de la tecnología británica en el año 1999 y fundador del Auto-ID Center en el Instituto de Tecnología de Massachusetts. La idea fundamental consistía en una tecnología que permitiera el almacenamiento y recuperación de datos remotos denominada RFID

(Identificación por Radiofrecuencia) que transmite el identificador de un objeto mediante ondas de radio. De esta manera los aparatos pueden interactuar entre ellos mediante este sistema, sensores móviles, etc. [9]

Con los años y los numerosos avances que se han ido implementando, la idea original ha ido también evolucionando para permitir el uso de diversas tecnologías electrónicas e inalámbricas para la comunicación y el procesado de datos. Esto refleja que se trata de un mercado donde entran no solo empresas tecnológicas, sino que también tienen cabida sectores como la automatización de los hogares, generación de electricidad, automoción, telecomunicación y/o tecnologías de la información. [4]

Además de intercambiar información entre dispositivos, también se analiza la cantidad de datos que se ven implicados en esta transacción. Para poder convertirlos en información útil y garantizar la interoperabilidad, es indispensable proporcionar datos suficientes en formatos y protocolos bien definidos. Al estar tratando con una gran cantidad de datos, que como se ha mencionado antes, pueden ser personales o confidenciales, la seguridad debe ser una propiedad clave que se debe tener en cuenta a la vez que el diseño del sistema.

Internet de las Cosas se puede considerar en distintas visiones: orientado a objetos, orientado a Internet y orientado a semántica. Orientado a objetos ya que se centra en el propio objeto siempre que este tenga un identificador único que lo diferencie del resto; orientado a Internet ya que se enfoca como un conjunto de nodos dentro de un sistema distribuido que reciben la información del mundo físico y la deben transcribir al mundo digital para poder enviarla; orientada a semántica ya que es necesario buscar, almacenar, conectar y representar toda la información que se recolecta.

Para poder unir estas visiones es necesario incorporar los objetos físicos con las formas de red (red de área local o LAN, red de área metropolitana o MAN y red de área extensa o WAN) para conseguir comunicación entre ellos, por lo que se necesita no sólo una red que asigne direcciones únicas a cada uno de los objetos, sino que, además, se tienen que proporcionar con mecanismos de conectividad. [5]

La plataforma de IoT se puede subdividir en tres componentes:

- **Equipamiento**: todo componente físico, normalmente electrónico, que se emplea para el despliegue de una infraestructura IoT.

- **Aplicativos**: cualquier componente que defina la lógica de funcionamiento del sistema.

- **Conectividad**: cualquier componente que permita la interconexión con cada unidad de equipamiento.

Al ser millones de dispositivos los que se conectan a Internet y a plataformas en la nube, es necesario el uso de una red segura y fiable para que los datos no se vean comprometidos. Para ello, es necesario que se garantice seguridad en todas las partes de una plataforma de IoT, pero en especial la de conectividad ya que es la que se muestra más vulnerable a ataques a través de la red de Internet que es la que más expuesta se encuentra. Por ejemplo, se usan millones de sensores y dispositivos informáticos que detectan, recopilan y analizan gran parte de nuestra información personal como la ubicación, la lista

de contactos, patrones de navegación e incluso el estado físico o de salud. Esta recopilación es muy útil para la comodidad del usuario ya que contando con ella los dispositivos pueden reaccionar mejor a sus necesidades, deseos o estados de ánimo, sin embargo, esta comodidad se consigue a expensas de un desafío que puede llegar a poner en peligro todos esos datos que es la privacidad y la seguridad. [6]

El tipo de información que recopilan la mayoría de los dispositivos que usan IoT es privada y personalizada, y si es accesible a un agente malicioso puede causar graves daños a la reputación, la riqueza y seguridad personal del usuario. Estos dispositivos incluyen modos de firmware y depuración introducidos por los propios fabricantes, cuyo acceso no autorizado puede ocasionar la pérdida de millones de euros en propiedades intelectuales robadas o el mal uso de estos activos. Por tanto, es evidente que las vulnerabilidades de seguridad pueden llegar a ser catastróficas.

Existen diferentes tipos de ataques según el nivel en el que se esté: [7]

- **Ataques al nivel de equipamiento:**
    - Jamming: se crean interferencias mediante señales de radio impidiendo el envío/recepción de datos y la retransmisión repetida posterior, si existe.
    - Tampering: se introduce un programa maligno en el dispositivo para obtener información de la red redireccionándola fuera para otorgar información privilegiada al atacante como contenido protegido y/o claves de cifrado.
- **Ataques al nivel de aplicación:**
    - Replicación: se duplican entidades, como identificadores de dispositivos, para que el sistema permita intercambiar datos con los dispositivos fraudulentamente creyendo que son lícitos.
- **Ataques al nivel de conectividad:**
    - Denegación de Servicio: se satura la red enviando un alto número de peticiones generando de esta manera una interrupción del servicio. Cuando se origina desde múltiples fuentes que se sincronizan para realizar el ataque, se denomina ataque distribuido de denegación de servicio.
    - Sniffing: se extrae información de los paquetes mediante la captura de tráfico de la red desprotegido.

2.3 Ataque de Denegación de Servicio

Un ataque de denegación de servicio (ataque DoS) se lleva a cabo inundando a un equipo de red con peticiones de forma que genera un alto tráfico de datos agotando los recursos disponibles de red o del propio equipo, llegando incluso a filtrar información no cifrada. Además, se pueden combinar varias computadoras (Ataque Distribuido de Denegación de Servicio, DDoS) o dispositivos secuestrados que se conectan a Internet denominados bots, para poder realizar ataques a mayor escala y aumentar exponencialmente el poder de los mismos. [2]

La disponibilidad de recursos es fundamental para los usuarios para poder garantizar la calidad del servicio (QoS), y este es precisamente el objetivo del DoS, hacer que los usuarios no sean capaces de acceder a los recursos beneficiándose de vulnerabilidades de seguridad de los sistemas.

Según un estudio realizado por Kaspersky, un 33% de las organizaciones sufrieron en 2017 un ataque DDoS que supone un 16% más que en 2016. Además de aumentar el porcentaje de los ataques también crece su tamaño debido al uso de botnets. Una botnet es un gran conjunto de ordenadores infectados que son controlados por un agente malicioso para realizar ataques de denegación de servicios [11]. Estos ataques se usan también para ocultar otros tipos de ataques, en 2018 se publicó en el informe anual del CCN-CERT (Centro Criptológico Nacional) que este tipo de ataques se había usado para ocultar otros como infección por código dañino (50%), fuga o sustracción de datos (49%), intrusión de red o piratería (42%) o sustracción económica (26%). En 2017 se informó sobre la botnet Persirai, era capaz de atacar hasta 100 modelos diferentes de cámaras IP (se estima que hay 120000 cámaras en riesgo de formar una botnet). Según CCN-CERT el gran aumento de este tipo de ataques se debe principalmente al descenso de los costes de las herramientas que se requieren para realizarlos, en otras palabras, el ataque de Denegación de Servicio sale barato, por ello se ha elegido este tipo de ataque en este proyecto. [8]

2.4 Dispositivos contrafuegos

Un cortafuegos, firewall en inglés, es una combinación de software y hardware que separa la red interna de un hogar, empresa o institución, de la red pública, es decir, Internet, dejando pasar unos paquetes y bloqueando otros. El criterio que sigue el cortafuegos para decidir qué paquetes deja pasar y cuáles no lo decide un administrador de red por medio de reglas. [2]

Un cortafuegos tiene tres objetivos principales:

- **Todo el tráfico pasa por el cortafuegos:** Tanto las peticiones recibidas como las de respuesta deben pasar por el firewall y que este decida si las deja pasar o las bloquea.

- **Paso autorizado según la política de seguridad local:** El firewall puede restringir el acceso al tráfico autorizado.

- **Inmunidad a la invasión:** El firewall debe ser inmune a los ataques y no dar una falsa sensación de seguridad, que es mucho peor que no contar con un cortafuegos.

2.4.1 Funciones del Cortafuegos

El cortafuegos puede ser implementado como un equipo de red especializado o como una aplicación que se ejecute en un computador o servidor. Un ejemplo de este último son los paquetes Netfilter/IPTables que permiten interceptar paquetes de la pila de red del núcleo del sistema operativo Linux y definir las políticas que se deben aplicar sobre los paquetes que sean interceptados.

Según las funciones que realiza un cortafuegos, se pueden dividir en: [2]

- **Filtrado de paquetes**: realizan un filtrado de paquetes basado en reglas comprobando solo campos del paquete como direcciones IP y puertos. Si se detecta un paquete que cumple una regla, se acepta o rechaza el paquete.

- **Filtrado de flujos**: mantienen información de estado de los paquetes permitiendo componer flujos y saben si los paquetes son el inicio de una nueva conexión, son de una conexión ya existente, si terminan una conexión o si son erróneos. Suponen una mayor carga para el rendimiento de la red, pero ofrecen una mayor seguridad y son capaces de prevenir algunos ataques de denegación de servicio.

- **Filtrado de servicios**: trabajan sobre la capa de aplicación del Modelo OSI, Open System Interconnection y son capaces de distinguir los protocolos y contenido de los paquetes y determinar si deben ser bloqueados.

- **Cortafuegos de inspección multicapa:** permiten conexiones directas entre hosts locales y remotos mediante algoritmos para dar máximo control al usuario sobre qué paquetes rechazar. Estos sin embargo tienen algunas desventajas. Al estar paralizando los paquetes para analizarlos el sistema se ralentiza, y para cada aplicación que se usa se debe aplicar una pasarela de aplicación lo que dificulta que varios usuarios la usen a la vez o que usen la misma pasarela, y, por último, el software del receptor debe saber cómo contactar con la pasarela al recibir una solicitud.

2.4.2 Reglas IPTables

En este proyecto se va a usar un conjunto de reglas denominadas IPTables que están vinculadas al kernel de Linux. En este apartado se van a explicar las secciones más importantes de IPTables y que van a ser relevantes para llevar a cabo este proyecto. Si se quiere profundizar más sobre ellas, la información completa se encuentra en la página oficial [13].

IPTables es un cortafuegos configurable y flexible que se encarga de recibir paquetes que serán aceptados o no al consultar las reglas existentes en las tablas. Las tablas están ya formadas con un conjunto de reglas iniciales que contienen unas condiciones (target) que si se cumplen hacen que la regla se ejecute. [12] Dentro de IPTables se encuentran cinco tablas, cuyas funciones se explican a continuación: [13]

- **RAW:** Filtra los paquetes con prioridad a otras tablas. Configura exenciones de seguimiento con objetivo NOTRACK. Proporciona las cadenas PREROUTING y OUTPUT.

- **FILTER:** Tabla por defecto que permite el filtrado de las comunicaciones.

- **NAT:** Se usa para traducir direcciones de red. El filtrado no se hace en esta tabla.

- **MANGLE:** Se usa para la alteración especializada de paquetes. Se admiten las cadenas PREROUTING, OUTPUT, INPUT, FORWARD y POSTROUTING.

- **SECURITY:** Esta tabla es para las reglas de MAC, como las habilitadas por SECMACK. Proporciona las cadenas OUTPUT, INPUT y FORWARD.

Además, cuenta con cinco cadenas que son listas de reglas que van incluidas por defecto en las tablas; estas son: INPUT, OUTPUT, PREROUTING, FORWARD y POSTROUTING.

Una vez se tienen las cadenas y las listas definidas, es necesario establecer una condición (Target), para saber qué hacer con el paquete recibido. Existen tres opciones:

- **ACCEPT**: se deja pasar el paquete.
- **DROP**: el paquete se rechaza.
- **RETURN**: pasar a la siguiente cadena bien, si esta ha terminado o si una regla en una cadena tiene como objetivo directamente RETURN.

Para gestionar las reglas IPTables se suelen usar unos comandos por terminal según las funciones que se quieran realizar. Para ello, se usa el comando "iptables" seguido de la acción que se quiera emplear, todas estas acciones se muestran en la siguiente tabla:

-t, table	Indica la tabla de coincidencias de paquetes en la que se debe operar. Dichas tablas son: filter, nat, mangle, raw y security. (Explicadas en el estado del arte en el apartado Firewall).
-A, append	Agrega una o más reglas al final de la cadena seleccionada.
-C, check	Comprueba si existe una regla que coincida con la especificación en la cadena seleccionada.
-D, delete	Elimina una o más reglas de la cadena seleccionada. Esta regla se puede especificar como un número de la cadena que se corresponda con su posición o bien la regla directamente.
-I, insert	Inserta una o más reglas a la cadena seleccionada.
-R, replace	Reemplaza una regla en la cadena seleccionada.
-L, list	Lista todas las reglas existentes en la cadena seleccionada. Si no se selecciona una cadena, se listan todas ellas. Por ejemplo: -t NAT -n -L (La n es para evitar largas búsquedas de DNS inversas).
-S, list rules	Imprime todas las reglas de la cadena seleccionada.
-F, flush	Limpia la cadena seleccionada, es decir, borra todas las reglas que estén contenidas en ella.

-Z, zero	Pone a cero los contadores de los paquetes y los bytes de la cadena o regla seleccionada.
-N, new chain	Crea una cadena nueva.
-X, delete chain	Elimina la cadena definida por el usuario. Antes es necesario borrar las reglas que contiene la cadena.
-P, policy	Establece una política (ACCEPT o DROP) para la cadena seleccionada que debe ser la incorporada por el usuario.
-E, rename chain	Renombra la cadena creada por el usuario.

Tabla 2 – Acciones reglas IPTables por terminal

2.5 Docker

Docker es un proyecto de código abierto que permite el uso de aplicaciones dentro de contenedores de forma aislada y segura con sus respectivas bibliotecas. Un contenedor es una unidad estándar de software que empaqueta el código y todas sus dependencias correspondientes para que la aplicación se ejecute de forma rápida y aislada de un entorno informático a otro. Una imagen de contenedor Docker es un paquete software de poco peso y ejecutable que incluye todo lo necesario para ejecutar la aplicación que deseemos: código, herramientas del sistema, bibliotecas del sistema y configuraciones. Se aísla el software de su entorno y así se garantiza un funcionamiento de manera uniforme. De esta manera al poder usar estos contenedores de manera independiente evitan la sobrecarga del funcionamiento de una máquina virtual, como se observa en la siguiente figura:

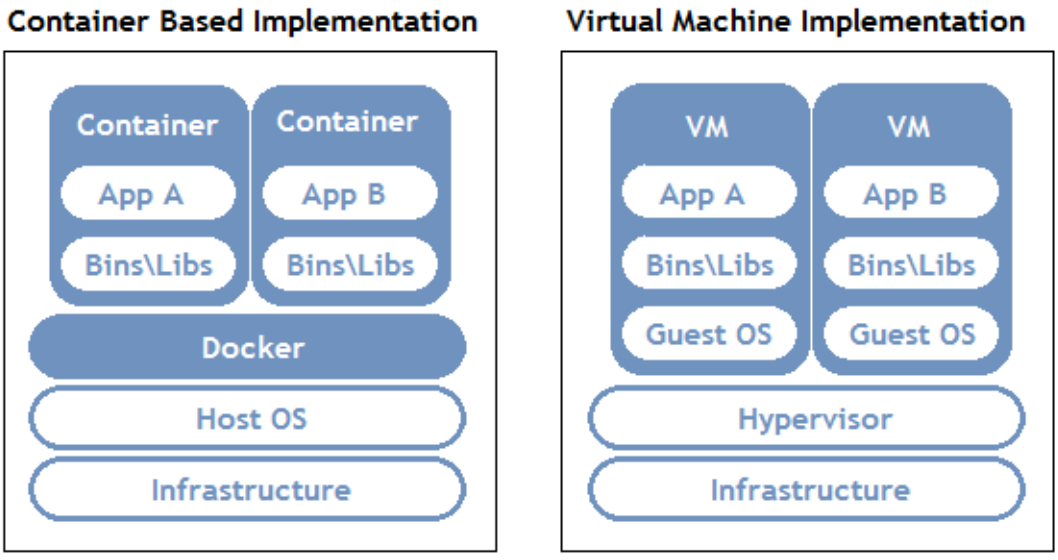

Figura 3 – Docker vs Máquina Virtual [20]

En este proyecto se va a usar un contenedor Docker de Ubuntu [18] para poder ejecutar en él los comandos del intermediario de mensajes Mosquitto (explicado en la sección 2.6) y realizar las simulaciones correspondientes emulando multitud de dispositivos que conectan y mandan mensajes al intermediario.

2.6 Bróker Mosquitto (MQTT)

Eclipse Mosquitto es un intermediario de mensajes de código abierto que proporciona un método ligero para llevar a cabo la mensajería utilizando un modelo de publicación/suscripción. Este modelo lo hace adecuado para la mensajería de Internet de las Cosas.

El protocolo MQTT se basa en el principio de publicar mensajes y suscribirse a temas, de forma que varios clientes se conectan a un intermediario y se suscriben a los temas que les interesen o publican mensajes en dichos temas. El intermediario y MQTT actúan como una interfaz simple y común para que se realice la interconexión. Al igual que para el caso de las reglas IPTables, en el caso de Mosquitto se va a explicar en este apartado la información más relevante para poder realizar este proyecto, en este caso el modelo publicador-suscriptor para poder realizar las simulaciones para enviar peticiones.

Primero se necesita un publicador (Publisher) que va a ser el encargado de enviar el mensaje del cliente mediante un campo denominado “topic”. Para poder recibir y leer el mensaje que el publicador ha enviado se necesita un suscriptor (Suscriber) que usando el mismo “topic” que el publicador será capaz de recibir el mensaje.

3 Diseño y Desarrollo

En este apartado, una vez expuesta la idea del proyecto y el estado actual de la tecnología que se va a usar, pasamos a analizar todas las actividades que debe realizar el sistema propuesto, exponiendo los requisitos que debe cumplir. Estos servirán como guía para el diseño, desarrollo y pruebas del correcto funcionamiento del sistema. Además, una vez tratados los requisitos del diseño se pasará a tratar el diseño y desarrollo de este Trabajo Fin de Grado.

3.1 Requisitos

Para poder detectar un ataque de denegación de servicio en nuestro sistema, es necesario monitorizar el tráfico entrante y analizar las direcciones IP de origen y destino de las peticiones. De esta forma, podremos contabilizar el número de veces que una dirección IP o puerto envía peticiones para saber si se trata de tráfico normal o de un ataque. Entonces, deberemos cumplir con los siguientes requisitos: captura del tráfico en tiempo real, tratamiento automatizado de datos, funcionamiento en tiempo real y capacidad de configuración de parámetros.

- **Captura del tráfico en tiempo real**: El sistema empleará TShark, que es un analizador de protocolo de red que permite capturar paquetes de una red y así poder monitorizar el tráfico entrante. Para ello se va a realizar un filtrado según los criterios del programador para localizar ataques, en este caso los campos a filtrar van a ser las direcciones IP origen y destino, y los puertos TCP origen y destino.

  Para poder analizar el tráfico de una forma eficaz y segura es necesario que el sistema no sufra una pérdida de paquetes y que la velocidad de procesamiento de estos sea lo suficientemente rápida como para poder tratar todas las peticiones sin problemas en un escenario de IoT ya que se puede tratar con un alto número de dispositivos conectados al nodo de agregación. Esto es necesario para que el sistema no esté en ningún momento sin protección, y, por lo tanto, vulnerable ante un ataque DoS ni sobrecargar el sistema de procesado. La librería que se usa para que la captura soporte el filtrado de los paquetes es la librería pcap, y puesto que el lenguaje de programación elegido en este proyecto en Python, se necesita usar la versión de TShark adaptada a este lenguaje: PyShark. [14]

- **Tratamiento automatizado de datos**: Una vez se ha conseguido capturar el tráfico de las peticiones que van llegando, se debe contar con un mecanismo eficiente que sea capaz de analizarlas y poder determinar si se está ante una situación de riesgo o no. Para ello se va a desarrollar un módulo que conste de un algoritmo Heavy Hitters [16] que será el encargado de analizar las direcciones IP origen que se han ido extrayendo de los paquetes con PyShark. El algoritmo Heavy Hitters contabilizará de forma relativa el número de paquetes enviados por una IP al servidor y así poder detectar ataques de Denegación de Servicio (DoS).

- **Funcionamiento en tiempo real**: Para que el sistema planteado en este Trabajo Fin de Grado funcione correctamente y en ningún momento haya ninguna vulnerabilidad, es necesario mantener el análisis y monitorizar los datos en tiempo real y funcionando en paralelo. De esta manera se podrá detectar y solucionar los

ataques que se reciban en el momento en el que sucedan y así se conseguirá que se produzca el mínimo de consecuencias posibles.

- **Utilidad de configuración de parámetros**: Para poder ajustar los parámetros que se usan en este sistema según las necesidades del usuario, se permite que este pueda modificarlos para adecuarlos a cada situación.

  Los parámetros que se pueden configurar son los siguientes: parámetros del algoritmo Heavy Hitters (numbuckets, threshold, blocktime), parámetros de filtros para el TShark (element(cola), interface, capFilter) y los parámetros del Docker para el MQTT (num_publishers, num_messages y delay).

- **Documentación:** El código que se programe debe estar bien documentado para que se pueda usar en futuras mejoras que se propondrán en el último punto de esta memoria.

3.2 Desarrollo

Una vez definidos los requisitos que debe cumplir el sistema, se va a tratar detalladamente el proceso de creación del sistema completo de detección de ataques de Denegación de Servicios en nodos de agregación para lograr el cumplimiento de todos los requisitos que se han tratado previamente. Para explicar el desarrollo y la implementación del proyecto, se va a dividir el desarrollo en cinco apartados: **i)** Sistema completo, **ii)** Modulo Capturador de Tráfico y Filtrado: PyShark, **iii)** Estudio del Filtrado: Algoritmo Heavy Hitters, **iv)** Mecanismo de Defensa: Cortafuegos con IPTables, **v)** Docker de MQTT y **vi)** Conclusiones.

3.2.1 Sistema Completo

El sistema que se ha desarrollado en este Trabajo Fin de Grado se compone de un conjunto de guiones de comandos (scripts en inglés) escritos en el lenguaje de programación Python en la versión 3.6, cuyo manual de instalación se encuentra en el anexo A.A.2. La elección de esta versión se debe a su compatibilidad con el módulo PyShark y sus correspondientes librerías, ya que con versiones inferiores no es compatible. PyShark es una librería que permite a Python interactuar con TShark. [14]

Además, el entorno elegido ha sido Linux que es donde residen las reglas de cortafuegos basados en IPTables (explicado en el 2.4.2) que se van a usar y que han sido anteriormente definidas.

El programa se ejecutará lanzando en Python el fichero `main.py`, donde se encuentran los parámetros configurables por el usuario, y las comunicaciones se realizarán a través del bróker Mosquitto MQTT para filtrar las peticiones de su puerto, el 1883.

La siguiente figura representa un diagrama de arquitectura para dar una idea general de su funcionamiento:

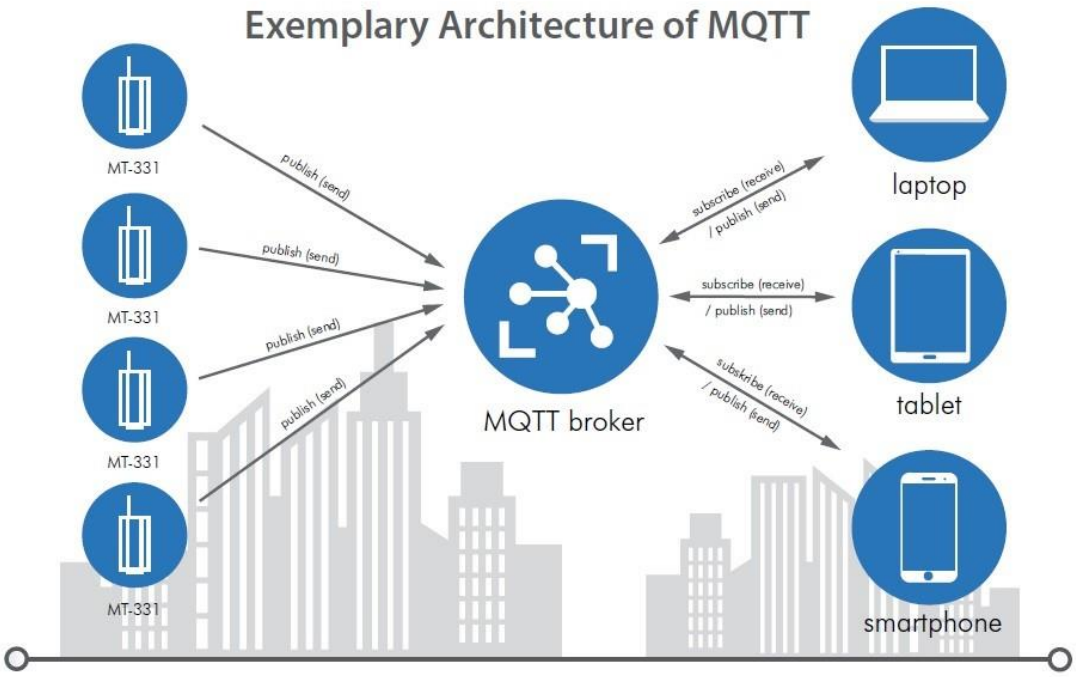

Figura 4 – Arquitectura MQTT [17]

Se han desarrollado dos tipos de conexiones:

- En la primera se han empleado tres terminales desde las que se arranca el MQTT dos en modo publicador para publicar mensajes y una en modo suscriptor para escuchar mensajes publicados, definidos en el apartado 2.6. Esta primera conexión se ha realizado para comprobar si el sistema funciona con un número pequeño de direcciones IP antes de pasar a las pruebas más pesadas en cuanto a peticiones. Se ha usado una máquina virtual cuyo manual de instalación de se encuentra detallado en el A.A.1, y una Shell de Linux en Windows para poder tener dos direcciones IP distintas. A continuación, se muestra una figura para visualizar la conexión.

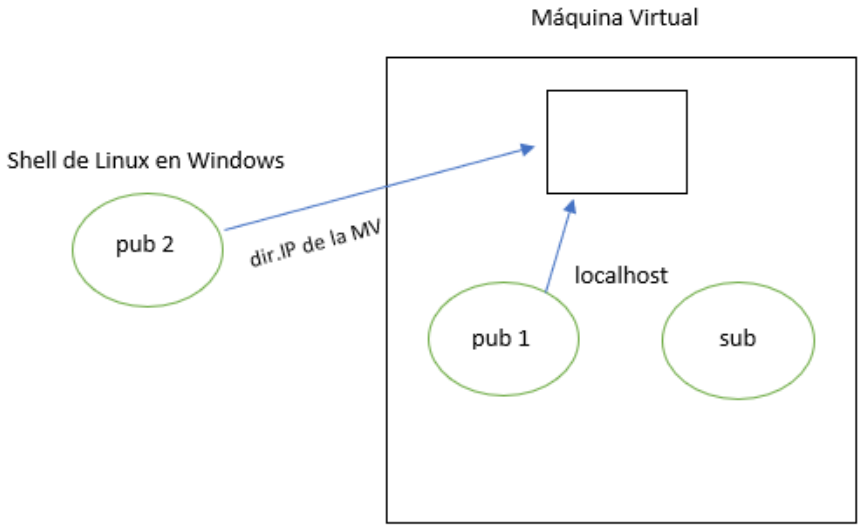

Figura 5 – MQTT con 2 Publicadores (pub)

A continuación, se muestran los comandos de las tres terminales utilizadas para llevar a cabo las comunicaciones:

$ mosquitto_pub -h localhost -t “main.py” -m “I am listening”

$ mosquitto_sub -h localhost -t “main.py” -v

mosquitto_pub -h 192.168.1.49 -t “main.py” -m “Windows”

Tabla 3 – Comandos MQTT

Los dos primeros comandos son los que se ejecutan desde la Máquina Virtual y se corresponden, el primero con el publicador que se encarga de enviar el mensaje “I am listening” con el topic “main.py”. El segundo es el suscriptor que es el que realiza la escucha de los dos publicadores, con el mismo topic “main.py”. Finalmente, el tercer comando es el publicador de la shell de Linux de Mindows cuya direccion IP en vez de ser localhost como los anteriores, se tiene qe corresponder con la direccion IP de la Máquina Virtual y el topic el mismo que el de los dos comandos anteriores, “main.py”. Con estos requisitos ya se tiene el sistema de envío y recibimiento de peticiones preparado.

- La segunda se ha llevado a cabo usando Docker, con un contenedor de Mosquitto para poder probar el sistema con más de dos direcciones IP distintas, permitiéndonos realizar una prueba más completa del funcionamiento del sistema desarrollado. Además, se han definido unos parámetros variables escogidos por el usuario para poder analizar el rendimiento del sistema. Estos parámetros son la cantidad de publicadores con direcciones IP distintas mediante la variación del **número de publicadores** que se arrancan, el **número de mensajes** que se envían por publicador, y el **retraso** que existe entre cada uno de los mensajes publicados por un publicador. Con estos tres parámetros se recogerán en el capítulo 4 los resultados de las distintas pruebas que se realizarán para determinar el rendimiento del sistema.

Para finalizar este primer subapartado, se muestra una figura del sistema con todos los módulos y scripts para tener una percepción más clara del proyecto. En ella se observa el módulo principal que es el *main* desde donde se configuran los parámetros elegidos por el usuario. Una vez elegidos la interfaz y el filtro, se llama al módulo TShark para empezar a capturar el tráfico entrante para rellenar una cola con todas las direcciones IP filtradas y que se puedan analizar en el siguiente paso, en el algoritmo Heavy Hitters. A este algoritmo se le pasan también desde el *main* las medidas del diccionario y el número máximo de repeticiones de IPs a partir de la cual se considera que se está ante un ataque de Denegación de servicio. Si dicho ataque existe, se ejecuta una función de *callback* dada para poder añadir la dirección IP causante al conjunto de reglas IPTables para defender el sistema. Pasado un tiempo determinado, se comprueba si el ataque persiste, en cuyo caso la regla se mantiene, si no se borra. En todo momento, el filtrado continúa junto con el análisis del mismo.

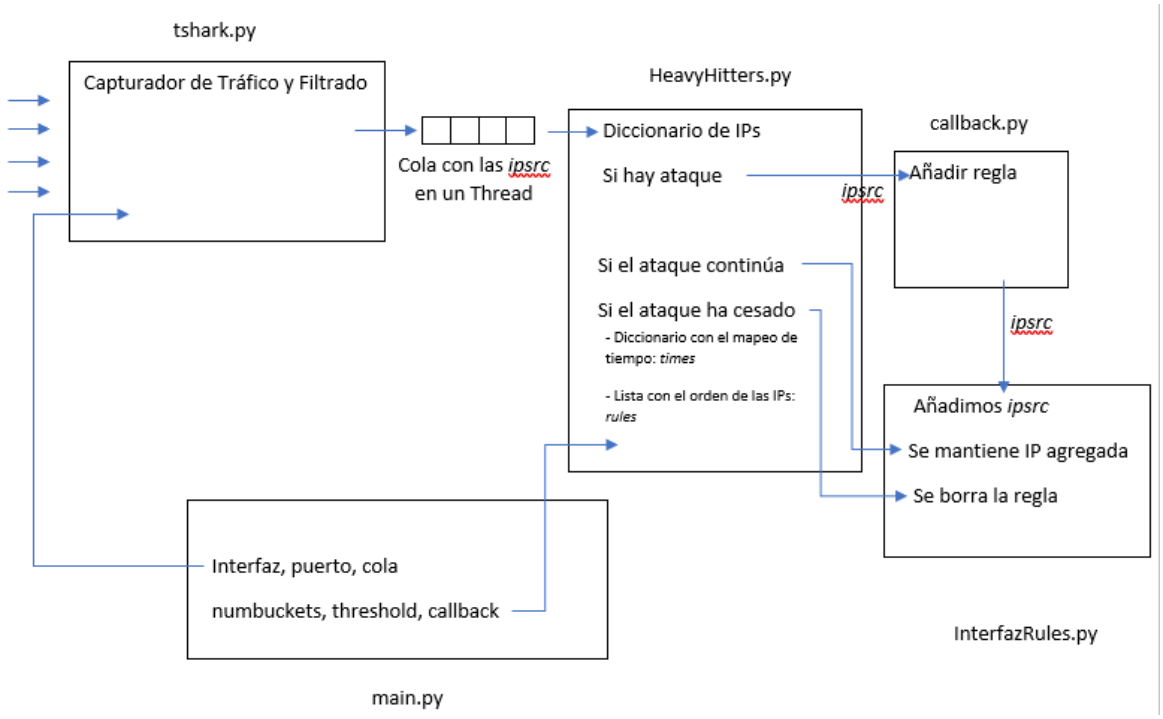

Figura 6 – Sistema completo

Una vez se tiene una idea general del sistema, se va a tratar cada módulo para así explicar sus respectivas funcionalidades.

3.2.2 Módulo Capturador de Tráfico: PyShark

El primer paso para que funcione PyShark es instalar su librería, cuyos pasos se detallan en el A.A.4. Una vez contamos con la librería instalada podemos importarla a nuestro script y empezar a trabajar con ella. Además, hay que instalar el TShark cuyos pasos se detallan también en el A.A.4.

Se definen las variables de **interfaz**, el filtro que especifica el **protocolo** y la **cola** en la que se van a ir guardando las direcciones IP de origen pendientes de procesar para luego poder analizar el tráfico capturado. Estos parámetros se inicializan desde el módulo *main*. A continuación, se muestra una tabla de las variables que se han mencionado:

```
def __init__(self, interface, capFilter, element):
    self.__interface = interface #'any'
    self.__capFilter = capFilter #'tcp'
    self.__element = element #queue
```

Tabla 4 – Variables Capturador

Una vez definidos los parámetros con los que se va a realizar la monitorización del tráfico, se procede a realizar la captura continúa de los paquetes de entrada que contienen el tráfico del cliente al servidor, uno de los primeros requisitos establecidos para no ser vulnerable a un ataque en ningún momento, mediante la función *sniff_continuously()*. Aparte de filtrar por los protocolos UDP o TCP, se filtra el puerto 1883 que es el puerto del bróker Mosquitto (MQTT) a través del cual se realiza la escucha y el envío de peticiones mediante los comandos especificados en el subapartado anterior.

PyShark es llamado desde el módulo *main* a través de un hilo para que se esté continuamente filtrando sin paralizar el resto de funcionalidades del sistema. Una vez se realiza la captura según la interfaz y el protocolo elegido, se recopila la informacion que nos va a ser útil para el análisis del tráfico, en este caso, dirección IP origen, dirección IP destino, puerto origen y puerto destino. En un caso real habría que tener en cuenta otros campos como las banderas TCP y hacer análisis de flujos más detallados, que se tratará en el apartado de Trabajo Futuro.

De estos cuatro parámetros, se inserta el dato de la direccion IP origen a la cola que se envía al algoritmo Heavy Hitters (descrito en 3.2.3) para que este se encargue de analizarla y reaccionar como convenga.

3.2.3 Estudio del Filtrado: Algoritmo Heavy Hitters

En este módulo es donde se analiza la información que se ha capturado por Pyshark. Esta información son las direcciones IP origen de paquetes que han pasado los filtros indicados en la sección 3.2.2, que se introducen en un diccionario según van llegando por la cola que se envía desde el PyShark. A cada una se le adjudica como clave la dirección IP y un contador que irá llevando el recuento (relativo al resto de publicadores activos) de veces que un usuario intenta establecer conexión desde una misma dirección IP. Si este contador supera el valor límite **threshold** (línea 1 de la Tabla 5), un parámetro determinado por el usuario desde el módulo *main*, se llama al módulo del control del firewall de IPTables mediante una función de **callback** (línea 6 de la Tabla 5). De esta manera la dirección IP que ha sobrepasado el número de peticiones establecido como normal por el usuario pasa a formar parte de las reglas bloqueadas en *InterfazRules* que es donde se reconfigura el firewall y de esta forma se evita que el ataque se lleve a cabo, tal y como se ve reflejado en la siguiente tabla:

```
1   if contador >= self.__threshold:
2       #If callback is not none, call callback
3        if self.__callbacks in not None:
4           #Add rule calling Callback
5           print('Calling Callback from HH with', element)
6           self.__callback(element)
```

Tabla 5 – Heavy Hitters I

Además, el diccionario tiene un límite de direcciones IP introducidas que se controla con la variable **numbuckets** (línea 1 de la Tabla 6), tambien establecida por el usuario desde el módulo *main*. Si en algún momento alguno de los contadores supera el valor de esta variable, se restará uno a todos los contadores (línea 4 de la Tabla 6), de forma que al menos una dirección IP queda a cero para poder detectar elementos que destacan por su elevada frecuencia de ocurrencia, eliminando del diccionario la que quede a cero dejando de nuevo espacio para las que sigan llegando (línea 6 de la Tabla 6). Así no saturamos el diccionario y vamos liberando recursos para mantener la eficacia del sistema.

```
1    if len(self.__dictionary) >= self.__numbuckets:
2          remips = set()
3          for element_, contador_ in self.__dictionary.items():
4             self.__dictionary.update({element_: contador_-1})
5              if self.__dictionary.get(element_) == 0:
6                 remips.add(element_)
7          for remip in remips:
8             del self.__dictionary[remip]
9       self.__dictionary[element] = 1
```

Tabla 6 – Heavy Hitters II

Para que no se agreguen direcciones IP repetidas a las reglas IPTables, se creará una lista en la que se guarden las direcciones que se vayan incorporando al cortafuegos en el orden en el que estas se vayan incluyendo. Además, se empleará un diccionario que se encargará del mapeo del instante de tiempo en que se añade cada dirección IP. Con la lista y el diccionario se comprobará pasado un tiempo definido por **blocktime** pasado un tiempo calculado denominado **delay** siguen llegando peticiones de direcciones ya agregadas. Si no siguen llegando se eliminará la regla que bloquea dicha dirección IP permitiendo que vuelva a establecer contacto si lo desea.

Para ello se han definido dos funciones que son las encargadas de que no se dupliquen las direcciones en las IPTables y calcular los delay de cada una de las peticiones entrantes. En la primera función, *getElementToUnblock* (Tabla 7) se calcula el retardo *delay* (línea 6) de cada una de las direcciones dentro de la lista *rules*. Esta función se llama luego en la función *unblock* (Tabla 8) para saber si se debe eliminar o no de la lista de reglas IPTables. Si el valor calculado de delay es negativo (línea 6), se elimina la regla de la lista con el orden de IPs (línea 7), del diccionario del mapeo de tiempo (línea 8), del diccionario con las IPs (línea 9) y, finalmente de las reglas de IPTables (línea 10).

```
1   def getElementToUnblock(self):
2      with self.__lock:
3         if len(self.__rules) == 0: return None, None
4         element = self.__rules[0]
5         t=time.time()
6         delay=self.__times[element]+self.__blocktime-t
7         return element, delay
```

Tabla 7 – Heavy Hiiters III

```
1    def unblock(self, direcip):
2        iptables = IPTables()
3        with self.__lock:
4            element, delay = self.getElementToUnblock()
5            if direcip != element: return
6            if delay>0: return
7            self.__rules.remove(element)
8            del self.__times[element]
9            del self.__dictionary[element]
10           iptables.deleterule(direcip)
```

Tabla 8 – Heavy Hitters IV

Estas dos funciones se llaman desde un hilo en el módulo *main* que se encarga de realizar sus correspondientes comprobaciones sin paralizar el funcionamiento del sistema.

Una vez se ha detallado el funcionamiento de todas las funciones usadas en este módulo, se pasará a ver cómo, una vez analizados todos los parámetros necesarios de las peticiones que van llegando al sistema, se reconfigura el cortafuegos haciendo uso de las reglas IPTables.

3.2.4 Mecanismo de Defensa: Firewall con IPTables

Para poder hacer uso de las reglas IPTables es necesario instalarlas tal y como se detalla en el A.A.3. Una vez contamos con las instalaciones realizadas, se puede reconfigurar el firewall según las necesidades del usuario. Si en el módulo del *HeavyHitters* el **threshold** es excedido se enviará un **callback** al módulo de las IPtables denominado *InterfazRules* donde se procederá a la adicción de la nueva regla.

El primer paso es hacer un borrado de las reglas existentes incialmente al comenzar con nuestro filtrado, para no empezar con reglas ya agregadas de alguna ejecución previa. Para ello se ha desarrollado la función *flushrules()* que se explica a continuación:

```
1    def flushrules(self):
2        "" "Method Flush Chain" ""
3        print ('Flushing rules')
4        table = iptc.Table('filter')
5        table.autocommit = False
6        chain = iptc.Chain(table, "INPUT")
7        for rule in chain.rules:
8            chain.delete_rule(rule)
9        table.commit()
10       table.autocommit = True
```

Tabla 9 – Función borrado inicial (flush)

La función empieza seleccionando la tabla en la que se van a modificar las reglas, en este caso *filter* (línea 4). Después, para evitar que tras la primera operación de eliminación el estado de la cadena cambie y reinicie el recorrido, se deshabilitan las confirmaciones automáticas mediante el comando *autocommit=False*, permitiendo aplicar múltiples cambios en la cadena a la vez (línea 5). De esta forma realizamos un borrado previo y el sistema está listo para empezar a añadir o eliminar reglas según sus necesidades (línea 7 y 8). Se ha escogido esta opción por simplicidad, pero en un sistema donde múltiples procesos puedan reconfigurar reglas sería necesario selecciona qué reglas se pueden o no borrar.

La segunda función dentro del módulo *InterfazRules* es la que permite añadir reglas a nuestro Firewall para bloquear las peticiones provenientes de la dirección IP agregada, esta función se ha denominado *addrule()* (Tabla 10). Esta función selecciona la cadena INPUT (línea 3) para la adición de reglas, indícando además como parámetros la *rule.src* que se corresponde con la dirección IP que se va a agregar (línea 7). Después, crea una instancia de una regla, usando *Target* (línea 9), para poder después insertar la regla que se quiere añadir al firewall (12).

```
1   def addrule(self, ipcall):
2       print ('Adding rule', ipcall)
3       chain = iptc.Chain(iptc.Table(iptc.Table.FILTER), "INPUT")
4       # Create rule
5       rule = iptc.Rule()
6       # Any interface
7       rule.src = ipcall
8       # Target Rule
9       target = iptc.Target(rule, "DROP")
10      rule.target = target
11      # Insert rule
12      chain.insert_rule(rule)
13      print ('Added Rule')
```

Tabla 10 – Función para añadir reglas

La última función de este módulo es la encargada de eliminar una regla ya añadida previamente a las IPTables, denominada *deleterule()* en la Tabla 11.

La metodología es la misma que para añadir una regla, se selecciona la cadena INPUT (línea 3) , indícando además como parámetros la *rule.src* que se corresponde con la dirección IP que se va a eliminar (línea 7). Después, crea una instancia de una regla, usando *Target* (línea 9), para poder después eliminar la regla que se quiere añadir al firewall (12). A diferencia del borrado general de reglas realizado en *flushrules*, aquí al solo tener que eliminar una regla concreta no es necesario deshabilitar las confirmaciones automáticas.

```
1   def deleterule(self, ipcall):
2       print ('Deleting rule', ipcall)
3       chain = iptc.Chain(iptc.Table(iptc.Table.FILTER), "INPUT")
4       # Create rule
5       rule = iptc.Rule()
6       # Any interface
7       rule.src = ipcall
8       # Target Rule
9       target = iptc.Target(rule, "DROP")
10      rule.target = target
11      # Insert rule
12      chain.delete_rule(rule)
13      print ('Deleted Rule')
```

Tabla 11 – Función borrado regla añadida

Con estas funciones implementadas se cuenta con todas las herramientas necesarias para poder realizar la captura y filtrado del tráfico, analizar las tramas y reconfigurar el cortafuegos cuando sea necesario.

3.2.5 Docker de MQTT

Para poder lanzar el Mosquitto con Docker, se han realizado unos guiones de comando para facilitar su uso. El primero denominado *0-clean-up.sh* se encarga de eliminar los contenedores con su contenido antes de comenzar con las ejecuciones del sistema. Tal y como representa la Tabla 12, en la línea 4 se realiza el borrado del contenedor y en la línea 8 la red de Mosquitto utilizada.

```
1   #!/bin/bash
2
3   echo "----- Remove 'mosquitto' containers ----------"
4   docker ps -a | grep "mosquitto" | awk '{print $1}' | xargs -r docker rm -f
5   echo
6
7   echo "----- Remove 'mosquitto' network ----------"
8   docker network ls | grep "mosquitto" | awk '{print $1}' | xargs -r docker network rm
9   echo
```

Tabla 12 – Docker de Mosquitto I

A continuación, es necesario crear los nuevos contenedores para poder llevar a cabo las nuevas conexiones tal y como se muestra en la Tabla 13. Para ello lo primero es crear las nuevas redes de Mosquitto (línea 5) para después crear los nuevos contenedores (línea 9). Esto está contenido en el fichero *1-deploy-network-server.sh.*

```
1   #!/bin/bash
2
3   echo "----- Create 'mosquitto' network ----------"
4    # create network "mosquitto" with subnet 172.20.0.0/16 and gateway 172.20.0.1 (this
is your host's IP)
```

```
5  docker network create --driver bridge --subnet 172.20.0.0/16 --gateway 172.20.0.1 mosquitto
6  echo
7
8  echo "----- Create 'mosquitto-server' container ----------"
9  docker run -d --name mosquitto-server --network mosquitto --ip 172.20.0.2 eclipse-mosquitto
10 echo
```

Tabla 13 – Docker de Mosquitto II

El tercer paso es ejecutar el susciptor para poder ir viendo los mensajes de los publicadores que se van a ir ejecutando. Para ello, en el guión de comandos *2-deploy-sub.sh* se ha desarrollado el siguiente código recogido en la Tabla 14:

```
1  #!/bin/bash
2
3  echo "----- Create 'mosquitto-sub' container ----------"
4  docker run -it --rm --name mosquitto-sub --network mosquitto --ip 172.20.0.3 eclipse-mosquitto mosquitto_sub -h 172.20.0.2 -t '#' -v
```

Tabla 14 – Docker de Mosquitto III

En ella se observa como en la línea 4 se ejecuta el comando para poder arrancar el suscriptor e ir viendo qué publicador es el que realiza las peticiones en cada momento.

Finalmente, en el fichero denominado *3-deploy-pubs.sh* representado en la Tabla 15, es donde se realizan los siguientes pasos: el primero es inicializar y dar valor a las variables que se van a usar (línea 3 a línea 10) que son el número de publicadores normales, número de publicadores comprometidos, número de mensajes, delay en el primer caso de publicadores, delay del segundo caso de publicadores, el topic, y los mensajes de cada uno de los publicadores para poder distinguirlos en el suscriptor. A continuación, se definen dos bucles. El primero de ellos (línea 13 a línea 17) se encarga de ir enviando las peticiones correspondientes al primer grupo de publicadores con sus respectivas variables asociadas. El segundo (de la línea 20 a la 24) sigue el mismo formato que el anterior pero cambiando las variables ya que se corresponde a los publicadores comprometidos.

```
1  #!/bin/bash
2
3  NUM_PUBLISHERS=20 # max 250
4  NUM_PUBLISHERS_COMP=1 # max 250
5  NUM_MESSAGES=10
6  INTER_MESSAGE_DELAY=1 #seconds
7  INTER_MESSAGE_DELAY_COMP=1 #seconds
8  TOPIC='house/bulb'
9  MESSAGE='on'
10 MESSAGE_COMP='compromised'
11
12 echo "----- Create 'mosquitto-pub' containers  ----------"
13 for ((i=1;i<=$NUM_PUBLISHERS;i++));
```

```
14  do
15      HOSTID=$((i + 3))
16      docker run -d --rm --name mosquitto-pub-$i --network mosquitto --ip
172.20.0.$HOSTID eclipse-mosquitto mosquitto_pub -h 172.20.0.2 -t $TOPIC -m
$MESSAGE-$i --repeat $NUM_MESSAGES --repeat-delay
$INTER_MESSAGE_DELAY
17  done
18
19  echo "----- Create 'mosquitto-pub' containers comp ----------"
20  for ((i=1;i<=$NUM_PUBLISHERS_COMP;i++));
21  do
22      HOSTID=$((i + 3))
23      docker run -d --rm --name mosquitto-comp-$i --network mosquitto --ip
172.20.1.$HOSTID eclipse-mosquitto mosquitto_pub -h 172.20.0.2 -t $TOPIC -m
$MESSAGE_COMP-$i --repeat $NUM_MESSAGES --repeat-delay
$INTER_MESSAGE_DELAY_COMP
24  done
```

Tabla 15 – Docker de Mosquitto IV

Los tres primeros ficheros se ejecutan desde una terminal, el cuarto fichero desde una segunda terminal diferente para, en una tercera, ejecutar el programa completo desde el *main.py*.

3.2.6 Conclusiones

En este capítulo se han explicado todas las implementaciones que se han realizado en cada uno de los módulos de programa que se han utilizado, así como el funcionamiento de los mismos.

Para ello se ha dividido la explicación en los tres principales módulos que se han usado para el correcto funcionamiento del sistema para poder tener una idea clara de lo que se pretende con cada uno de ellos por separado, y a su vez el funcionamiento que se alcanza al usarlos todos en conjunto.

Además de explicar los módulos por separado, se ha explicado el funcionamiento del sistema completo y sus respectivas implementaciones para después realizar las diferentes pruebas, cuyos resultados se comentarán más adelante en el Capítulo 4.

4 Integración, pruebas y resultados

En este capítulo se probará el sistema completo que permitirá analizar el tráfico entrante y con ello las diferentes direcciones IP que vayan llegando para poder ir pasándolas al algoritmo Heavy Hitters e ir comprobando si existe ataque de Denegación de Servicio y tomar las medidas oportunas.

Una vez se ha probado el sistema explicado en 3.2.1 haciendo uso de la Máquina Virtual y la Shell de Linux en Windows, se van a realizar las pruebas más extensas haciendo uso del Docker de Mosquitto para poder tener más direcciones IP en el tráfico entrante. la terminal de la Máquina Virtual, y el otro desde la Shell de Linux en Windows, para poder ver con mayor detalle el funcionamiento del sistema.

4.1 Prueba del Sistema con Docker

Una vez se ha comprobado que el sistema funciona como se desea con un número bajo de direcciones IP enviando peticiones, necesitamos un escenario en el que el número de peticiones sea mayor y ver cómo reacciona el sistema.

Para ello, se cuenta con unos scripts de Docker para Mosquitto para poder realizar las diversas pruebas. En ellos, lo primero es determinar tres parámetros: el número de publicadores, el número de mensajes que enviará cada uno de ellos y el tiempo entre los mensajes. Una vez fijados estos valores, en la terminal y accediendo a la carpeta en la que están estos ficheros, es necesario seguir los siguientes pasos, los tres primeros desde una misma terminal, y el cuarto desde otra terminal:

1. Ejecutamos el guión de comandos 0-clean-up.sh explicado en 3.2.5 que se va a encargar de eliminar los contenedores existentes en el computador ejecutando el siguiente comando:

```
sh 0-clean-up.sh
```

Tabla 16 – Comando Docker I

El mensaje que se observa por terminal es el siguiente, correspondiente a los identificadores de los contenedores que se eliminan:

```
----- Remove ‘mosquitto’ containers --------
Bb84e21da325

----- Remove ‘mosquitto’ network -----------
09b95d7dff58
```

Tabla 17 – Resultado Comando Docker I

2. El siguiente paso es ejecutar el fichero 1-deploy-network-server.sh que crea un nuevo contenedor de Mosquitto mediante el siguiente comando:

```
sh 1-deploy-network-server.sh
```

Tabla 18 – Comando Docker II

En cuyo caso el mensaje que se obtiene por pantalla son los identificadores de cada contenedor que se crea:

```
----- Create ‘mosquitto’ network ----------------
6130419d41bd7a67164c92c1474b5cb52bf7543ee86021d999e2d41eba723d10

----- Create ‘mosquitto-server’ container ------
6bea3a1e066e9ce8d4baf99736caffde3baf2b34d3a15f7da0f255ef99ece703
```

Tabla 19 – Resultado Comando Docker II

3. El último paso de esta terminal es para ver en terminal los mensajes que se publican y contiene las instrucciones del suscriptor del Mosquitto, para lo que se ejecuta el siguiente comando:

```
sh 2-deploy-sub.sh
```

Tabla 20 – Comando Docker III

Una vez ejecutado, cuando se ejecuta en la segunda terminal el publicador del Mosquitto, se contemplan el siguiente tipo de mensajes correspondientes al parámetro de mensaje de cada uno de los publicadores que se definen en estos ficheros, como se ha explicado en 3.2.5:

```
house/bulb on-19
house/bulb on-18
house/bulb on-20
house/bulb compromised-1
```

Tabla 21 – Resultado Comando Docker III

4. Desde una segunda terminal, ejecutamos el último fichero del Docker que es el encargado de desplegar varios publicadores de mensajes:

```
bash 3-deploy-pubs.sh
```

Tabla 22 – Comando Docker IV

En este caso contemplamos por pantalla el identificador de cada contenedor con su publicador según el número de ellos que se haya definido.

En las pruebas que se van a realizar a continuación se van a ir variando los valores de todos los parámetros que se pueden configurar. Para ello, se va a emplear el siguiente sistema de tuplas facilitar su entendimiento y su seguimiento: (PN, PC, NB, OB, TB), donde PN=Publicadores Normales, PC=Publicadores Comprometidos, NB=Número de buckets, OB= Ocurrencias para el Bloqueo, TB=Tiempo de Bloqueo,

4.1.1 Prueba 1: Variación de la variable Blocktime

La primera prueba que se va a realizar va a ser variando la variable blocktime que es una variable que se usa para calcular el tiempo de bloqueo especificado (delay) de cada una de las direcciones IP agregadas a la lista de reglas bloqueadas para comprobar se siguen recibiendo peticiones de la misma IP, como se explica en el apartado 3.2.3.

La primera prueba se llevará a cabo con un valor muy alto de este parámetro y se irá disminuyendo para ver qué pasa.

1. **Blocktime=200 segundos**. En este caso se va a fijar un número de publicadores normales de 20, de comprometidos 10, número de buckets a 10 y el threshold a 5. Al tener un tiempo de blocktime tan elevado se detectarán todos y, en consecuencia, se bloquearán. El resultado de las reglas bloqueadas se observa en la Tabla 23 en la que aparecen todas ellas:

```
-P INPUT ACCEPT
-A INPUT -s 172.20.1.13/32 -j DROP
-A INPUT -s 172.20.1.12/32 -j DROP
-A INPUT -s 172.20.1.11/32 -j DROP
-A INPUT -s 172.20.1.10/32 -j DROP
-A INPUT -s 172.20.1.9/32 -j DROP
-A INPUT -s 172.20.1.8/32 -j DROP
-A INPUT -s 172.20.1.7/32 -j DROP
-A INPUT -s 172.20.1.6/32 -j DROP
-A INPUT -s 172.20.1.5/32 -j DROP
-A INPUT -s 172.20.1.4/32 -j DROP
-A INPUT -s 172.20.0.23/32 -j DROP
-A INPUT -s 172.20.0.22/32 -j DROP
-A INPUT -s 172.20.0.21/32 -j DROP
-A INPUT -s 172.20.0.20/32 -j DROP
-A INPUT -s 172.20.0.19/32 -j DROP
-A INPUT -s 172.20.0.18/32 -j DROP
-A INPUT -s 172.20.0.17/32 -j DROP
-A INPUT -s 172.20.0.16/32 -j DROP
-A INPUT -s 172.20.0.15/32 -j DROP
-A INPUT -s 172.20.0.14/32 -j DROP
-A INPUT -s 172.20.0.13/32 -j DROP
-A INPUT -s 172.20.0.12/32 -j DROP
-A INPUT -s 172.20.0.11/32 -j DROP
-A INPUT -s 172.20.0.10/32 -j DROP
-A INPUT -s 172.20.0.9/32 -j DROP
-A INPUT -s 172.20.0.8/32 -j DROP
-A INPUT -s 172.20.0.7/32 -j DROP
-A INPUT -s 172.20.0.6/32 -j DROP
-A INPUT -s 172.20.0.5/32 -j DROP
-A INPUT -s 172.20.0.4/32 -j DROP
```

Tabla 23 – (PN=20, PC=10, NB=10, OB=5, TB=200)

2. **Blocktime=100 segundos**: en esta segunda ejecución se ha disminuido hasta 100 los segundos de esta variable. El número de publicadores normales se ha fijado a 20 y el de publicadores comprometidos a 1, mientras que el nºbuckets se fija a 10 y el threshold a 5. El resultado de las reglas finales añadidas es el representado en la Tabla 24, en la que se observa que se han agregado al bloqueo las direcciones IP de todos los publicadores, comprometidos o no:

```
-P INPUT ACCEPT
-A INPUT -s 172.20.1.4/32 -j DROP
-A INPUT -s 172.20.0.23/32 -j DROP
-A INPUT -s 172.20.0.22/32 -j DROP
-A INPUT -s 172.20.0.21/32 -j DROP
-A INPUT -s 172.20.0.20/32 -j DROP
-A INPUT -s 172.20.0.19/32 -j DROP
-A INPUT -s 172.20.0.18/32 -j DROP
-A INPUT -s 172.20.0.17/32 -j DROP
-A INPUT -s 172.20.0.16/32 -j DROP
-A INPUT -s 172.20.0.15/32 -j DROP
-A INPUT -s 172.20.0.14/32 -j DROP
-A INPUT -s 172.20.0.13/32 -j DROP
-A INPUT -s 172.20.0.12/32 -j DROP
-A INPUT -s 172.20.0.11/32 -j DROP
-A INPUT -s 172.20.0.10/32 -j DROP
-A INPUT -s 172.20.0.9/32 -j DROP
-A INPUT -s 172.20.0.8/32 -j DROP
-A INPUT -s 172.20.0.7/32 -j DROP
-A INPUT -s 172.20.0.6/32 -j DROP
-A INPUT -s 172.20.0.5/32 -j DROP
-A INPUT -s 172.20.0.4/32 -j DROP
```

Tabla 24 – (PN=20, PC=1, NB=10, OB=5, TB=100)

3. **Blocktime=50 segundos:** en este tercer caso se ha disminuido aún más el valor a estudiar para ver qué sucede. El número de publicadores normales es de 20 y el de publicadores comprometidos es de 5, ambos con una tasa de mensaje de 1 paquete por segundo, nºbuckets y threshold ambos son 10. El resultado se va a representar en la Tabla 25:

```
-P INPUT ACCEPT
-A INPUT -s 172.20.1.8/32 -j DROP
-A INPUT -s 172.20.1.7/32 -j DROP
-A INPUT -s 172.20.1.6/32 -j DROP
-A INPUT -s 172.20.1.5/32 -j DROP
-A INPUT -s 172.20.1.4/32 -j DROP
```

Tabla 25 – (PN=20, PC=5, NB=10, OB=10, TB=50)

Observamos que, a diferencia de los dos anteriores casos, las reglas que se mantienen bloqueadas son las de los publicadores comprometidos y no las de los normales. Sin embargo, si el sistema sigue corriendo, ya que está

diseñado para que esté en continuo funcionamiento hasta que se le diga lo contrario, las reglas de los publicadores comprometidos también se borraran al no seguir recibiendo peticiones pasado el tiempo bloqueo especificado.
Y el resultado final es el siguiente:

-P INPUT ACCEPT

Tabla 26 – (PN=20, PC=5, NB=10, OB=10, TB=50)

Por tanto, después de estos tres casos, podemos concluir respecto a la variable blocktime, que cuanto mayor sea su valor más reglas bloqueará ya sea de los publicadores comprometidos como de los normales. A continuación, se estudiará la variación de más parámetros para ver su comportamiento.

4.1.2 Prueba 2: Variación de la variable Threshold

En esta segunda prueba se va a comprobar cómo reacciona el sistema ante una variación de la variable threshold, definida en la sección 3.2.3.

1. **Threshold=5**: en el primer caso que se va a ejecuta, el número de publicadores normales es de 20 y el de comprometidos 1. Además, la variable numbuckets se mantiene a 10 y el del blocktime a 10. El resultado se las reglas que se van añadiendo y eliminando es el siguiente:

```
Initializing iptables
Flushing rules
Adding rule 172.20.0.4
Adding rule 172.20.0.5
Adding rule 172.20.0.6
Adding rule 172.20.0.7
Adding rule 172.20.0.8
Adding rule 172.20.0.9
Adding rule 172.20.0.10
Deleting rule 172.20.0.4
Deleting rule 172.20.0.5
Adding rule 172.20.0.11
Deleting rule 172.20.0.6
Deleting rule 172.20.0.7
Adding rule 172.20.0.12
Adding rule 172.20.0.13
Deleting rule 172.20.0.8
Deleting rule 172.20.0.9
Adding rule 172.20.0.14
Deleting rule 172.20.0.10
Adding rule 172.20.0.15
Deleting rule 172.20.0.11
Adding rule 172.20.0.16
Deleting rule 172.20.0.12
Adding rule 172.20.0.17
Deleting rule 172.20.0.13
```

Adding rule 172.20.0.18 Deleting rule 172.20.0.14 Adding rule 172.20.0.19 Deleting rule 172.20.0.15 Adding rule 172.20.0.20 Deleting rule 172.20.0.16 Adding rule 172.20.0.21 Deleting rule 172.20.0.17 Adding rule 172.20.0.22 Deleting rule 172.20.0.18 Adding rule 172.20.0.23 Deleting rule 172.20.0.19 Adding rule 172.20.1.4 Deleting rule 172.20.0.20 Deleting rule 172.20.0.21 Deleting rule 172.20.0.22 Deleting rule 172.20.0.23

Tabla 27 – (PN=20, PC=1, NB=10, OB=5, TB=10)

Y, después de este barrido de agregados y borrado, el resultado final en las reglas que permanecen bloqueadas es el siguiente:

-P INPUT ACCEPT -A INPUT -s 172.20.1.4/32 -j DROP

Tabla 28 – (PN=20, PC=1, NB=10, OB=5, TB=10)

Como se observa solo se mantiene la de los publicadores comprometidos. Pero, al tener un número de blocktime bajo, esta regla también se eliminará pasado un tiempo de bloqueo al no seguir recibiéndose peticiones suyas.

2. **Threshold=10:** en esta segunda ejecución se va a probar con 20 publicadores normales y 5 comprometidos, además de 50 mensajes cada uno de ellos a una tasa de 1 mensaje por segundo. El valor de blocktime y el del nºbuckets en este caso se mantienen a 10. El resultado de este caso es el siguiente:

-P INPUT ACCEPT -A INPUT -s 172.20.1.7/32 -j DROP -A INPUT -s 172.20.1.4/32 -j DROP -A INPUT -s 172.20.1.5/32 -j DROP -A INPUT -s 172.20.1.6/32 -j DROP -A INPUT -s 172.20.1.8/32 -j DROP

Tabla 29 – (PN=20, PC=5, NB=10, OB=10, TB=10)

Como se puede ver en la tabla superior, las reglas que se mantienen bloqueadas son las de los publicadores comprometidos, las cuales al tener un blocktime bajo se eliminarán pasado un tiempo de bloqueo.

3. **Threshold=25:** como tercer caso se va a fijar el número de publicadores normales a 50 y el de comprometidos a 10. Además, la variable numbuckets será 10 y el blocktime 50. Con estos valores los resultados de las reglas añadidas al final de la ejecución son:

```
-P INPUT ACCEPT
-A INPUT -s 172.20.1.13/32 -j DROP
-A INPUT -s 172.20.1.12/32 -j DROP
-A INPUT -s 172.20.1.11/32 -j DROP
-A INPUT -s 172.20.1.10/32 -j DROP
-A INPUT -s 172.20.1.9/32 -j DROP
-A INPUT -s 172.20.1.8/32 -j DROP
```

Tabla 30 – (PN=50, PC=10, NB=10, OB=25, TB=50)

Observamos como de las 10 direcciones los publicadores comprometidos solo se han añadido 6.

4. **Threshold=50:** para comprobar qué ocurra si aumentamos aún más este valor, vamos a probar ahora el mismo número de publicadores que en el caso previo, pero de 50 mensajes cada uno de ellos. El resultado de las reglas añadidas y borradas es el siguiente:

```
Initializing iptables
Flushing rules
Adding rule 172.20.1.13
Deleting rule 172.20.1.13
```

Tabla 31 – (PN=50, PC=10, NB=10, OB=50, TB=50)

Como se puede apreciar, se ha agregado solo una regla perteneciente a uno de los publicadores comprometidos, pero al tener un blocktime de 50 segundos, pasado un tiempo de bloqueo esta se ha borrado.

Por tanto, si analizamos los resultados al ir variando el valor del threshold se puede observar que según se va aumentando el número de direcciones IP procedentes de equipos comprometidos que se cuela es mayor. Por lo que lo más conveniente para un sistema en uso real sería disminuir este valor para que fuera capaz de bloquear las direcciones de todos los equipos que intenten una conexión con una frecuencia alta de repeticiones.

4.1.3 Prueba 3: Variación de la variable numbuckets

En este tercer apartado de pruebas vamos a ver cómo reacciona el sistema diseñado ante la variación de esta variable.

1. **Numbuckets=10:** adaptamos el número de publicadores normales a 20 y el de publicadores comprometidos a 5. El número de mensajes enviados por cada uno es de 50 y con una tasa de 1 mensaje por segundo, además de fijar el threshold a 10 y el blocktime a 50. El resultado de las reglas añadidas es el que se recoge en

la Tabla 32 y en ella se observa cómo se bloquean las direcciones pertenecientes a los 5 publicadores comprometidos:

```
-P INPUT ACCEPT
-A INPUT -s 172.20.1.8/32 -j DROP
-A INPUT -s 172.20.1.7/32 -j DROP
-A INPUT -s 172.20.1.6/32 -j DROP
-A INPUT -s 172.20.1.5/32 -j DROP
-A INPUT -s 172.20.1.4/32 -j DROP
```

Tabla 32 – (PN=20, PC=5, NB=10, OB=10, TB=50)

Sin embargo, al tener una variable de blocktime de bajo valor, esta tabla de reglas añadidas se vaciará borrándolas todas pasados un tiempo de bloqueo al no seguir recibiendo peticiones suyas.

2. **Numbuckets=25:** en este segundo caso se han establecido 50 publicadores normales y 10 comprometidos, cada uno con 30 mensajes y una tasa de un mensaje por segundo. El threshold y el blocktime se mantienen a 10 y 50 respectivamente. El resultado obtenido ha sido el siguiente:

```
-P INPUT ACCEPT
-A INPUT -s 172.20.1.13/32 -j DROP
-A INPUT -s 172.20.1.12/32 -j DROP
-A INPUT -s 172.20.1.11/32 -j DROP
-A INPUT -s 172.20.1.10/32 -j DROP
-A INPUT -s 172.20.1.9/32 -j DROP
-A INPUT -s 172.20.1.8/32 -j DROP
-A INPUT -s 172.20.1.7/32 -j DROP
-A INPUT -s 172.20.1.6/32 -j DROP
-A INPUT -s 172.20.1.5/32 -j DROP
-A INPUT -s 172.20.1.4/32 -j DROP
```

Tabla 33 – (PN=50, PC=10, NB=25, OB=10, TB=50)

Como se ve en la Tabla 33, las reglas bloqueadas son las pertenecientes a los publicadores bloqueados, ninguna de los normales. Además, como en los dos casos anteriores, pasado un tiempo se borrarán al no seguir recibiendo peticiones suyas y tener un valor bajo de blocktime.

3. **Numbuckets=50:** para esta tercera ejecución se ha fijado a 60 el número de publicadores normales y 15 los comprometidos con 60 mensajes cada uno y una tasa de 1 mensaje por segundo. Además, el threshold y blocktime a 10 y 50. El resultado ha sido el mismo que en los anteriores, se agregan todas las direcciones IP de los publicadores comprometidos. Se muestra en la Tabla 34:

```
-P INPUT ACCEPT
-A INPUT -s 172.20.1.18/32 -j DROP
-A INPUT -s 172.20.1.17/32 -j DROP
-A INPUT -s 172.20.1.16/32 -j DROP
-A INPUT -s 172.20.1.15/32 -j DROP
```

-A INPUT -s 172.20.1.14/32 -j DROP
-A INPUT -s 172.20.1.13/32 -j DROP
-A INPUT -s 172.20.1.12/32 -j DROP
-A INPUT -s 172.20.1.11/32 -j DROP
-A INPUT -s 172.20.1.10/32 -j DROP
-A INPUT -s 172.20.1.9/32 -j DROP
-A INPUT -s 172.20.1.8/32 -j DROP
-A INPUT -s 172.20.1.7/32 -j DROP
-A INPUT -s 172.20.1.6/32 -j DROP
-A INPUT -s 172.20.1.5/32 -j DROP
-A INPUT -s 172.20.1.4/32 -j DROP

Tabla 34 – (PN=60, PC=15, NB=50, OB=10, TB=50)

Y, pasado el tiempo de bloqueo calculado para poder borrar una regla ya añadida el resultado de las reglas nos indica que las reglas han sido eliminadas al haber finalizado la emisión masiva como muestra la Tabla 35:

-P INPUT ACCEPT

Tabla 35 – (PN=60, PC=15, NB=50, OB=10, TB=50)

Después de estas pruebas de variación del número de buckets, se puede concluir que no influye en el comportamiento del sistema ya que se aumente o disminuya no afecta al bloqueo de las reglas ante un ataque DoS.

4.1.4 Prueba 4: Variación de la variable delay

En este cuarto subapartado de pruebas se va a comprobar cómo afecta la tasa de mensajes, es decir, cada cuánto envía un mensaje un publicador, al sistema completo.

1. **Delay=2 segundos**: en este primer caso se ha puesto el publicador normal a 20 y el publicado comprometido a 15 con 20 mensajes cada uno. El número de buckets y el threshold se han fijado a 10 ambos, y el blocktime a 40 segundos. El resultado ha sido el siguiente:

-A INPUT -s 172.20.1.17/32 -j DROP
-A INPUT -s 172.20.1.15/32 -j DROP
-A INPUT -s 172.20.1.16/32 -j DROP
-A INPUT -s 172.20.1.14/32 -j DROP
-A INPUT -s 172.20.1.18/32 -j DROP
-A INPUT -s 172.20.0.5/32 -j DROP
-A INPUT -s 172.20.0.4/32 -j DROP

Tabla 36 – (PN=20, PC=15, NB=20, OB=10, TB=20)

En este caso se han colado dos direcciones pertenecientes a los publicadores normales y de los comprometidos solo 5 se han mantenido bloqueadas.

2. **Delay = 3 segundos:** al aumentar más el tiempo de bloqueo (delay), en este caso 3, y con el resto de los valores idénticos al caso anterior se obtienen los siguientes resultados:

```
Initializing iptables
Flushing rules
```

Tabla 37 – (PN=20, PC=15, NB=20, OB=10, TB=20)

Es decir, se realiza el borrado inicial pero no se llega a añadir ninguna regla, no se bloquean ni los publicadores normales ni los comprometidos.

Por tanto, como conclusión del análisis de la variable de tiempo de bloqueo delay, se observa que según aumentamos su valor ninguna de las reglas se añade pues llegan con suficiente tiempo entre ellas como para no detectar un ataque DoS.

4.1.5 Prueba 5: Variación del número de publicadores

En esta última prueba se va a ir variando el número de publicadores para ver la reacción del sistema diseñado.

1. **Nº Publicadores Normales=20, Nº Publicadores Comprometidos=0**: en este primer caso se han fijado los mensajes a 50 y la tasa a 1 mensaje por segundo. Además, el número de buckets, threshold y blocktime a 10 cada uno de ellos. Una vez se ejecuta, se obtienen los siguientes resultados:

```
Initializing iptables
Flushing rules
Adding rule 172.20.0.4
Adding rule 172.20.0.5
Adding rule 172.20.0.6
Adding rule 172.20.0.7
Adding rule 172.20.0.8
Adding rule 172.20.0.9
Deleting rule 172.20.0.4
Adding rule 172.20.0.10
Deleting rule 172.20.0.5
Deleting rule 172.20.0.6
Deleting rule 172.20.0.7
Deleting rule 172.20.0.8
Deleting rule 172.20.0.9
Deleting rule 172.20.0.10
Adding rule 172.20.0.18
Adding rule 172.20.0.21
Adding rule 172.20.0.23
Adding rule 172.20.0.20
Adding rule 172.20.0.19
Adding rule 172.20.0.22
Deleting rule 172.20.0.18
Deleting rule 172.20.0.21
```

Deleting rule 172.20.0.23 Deleting rule 172.20.0.20 Deleting rule 172.20.0.19 Deleting rule 172.20.0.22

Tabla 38 – (PN=20, PC=0, NB=10, OB=10, TB=10)

Como se observa en la Tabla 38, se agregan todas las reglas, pero se borran hasta finalmente no quedar ninguna en la tabla de reglas añadidas, lo que tiene sentido al tratarse de publicadores normales.

2. **Nº Publicadores Normales=50, Nº Publicadores Comprometidos=50:** en este segundo caso se ha tratado con una tasa de 1 mensaje por segundo, y ambos enviarán 20 mensajes. El número de buckets y el threshold se han fijado ambos a 10, y el blocktime a 40 segundos. El resultado de las reglas añadidas y por tanto bloqueadas ha sido el siguiente:

-P INPUT ACCEPT -A INPUT -s 172.20.1.53/32 -j DROP -A INPUT -s 172.20.1.52/32 -j DROP -A INPUT -s 172.20.1.51/32 -j DROP -A INPUT -s 172.20.1.50/32 -j DROP -A INPUT -s 172.20.1.49/32 -j DROP -A INPUT -s 172.20.1.48/32 -j DROP -A INPUT -s 172.20.1.47/32 -j DROP -A INPUT -s 172.20.1.46/32 -j DROP -A INPUT -s 172.20.1.45/32 -j DROP -A INPUT -s 172.20.1.44/32 -j DROP -A INPUT -s 172.20.1.43/32 -j DROP -A INPUT -s 172.20.1.42/32 -j DROP -A INPUT -s 172.20.1.41/32 -j DROP -A INPUT -s 172.20.1.40/32 -j DROP -A INPUT -s 172.20.1.39/32 -j DROP -A INPUT -s 172.20.1.38/32 -j DROP -A INPUT -s 172.20.1.37/32 -j DROP -A INPUT -s 172.20.1.36/32 -j DROP -A INPUT -s 172.20.1.35/32 -j DROP -A INPUT -s 172.20.1.34/32 -j DROP -A INPUT -s 172.20.1.33/32 -j DROP -A INPUT -s 172.20.1.32/32 -j DROP -A INPUT -s 172.20.1.31/32 -j DROP -A INPUT -s 172.20.1.30/32 -j DROP -A INPUT -s 172.20.1.29/32 -j DROP -A INPUT -s 172.20.1.28/32 -j DROP -A INPUT -s 172.20.1.27/32 -j DROP -A INPUT -s 172.20.1.26/32 -j DROP -A INPUT -s 172.20.1.25/32 -j DROP -A INPUT -s 172.20.1.24/32 -j DROP -A INPUT -s 172.20.1.23/32 -j DROP -A INPUT -s 172.20.1.22/32 -j DROP -A INPUT -s 172.20.1.21/32 -j DROP

```
-A INPUT -s 172.20.1.20/32 -j DROP
-A INPUT -s 172.20.1.19/32 -j DROP
-A INPUT -s 172.20.1.18/32 -j DROP
-A INPUT -s 172.20.1.17/32 -j DROP
-A INPUT -s 172.20.1.16/32 -j DROP
-A INPUT -s 172.20.1.15/32 -j DROP
-A INPUT -s 172.20.1.14/32 -j DROP
-A INPUT -s 172.20.1.13/32 -j DROP
-A INPUT -s 172.20.1.12/32 -j DROP
-A INPUT -s 172.20.1.11/32 -j DROP
-A INPUT -s 172.20.1.10/32 -j DROP
-A INPUT -s 172.20.1.9/32 -j DROP
-A INPUT -s 172.20.1.8/32 -j DROP
-A INPUT -s 172.20.1.7/32 -j DROP
-A INPUT -s 172.20.1.6/32 -j DROP
-A INPUT -s 172.20.1.5/32 -j DROP
-A INPUT -s 172.20.1.4/32 -j DROP
-A INPUT -s 172.20.0.53/32 -j DROP
-A INPUT -s 172.20.0.52/32 -j DROP
-A INPUT -s 172.20.0.51/32 -j DROP
-A INPUT -s 172.20.0.50/32 -j DROP
-A INPUT -s 172.20.0.49/32 -j DROP
-A INPUT -s 172.20.0.48/32 -j DROP
```

Tabla 39 – (PN=50, PC=50, NB=10, OB=10, TB=40)

Como se aprecia en la Tabla 39, se han bloqueado todas las direcciones IP pertenecientes a los publicadores comprometidos, pero se han colado seis de los publicadores normales: 172.20.0.53/32, 172.20.0.52/32, 172.20.0.51/32, 172.20.0.50/32, 172.20.0.49/32 y 172.20.0.48/32.

Por tanto, como conclusión de este último apartado, se deduce que según aumentan el número de publicadores con ello aumenta el riesgo de bloquear direcciones IP que no pertenecen a los publicadores comprometidos.

A continuación, se resumen todos los resultados en una tabla para observar en qué casos los resultados son adecuados y cuáles dejan pasar direcciones comprometidas.

DELAY	1	1	1	1	1	1	1	1	1	1	2	3	1	1
PN	20	20	20	20	20	50	50	20	50	60	20	20	20	50
PC	10	1	5	1	5	10	10	5	10	15	15	15	0	50
NB	10	10	10	10	10	10	10	10	25	50	20	20	10	10
OB	5	5	10	5	10	25	50	10	10	10	10	10	10	10
TB	200	100	50	10	10	50	50	50	50	50	20	20	10	40
RESULTADO	X	X	V	V	V	X	X	V	V	V	X	X	V	V

Tabla 40 – Resultados Pruebas

5 Conclusiones y trabajo futuro

5.1 Conclusiones

El incremento del uso de Internet de las Cosas de forma generalizada ha hecho que nuestros datos personales puedan estar en continuo cuando no se aplican las medidas de seguridad adecuadas. Es por ello que la integridad de los sistemas y por tanto de los datos de usuarios y empresas, debe estar totalmente garantizada y proporcionar seguridad ante ataques tales como la Denegación de Servicio.

Por ello la implementación realizada en este Trabajo Fin de Grado ha ido enfocada a intentar garantizar seguridad ante un ataque, para lo que el primer paso ha sido usar el capturador TShark/Wireshark de tráfico que nos permitiera filtrar según el protocolo y el puerto las peticiones entrantes.

En segundo lugar, el uso de un algoritmo que nos permitiera poder analizar las direcciones IP de origen según el número de repeticiones que se dan. Para lo que se han usado diccionarios y funciones que nos han ido permitiendo ir guardando la información necesaria para, no solo localizar un ataque DoS, sino también bloquear las direcciones IP que lo ocasionan.

Para este tercer punto, poder bloquear una dirección IP que nos está atacando, se ha hecho uso de un cortafuegos basado en IPTables que nos ha permitido reconfigurarlo según nuestras necesidades. En este caso, si una dirección IP excedía el número máximo de peticiones enviadas, esta se mandaba mediante un callback para añadirla a la lista de reglas de las IPTables para impedir que siga enviando peticiones. Además, si durante un determinado periodo de tiempo se dejaban de recibir peticiones de esa dirección, se eliminaba de la lista de reglas permitiéndola volver a establecer conexión. Este sistema permite filtrar sistemas comprometidos que inyectan más muestras de las esperadas.

Para poder analizar el sistema realizado, se han llevado a cabo una serie de pruebas en la sección 4 en la que, variando los parámetros del algoritmo, se ha ido determinando en qué momentos el sistema bloqueaba las direcciones IP comprometidas y en cuáles no. También se ha determinado en qué casos se bloquean direcciones IP que no están comprometidas, o se escapaba alguna que sí que lo estaba.

5.2 Trabajo futuro

El desarrollo de este Trabajo Fin de Grado sirve como punto de partida para la utilización de un sistema de uso de cortafuegos basado en IPtables + Algoritmo Heavy Hitters para poder defender los equipos ante ataques de Denegación de Servicio. Es por este motivo, que se da la posibilidad de mejorar este sistema de diferentes maneras, entre las cuales se destacan:

- Uso de otros campos de filtrado en el módulo del capturador, como puede ser el uso de banderas TCP y/o hacer análisis de flujos más detallados teniendo en cuenta otros datos aparte de las direcciones IP y los puertos.
- Una vez se detectan ataques de direcciones IP que están próximas, bloquearlas directamente por grupos en vez de ir una a una. Esto puede ser un punto de partida para resolver ataques DoS distribuidos.

- Se puede probar en otros dispositivos de red con capacidad de cortafuegos implementando los módulos nuevos que se puedan requerir para configurarlos.
- Estudiar el rendimiento del sistema más a fondo proponiendo una solución que permita aumentar la tasa de peticiones y datos sin dañar la integridad del sistema.

Estas son algunas de las posibles mejoras, por lo que este trabajo de fin de grado puede servir como punto de partida para otros futuros que permitan desarrollar estas ideas, mejorarlas e implementarlas en sistemas de uso real.

Referencias

[1] IHS Markit 2013/ http://www.appliedmaterials.com/nanochip/nanochip-fab-solutions/december-2013/cover-story-fabs-in-the-internet-of-things-era

[2] James F. Kurose, Keith W. Ross, "Computer Networking: A Top-Down Approach", Global Edition, Pearson, 2010.

[3] https://marketing4ecommerce.net/usuarios-internet-mundo

[4] Louis Columbus, Enterprise & Cloud, December 2018, Forbes

[5] Misra, G., Kumar, v., Agarwal, A., & Agarwal, K. (2016). Internet of Things – A Technological Analysis and Survey on Vision, Concepts, Challenges, Innovation Directions, Technologies, and Applications. American Journal of Electrical and Electronic Engineering, 4(1), 23-32.

[6] Chen, K., Zhang, S., Li, Z., Zhang, Y., Deng, Q. , Ray, S. , Jin, Y. (2018) Internet of Things Security and Vulnerabilities: Taxonomy, Challenge and Practice.

[7]] Maureen Valentina Parra Mondragón, Edward Paul Guillén, "Servicios de Autenticación y Autorización Orientados a Internet de las Cosas", Revista Telemática.

[8] Informe Ciberamenazas y Tendencias 2018 CCN-CERT IA-09/18

[9] Bassi, A., Bauer, M., Fiedler. M, Kramp, T., Lange. S, Meissner. S, Kranenburg, R. "Enable Things to Talk, Designing IoT Solutions with the IoT Architectural Reference Model", Springer Open.

[10] "Distributed Denial of Service Attack and Defense", Shui Yu. Springer, 2014

[11] "Tipos de Mecanismos para la Protección de los Servicios Informáticos y sus Modelos de Seguridad", Diego Moreno, 2018-02-21

[12] "IPTABLES manual práctico", Pello Xabier Altadill Izura: http://es.tldp.org/Manuales-LuCAS/doc-iptables-firewall/doc-iptables-firewall.pdf

[13] IPTables: https://netfilter.org

[14] TShark: https://www.wireshark.org/docs/man-pages/tshark.html

[15] Python: https://docs.python.org/3.6/

[16] Heavy Hitters: https://pt.coursera.org/lecture/advanced-algorithms-and-complexity/heavy-hitters-problem-wCILS

[17] MQTT: https://www.linkedin.com/pulse/protocolo-mqtt-en-mt-331-hern%C3%A1n-canales-navarrete

[18] Docker: https://docs.docker.com/install/linux/docker-ce/ubuntu/

[19] MQTT: https://mosquitto.org/

[20] https://www.aquasec.com/wiki/display/containers/Docker+Containers

[21] Bernard Marr. "Cómo beneficiarse de un mundo de Big Data, Analytics e Internet de las Cosas", ECOE Ediciones 2018

[22] Edge Computing, Leandro Zanoni. https://www.business-solutions.telefonica.com/es/cloud-hub/knowledge-center/what-is-edge-computing/

[23] VirtualBox: https://www.virtualbox.org/

[24] PIP: https://pip.pypa.io/en/stable/

[25] IPTables: https://github.com/ldx/python-iptables

[26] Wireshark: https://www.wireshark.org/

Glosario

API	Application Programming Interface
DoS	Denial of Service
IoT	Internet of Things
QoS	Quality of Service
TCP	Transmission Control Protocol

Anexos

A Manual de instalación

En este anexo se van a detallar los procesos de descarga e instalación que se han ido realizando en el desarrollo de este trabajo. En particular, una máquina virtual si no se tiene instalado el sistema operativo Linux, Python, las reglas IPTables si no están instaladas en el computador, el capturador TShark con sus correspondientes librerías y los contenedores Docker.

A.1 Máquina Virtual

Si no se tiene el sistema operativo Linux instalado, una de las soluciones posibles es la instalación de una máquina virtual. Para ello, lo primero es activar el "Virtualization Suport" en la BIOS o "Intel Virtualization Technology", que en cada ordenador se realiza de una forma distinta. Después, en Windows en "Agregar o quitar características de Windows" hay que desactivar el "Hyper-V".

Una vez se han completado estos dos pasos, desde la página de VirtualBox [23] descargar y seguir los pasos para instalar la máquina virtual.

A.2 Instalación de Python

Para poder utilizar todos los módulos y librerías requeridos en este proyecto, ha sido necesario el uso de la versión 3.6 de Python. Para ello, primero es necesario instalar el gestor de librerías PIP o PIP3 siguiendo los pasos que se indican en [24].

Una vez realizado este paso previo, desde la página oficial de Python y eligiendo la versión que se quiere instalar [15], se descarga y se siguen los pasos del correspondiente ejecutable para poder hacer uso de este lenguaje de programación.

A.3 Instalación de reglas IPTables

En algunos sistemas operativos Linux las reglas IPTables ya vienen instaladas, pero si no es el caso, la instalación de las mismas se detalla en [25] junto con unos pasos previos para comprobar la correcta instalación.

A.4 Instalación del capturador TShark

Para poder hacer uso de esta herramienta, es necesario tener instalado en nuestro ordenador el capturador Wireshark, ya que TShark es un analizador dependiente de Wireshark. Wireshark se descarga desde su página oficial [26]. Como el lenguaje de programación usado es Python, es necesario la instalación de la librería PyShark para que pueda interactuar con TShark Para ello es necesario ejecutar el siguiente comando: *pip3 install pyshark.* Ahora, desde la página oficial de TShark [14] se siguen los pasos para su instalación.

A.5 Instalación de contenedores Docker

Para poder hacer uso de los contenedores Docker primero es necesario instalar su repositorio [18] para después poder ejecutar dos comandos desde terminal, el primero para arrancar el docker y el segundo para ver los mensajes:

docker run -it --rm --name <name> eclipse-mosquitto mosquitto_pub -h <IP-server> -t house/bulb -m "prueba" --repeat <nºrepetitions> --repeat-delay <delay>

docker run -it --rm eclipse-mosquitto mosquitto_sub -h <IP-server> -t house/bulb

Una vez se ha comprobado el correcto funcionamiento con estos comandos se va a hacer uso de unos guiones de comando (scripts) ya se puede hacer unos de ellos.