Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Estudio e implementación de un sistema de detección de intrusiones

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Salvador Cuñat Negueroles

Tutor: Alberto Miguel Bonastre Pina

Cotutor: José Carlos Campelo Rivadulla

Curso 2019/2020

Resumen

La seguridad es uno de los aspectos fundamentales para la integración de las Tecnologías de la Información y Comunicaciones (TIC) en nuevos entornos, como pueden ser Internet de las Cosas o la Industria 4.0. En este contexto, resulta imprescindible garantizar una alta fiabilidad del sistema frente a amenazas externas e internas.

Este Trabajo de Fin de Grado tiene como objetivo final el diseño e implementación de modelo de un Sistema de Detección de Intrusiones (conocido habitualmente como IDS), que permita evaluar el grado de detección, y por tanto la fiabilidad, de diferentes aproximaciones de seguridad, frente a los perfiles de ataque más frecuentes en los sistemas informáticos.

Para ello se comenzará con un análisis previo de dichos ataques; definiendo en qué consisten; cómo funcionan; cuál es su finalidad y qué beneficio pretenden obtener los atacantes con dichos ataques. A partir de este estudio se obtendrán los patrones de comportamiento que permitirán detectar los diferentes tipos de ataques.

Tras este estudio teórico, se implementará una plataforma de simulación que modele una red informática con múltiples computadores, mediante un entorno de máquinas virtuales, como por ejemplo VirtualBox. En esta plataforma se realizará la ejecución de diferentes sistemas operativos reales, algunos de los cuales ya comprometidos. Éstos últimos intentarán diferentes ataques sobre el resto.

Cuando la plataforma esté operativa, se implementará sobre ella un IDS básico, capaz de detectar los patrones de ataque y alertar al administrador sobre los mismos.

Palabras clave: Ciberseguridad, Sistemas de Detección de Intrusiones, Virtualización.

Abstract

Security is one of the fundamental aspects for the integration of Information and Communication Technologies (ICT) in new environments, such as the Internet of Things or Industry 4.0. In this context, it is essential to guarantee high system reliability against external and internal threats.

This Final Degree Project faces as final objective the design and implementation of an Intrusion Detection System model (commonly known as IDS), which allows evaluating the degree of detection, and therefore the reliability, of different security approaches, against the most frequent attack profiles in computer systems.

For this, a preliminary analysis of aforementioned attacks is required; defining what they consist of; how do they work; what is their purpose and what benefit do attackers seek to obtain from such attacks. From this study, the behavior patterns that will detect the different types of attacks will be obtained.

After this theoretical study, a simulation platform that models a computer network with multiple computers will be implemented, using a virtual machine environment, such as VirtualBox. On this platform, the execution of different real operating systems will be carried out, some of which are already compromised. The latter will try different attacks on the rest.

When the platform is operational, a basic IDS will be implemented on it, capable of detecting attack patterns and alerting the administrator of them.

Keywords: Cybersecurity, Intrusion Detection Systems, Virtualization.

Tabla de contenidos

Contenido

1. Introducción

Como es bien sabido, la seguridad en una red informática es vital hoy en día. Lo que no es tan conocido; es lo verdaderamente vulnerables que son la mayoría de las redes, o la capacidad individual de llegar a detectar dichos ataques. En este trabajo, se estudiará el efecto que provocan la mayoría de los ataques más comunes a los que puede llegar a enfrentar un profesional, así como las razones que los atacantes tienen para intentar entrar en una red y qué puede hacer una persona sin demasiada experiencia en ciberseguridad para mejorar la defensa de la red. Esto último se debe a que hoy en día se sigue malinterpretando tanto la gravedad de este tema como la facultad de uno mismo para detectar y defenderse de ciberataques.

Lo que procede ahora, es estudiar cómo se lucha contra estos ataques; ¿qué formas de detectarlos existen? ¿Cuáles son las que suelen usar las grandes compañías? Esta última pregunta puede resultar difícil de resolver, ya que si un gran sistema de detección de intrusiones desvelara sus secretos tan fácilmente podría ser explotado con facilidad. ¿Pero, y el software gratuito? Este ofrece muchas más posibilidades, ya que cualquiera puede usarlo e ir modificando el código para adaptarlo a su gusto personal.

Es de este uso del software gratuito de donde emerge la siguiente idea: exponer los diferentes métodos y estrategias que una persona puede usar para defenderse de intrusiones utilizando este software gratuito. Por ejemplo, conseguir implementar un método de detección para varios tipos de ataques de denegación de servicio usando el software gratuito *tcpdump*.

Con esto en mente, se comenzará con un estudio en profundidad de los ataques más típicos; pero no por ello obsoletos, que se siguen produciendo en la actualidad. Estos ataques siguen siendo muy utilizados y sólo es necesaria una rápida búsqueda con cualquier navegador de Internet para corroborar esto. Se consideran como ataques típicos los: ataques de denegación de servicio distribuidos, ataques de análisis de puertos e intrusiones del tipo *man-in-the-middle.* Para recrear estos ataques se ha utilizado el software VirtualBox, simulando un servidor que va a ser atacado.

El siguiente paso en este estudio consiste en formular y responder la consecuente pregunta: ¿Qué medidas puede tomar un usuario no experto en ciberseguridad para defenderse de dichos ataques, sin necesidad de tener que recurrir a software de pago de terceros? La respuesta es que se puede hacer mucho más de lo que uno puede llegar a pensar, gracias a la cantidad de herramientas de software gratuito que existen actualmente. En este proyecto se usarán varias de estas herramientas para conseguir medidas de protección similares a algunos de los módulos de sistemas de detección de intrusiones más avanzados, verificando su funcionamiento en las máquinas virtuales que hemos usado para simular los diversos ataques. El propósito es comprobar que se ha logrado detectar las intrusiones con éxito, y además demostrar

de esta forma que no es preciso disponer de una gran cantidad de recursos para conseguir resultados frente a diversos ataques simples.

1.1. Motivación

Las motivaciones a la hora de plantear el proyecto siempre fueron bastante diversas. La realidad es que cada vez hay más sistemas informáticos, ciudades inteligentes, la industria 4.0, instrumentos médicos, etc. Con todo esto la importancia de la seguridad lo único que ha hecho es ir creciendo progresivamente. Deben de garantizarse entonces cada vez más cosas como el funcionamiento constante de dichos sistemas, así como su confidencialidad. Para esto último es cierto que han salido leyes de protección de datos que intentan regularizar y legalizar el proceso.

A modo resumen, la verdad es que, hasta ahora, la seguridad se había menospreciado, pero empieza a surgir conciencia de su necesidad y complejidad. De todo esto surgió la idea de crear un pequeño sistema de detección de intrusiones, como forma de demostrar que no se necesitan conocimientos extensivos en la materia para poder mejorar la defensa de una red.

1.2. Objetivos

En el proyecto primeramente se van a estudiar diversos tipos de ataques e intrusiones más frecuentes en toda clase de redes y sistemas, ya sea en oficinas importantes o en servidores para páginas web pequeñas. Estos ataques pueden ser ataques de denegación de servicio, *man-in-the-middle* o un simple escaneado de puertos, entre otros. Acto seguido dichos ataques serán recreados en entornos controlados para ser estudiados, y encontrar patrones que permitan su identificación automática mediante un sistema de detección de intrusiones (IDS).

Con todo esto, el objetivo principal del proyecto es el siguiente: hacer un IDS usando exclusivamente software gratuito. Con esto se garantizará que sea lo más barato y asequible posible. Además de garantizar su simplicidad para ejecutarlo en máquinas de bajo coste.

Para lograr este objetivo se fueron añadiendo una lista de objetivos secundarios. Para empezar, será necesario hacer una investigación de las técnicas de detección existentes. Una vez comprendidas se simularán las técnicas de detección en entornos controlados, de donde se extraerán características y conclusiones. Ahora se automatizarán las detecciones de ataques lo máximo posible, para que la interacción directa por el usuario no sea constante. Con esto completado, estarán hechas las bases del IDS que se querrá implementar, ahora habrá que repetir los pasos vistos hasta conseguir los resultados deseados. Por último se verificará el funcionamiento del IDS.

Como objetivo secundario extra, se pretende también intentar conseguir concienciar a la población de que ellos también pueden aprender a defenderse de ciberataques, o al menos aumentar la defensa de sus sistemas ellos mismos.

1.3. Metodología

La metodología que va a ser empleada va a ser como procede:

Primeramente, se deberá analizar el estado actual del arte, desde los últimos años hasta la actualidad. Esto conllevará estudiar técnicas de detección de ataques actuales por parte de los IDS actuales (mayoritariamente gratuitos). Pero los IDS sirven para detectar ciberataques, de forma que estos también deberán de ser estudiados en profundidad.

Una vez estudiados tanto los ciberataques como los IDS que los detectan se pasará a simular dichos ataques en entornos que puedan ser controlados, lo que a su vez conllevará obtener ciertos parámetros relevantes para la detección. Ahora se creará y verificará el propio algoritmo de detección del módulo. Después de estos pasos solo quedará programar el IDS y hacer pruebas con él en un entorno simulado.

1.4. Impacto esperado

Como consecuencia de la idea presentada y desarrollada en este proyecto, se espera crear un IDS que sea a la vez barato, sencillo e inteligente. Indudablemente el sistema planteado no será perfecto, siempre cabe la posibilidad de que existan otras configuraciones que sean óptimas, esto no es un gran inconveniente; ya que se habrá cumplido el objetivo principal de crear un IDS usando única y exclusivamente software gratuito. Adicionalmente se habrá logrado también la concienciación de que cualquiera es capaz de hacer algo para mejorar la seguridad de su sistema.

Con esto en mente, lo ideal sería vender el IDS a una empresa dedicada a la investigación para que pudiera ser expandido hasta llegar a su verdadero potencial. Otra alternativa sería ir desarrollando y ampliando el sistema como software libre y gratuito, al alcance de todos.

Al acabar el proyecto, se habrá desarrollado un sistema de detección de intrusiones de muy bajo coste y una puerta a un nuevo tipo de IDS habrá quedado abierta. Invariablemente de las decisiones tempranas que se tomen sobre el futuro de la idea, lo ideal sería que el proyecto pasara a ser financiado por una compañía dedicada a la ciberseguridad para su futuro desarrollo y ampliación.

1.5. Estructura

El capítulo 2 detallará el estado del arte actual y hará una evaluación crítica a de las lagunas halladas. El capítulo 3 incluye el análisis del problema a resolver con este proyecto, así como de las primeras decisiones de diseño que apuntan las líneas de trabajo para solucionar este problema. Se plantea el plan de trabajo y una evaluación de costes en forma de un presupuesto.

El capítulo 4 comienza el cuerpo principal del proyecto, detallando el diseño del sistema de detección de intrusiones planeado y que tecnología se va a usar. En el capítulo 5 se pasa al desarrollo por módulos del sistema, acompañado con esquemas, diagramas. Posteriormente, en el capítulo 6 se describirá la fase de implantación necesaria para poner al sistema desarrollado en una situación de posibles ataques reales.

Para concluir, el capítulo 7 mostrará los resultados de las pruebas para verificar que todo funciona correctamente y que se han cumplido los objetivos planteados de forma exitosa. Finalmente, el capítulo 8 será un apartado de conclusiones en el que se resumirán los resultados obtenidos y el cómo se han llegado a ellos.

2. Estado del arte

Para comenzar a analizar el estado del arte en la actualidad, primeramente, se ha de acotar la fecha a partir de la cual se considera actual el estado en el que se encuentra el mundo de la ciberseguridad. Dicha fecha en el caso de este proyecto está marcada como el año 2016, a partir del cual se analizarán algunos casos de ataques. Se ha considerado que fue el año 2016 aproximadamente cuando la seguridad pasó a ser un problema para el usuario general. Ejemplos de grandes ataques sobre estas fechas los ataques a DYN y GitHub en 2016[1] y 2018[2] respectivamente.

Desde diciembre de 2017 a finales de noviembre de 2019, *Akamai Research* ha pasado a registrar más de 9000 ataques de denegación de servicio (DDOS), mayormente dirigidos a empresas de unos campos muy específicos[3]. La industria de la alta tecnología y de los videojuegos son las dos áreas más atacadas, seguidas de la industria financiera. Pese a estar en tercer lugar, los ataques a empresas financieras abarcan un total del 42% en cuanto a número de empresas atacadas concierne, dado que los ataques a empresas tecnológicas y de videojuegos, pese a ser mayor en número; son muy localizados en pocas empresas. Entre todos estos ataques la cifra total permanece relativamente estable en cifras medias, sin embargo, el volumen de tráfico involucrado en estos ataques en ocasiones supera el orden de de terabytes por segundo[4].

Como ejemplos de grandes ataques, algunos de los mayores ocurrieron en 2016 con DYN y en 2018 con GitHub, que sufrieron ataques de 1.2Tbps y 1.35Tbps respectivamente. Las consecuencias de estos ataques fueron considerables, especialmente en el caso de DYN, ya que este controla gran parte del sistema DNS. Opuestamente a esto, GitHub consiguió detectar de forma temprana el inicio del ataque y avisar a su proveedor de seguridad, que se encargó de analizar el tráfico y redirigir el legítimo a los servidores de GitHub.

Con el paso de los años, ocurrencias como estas dejan de ser llevadas a cabo exclusivamente por computadores, y se ve como los participantes son cualquier tipo de dispositivo IoT (Internet of Things), dispositivos que han sido comprometidos por atacantes, de forma que puedan ser controlados remotamente. En el IoT, diferentes dispositivos electrónicos con acceso a internet se usan para expandir la red, convirtiéndose en accesibles e identificables por ella[5]. Un atacante puede comprometer estos dispositivos si son vulnerables al quedar comprometida una red.

Los motivos que pueden causar un ataque DDOS pueden ser muy variables. Por ejemplo, pueden ser para causar pérdida de reputación entre los clientes o incluso para distraer mientras se intenta hacer otro ataque más sutil y sofisticado. Independientemente de las razones, la necesidad de un IDS se hace cada vez más evidente, y no solo a nivel de grandes empresas sino prácticamente a nivel del propio hogar. El ciudadano medio no quiere que le roben la información o, simplemente, que sus dispositivos IoT pasen a formar parte de un ejército de *bots* (dispositivo que ha pasado a convertirse en herramienta usada por atacantes para esparcir *software maligno* por una red, infectando otros dispositivos) de un delincuente[6].

Naturalmente, los problemas a los que se puede enfrentar un particular en su domicilio no tienen nada que ver con los que se enfrenta una gran corporación en su granja de servidores. Sin embargo, pueden tener una solución más compleja pues dependen habitualmente del servicio de proveedores o terceros, que a menudo resultan ser sorprendentemente cuestionables. El hecho de que la *bot-net Mirai*[7] (red de *bots* IoT infectados por *malware* usada en ataques de denegación de servicio a *KrebsOnSecuruty, Dyn, OVH,* etc) utilizara cientos de miles de dispositivos conectados a internet, a los cuales sus usuarios les prestan poca o ninguna atención siempre y cuando parezca que cumplen su función, demuestra dolorosamente la incapacidad o el desinterés de muchos proveedores tecnológicos de mantener actualizados sus dispositivos con el paso del tiempo.

Inevitablemente, estos dispositivos desactualizados se convierten en nidos de vulnerabilidades sin parchear y, en consecuencia, candidatos perfectos para las *bot-nets*. Entre estos dispositivos, los más comúnmente identificados como vulnerables incluyen *routers* o *webcams*; ya que ¿Quién sabe si su compañía telefónica le ha actualizado el *software* o el *firmware* de su *router*? ¿Es posible efectuar esa actualización, acaso? Todas estas cuestiones se pueden aplicar, y de hecho están siendo aplicadas, a desarrollos tan sensibles de cara al futuro como los de la automoción automatizada, para la que pueden suponer un obstáculo insalvable.

2.1. Herramientas de uso común en la detección de intrusiones

El uso de un IDS en el hogar puede prevenir alguna de estas situaciones desagradables. De una rápida búsqueda en la web se deduce que las herramientas más ampliamente utilizadas en detección de intrusiones son las siguientes:

- *Snort:* sistema de detección y prevención de intrusiones en redes software gratuito de código abierto. Analiza tráfico, flujos de datos, protocolos y hace comprobaciones de contenido a tiempo real en una red mediante un anásis paquete por paquete. En el análisis se pasan los paquetes por una serie de normas escritas por el usuario para ver si se detectan ataques. Puede procesar archivos *pcap* y usa *scripting Lua*[8].
- *Suricata:* motor de detección de amenazas de código abierto gratuito. Capaz de detectar intrusiones a tiempo real, prevenirlas y monitorizar la seguridad de una red. Capaz de procesar archivos *pcap* y usar *scripting Lua.* Utiliza herramientas como *Splunk* para recolectar y analizar datos[9].
- *Security onion:* distribución gratuita de código abierto de Linux. Sirve principalmente para monitorizar seguridad en empresas e incorpora otros muchos programas gratuitos, entre los cuales están *snort, zeek* y *suricata.* Esto permite que soporte todo tipo funciones, entre las cuales está la de un IDS[10].
- *Sagan:* software gratuito de análisis de *logs* a tiempo real. Escrito en C con una arquitectura multi hilo que le proporciona un gran rendimiento. Tiene

una estructura de reglas similar a *snort* y *suricata* para poder correlacionar eventos con sistemas IDS basados en *snort y suricata.* Tmabién puede escribir en las bases de datos de estos dos programas[11].

- *Zeek:* Previamente conocido como *Bro,* es un software que lleva en desarrollo desde los años 90. Analiza tráfico de red y lo interpreta para crear *logs* de las transacciones de datos. Tiene un formato de salida completamente personalizable, apropiado tanto para revisiones manuales en disco o mediante alguna herramienta de análisis[12].
- *Solarwinds:* software de monitorización de redes que permite detectar, diagnosticar y resolver problemas de rendimiento y ataques entre otros. Este software es de pago, pero es altamente escalable. Permite correlacionar datos de red de varias fuentes rápidamente[13].

2.2. Clasificación de los IDS [14] [15] [16]

- Dependiendo de su ubicación, los IDS pueden clasificarse en:
  - *NIDS*: *Network based IDS*. Sistemas basados en red, que escanean el tráfico que circula por una red. Estos sistemas pueden ser independientes de los nodos de la red.
  - *HIDS*: *Host based IDS*. El sistema está ubicado en un host y no escanea el tráfico de la red, sino los registros generados por la propia máquina o por otras.

Generalmente, y para garantizar la mayor seguridad posible, una red debería contar con ambos tipos de IDS. La mayoría de los sistemas actuales de detección soportan ambos de forma predeterminada, o permiten integrar otras herramientas que los complementen.

- Dependiendo de su método de detección se pueden clasificar en:
  - *Signature Based*: Analizan las "firmas" conocidas de los diferentes tipos de ataque. No pueden detectar ataques nuevos.
  - *Anomaly Based*: Consisten en detectar anomalías en el tráfico de red o en los registros del sistema. Requieren establecer previamente que es "normal" y que no lo es. Esto se consigue “entrenando” a los sitemas.

Como en el caso anterior, ambos métodos son soportados por la mayoría del software actual, bien nativamente, bien integrando información generada por herramientas externas.

- Dependiendo del método de detección de las anomalías:
  - Basados en estadísticas: utilizan algoritmos estadísticos para procesar el tráfico de red.
  - Basados en patrones: identifican caracteres, formas y patrones.
  - Basados en reglas: utilizan "firmas" para detectar ataques potenciales.
  - Basados en estados: examinan el flujo de eventos con algoritmos probabilísticos.

- Basados en heurística: utilizan diferentes técnicas propias de la I.A. como algoritmos genéticos, redes neurales, o inteligencia de enjambre para diferenciar el tráfico normal del anómalo[17].

Los sistemas que más desarrollo e investigación están viendo en los últimos años son los basados en inteligencias artificiales. El objetivo natural por conseguir en este tipo de sistemas es el aprendizaje del propio sistema sin supervisión o con una supervisión lo más limitada posible. Las elevadas necesidades de estos sistemas en hardware y software acaban siendo limitaciones que los acaban restringiendo a grandes corporaciones o al mundo de la investigación. [18] [19] [15]

Los sistemas citados en el apartado 2.1 pueden ser clasificados según estos criterios. La clasificación se muestra en la siguiente tabla:

Software	**NIDS**	**HIDS**	**IPS**	**SB**	**AB**	**Otras**	**GUI**	**Opensource**
Snort	Si	No	No	Si	Si	No	No	Si
Suricata	Si	No	Si	Si	Si	Si	Si (***)	Si
Security onion (*)	Si (***)	Si (***)	Si (***)	Si (***)	Si (***)	Si (***)	Si (***)	Si
Sagan	Si (**)	Si	No	Si	Si	Si	Si (***)	Si
Zeek	Si	No	No	Si	Si	Si	Si (***)	Si
Solarwinds SEM	Si (**)	Si	Si	Si	??	No	Si	No

(*) No es un programa sino una versión de *Ubuntu* especialmente diseñada
(**) Nativamente solo HIDS, pero puede procesar datos de *snort* en tiempo real
(***) Usando software de terceros

2.3. Descripción y clasificación de ataques

Ataques de denegación de servicio (DOS)[20]: El funcionamiento de un DOS es bastante simple: consiste en enviar a una dirección la mayor cantidad posible de paquetes (en el caso de este proyecto, simples *pings*) en el menor tiempo posible, garantizando de esta forma una congestión de la red que como poco causará retrasos en el flujo de paquetes y, en situaciones más extremas con redes menos preparadas; puede incluso colapsar la red entera. Esto es lo que se conoce como un *ping flood.* El servidor atacado es se congestiona porque, sin medidas de protección, intentará responder con una cantidad equitativa de respuestas a los *pings* recibidos, y la cantidad *pings* bastará para sobrecargar el sistema.

Un tipo de ataque de denegación de servicio adicional que se ve en el proyecto es el *mac flood,* es decir, ataque por inundación de direcciones MAC. Este ataque consiste en mandar la mayor cantidad imaginable de paquetes ARP en el menor tiempo posible, de forma que se inunde la red de ellos. Esto es un problema que pasa a afectar principalmente a los *switches* de la red, ya que se inundan sus tablas ARP, que usan para el redireccionamiento de paquetes. Esto esencialmente hace que los

switches pasen a funcionar como *hubs,* ya que las tablas ARP dejan de ser utilizables ante la inmensa cantidad de paquetes. De esta forma, puedes temporalmente inundar una red, aunque no es terriblemente efectivo a la hora de hacer daño real. Este tipo de ataques se usan principalmente como distracción para hacer ataques más sofisticados y potencialmente peligrosos.

Otros tipos de ataques de denegación de servicio comunes son el ataque *ping of death* y el *TCP SYN flood.* Estos ataques quedan fuera del ámbito del proyecto, pero serán brevemente explicados a continuación:

En un ataque *ping of death* (POD) un host envía cientos de peticiones ICMP con un tamaño de paquete muy grande, de forma que el host atacado está tan ocupado respondiendo a estas peticiones que no puede dar servicio a sus clientes. Por el otro lado, en un ataque *TCP SYN flood* el atacante se aprovecha del protocolo estándar de TCP de tres vías (*three-way handshake*) enviando peticiones de conexión con direcciones de retorno inválidas, de forma que el host se queda bloqueado esperando a la respuesta.

Ataques de escaneado de puertos[21]: Un ataque de escaneo de puertos resulta en, como bien dice el nombre, escanear los puertos de una máquina para ver cuales tiene abiertos (también se pueden escanear todas las máquinas de una red si utilizas herramientas como *nmap*), para así obtener información importante sobre el sistema. Esto suele preceder un ataque, aunque los atacantes intentan averiguar si los defensores de la red monitorizan regularmente los puertos, ya que si un defensor detecta un escaneo de puertos ajeno se va haciendo una idea de que un ataque le seguirá.

En un ordenador hay típicamente 65.536 puertos, aunque típicamente no se escanean todos. Pueden ser agrupados en tres grupos: conocidos (0 - 1023), registrados (1024 - 49151) y dinámicos/privados (49152 - 65535). Un escaneo de puertos se dedica ahora a enviar un mensaje a cada puerto, uno por uno y a esperar una respuesta. Esto determinará si se puede hacer un ataque por ahí o no. Las técnicas de escaneo que se van a ver en este proyecto se detallan a continuación:

Escaneo sigiloso: diseñado para no ser detectado por herramientas de auditoría. Esto lo hace enviando paquetes TCP al destinatario con *flags* como SYN y FIN, que son consideradas sigilosas.

Escaneo TCP: puede llamarse así, pero una conexión TCP completa nunca llega a establecerse realmente. De hecho, si se establece completamente y el atacante puede ver que el puerto está aceptando conexiones, puede lanzar un ataque inmediatamente.

Escaneo UDP: el tipo menos usado, intenta encontrar puertos abiertos mediante UDP, pero como este es un protocolo sin conexión, no suele ser usado.

Ataques *man-in-the-middle*[22] [23]*:* La idea básica de un ataque como este para el atacante consiste en introducirse entre dos máquinas en una red, generalmente entre un servidor y el *router* de salida de la red; denominado habitualmente como *Gateway.* De esta forma todo el tráfico de entrada o salida del servidor pasa por la máquina del atacante, antes de ser redirigido hasta su destino.

Este es pues un ataque relativamente sofisticado, ya que habitualmente requiere alguna clase de preparación y es mucho más sutil que algunos de los otros vistos previamente en este proyecto, como el de denegación de servicio. De hecho, los ataques de denegación de servicio hoy en día suelen emplearse como método de distracción, todo con el fin de poder cometer otros actos nefarios del estilo de un ataque MITM, entre otros.

Hay múltiples tipos diferentes de ataques *man-in-the-middle,* que van a ser explicados brevemente a continuación*:*

Arp poisoning/spoofing: El atacante asocia su dirección MAC con la IP de otro host, causando que todo el tráfico que iba a ir hacia el host pase en su lugar por la máquina del atacante antes de dirigirse al host y viceversa con la respuesta.

IP spoofing: el atacante se camufla como una aplicación legítima alterando las cabeceras de los paquetes. De esta forma, los clientes que intenten acceder a la url asociada con la aplicación acaban en la página del atacante.

HTTPS spoofing: envía un mensaje de aprobación falso al navegador de la víctima cuando se hace la petición para verificar si el sitio es seguro. Esto esencialmente redirecciona el navegador a una web comprometida, en la que el atacante puede monitorizar las interacciones con la web.

DNS spoofing: hace que la víctima entre en una dirección comprometida haciéndose pasar por una web legítima por DNS.

SSL hijacking: El proceso SSL (*Secure Sockets Layer*) tiene lugar cuando una máquina se conecta a un servidor no seguro y el propio servidor lo redirige automáticamente al servidor seguro. Pues bien, aquí el atacante usa una máquina para hacer de intermediaria en el proceso SSL, interceptando esa comunicación y efectivamente situándose entre el servidor y el cliente.

Email hijacking: ocurre cuando un atacante envía un correo infectado y el usuario cae en la trampa de pensar que es fiable. A partir de aquí el atacante es capaz de acceder a los correos de la víctima.

Wi-Fi eavesdropping: consiste en establecer una red wifi fraudulenta, de forma que cuando un usuario se conecte el atacante pueda monitorizar todo el tráfico que envía. Esto también se aplica cuando un atacante se conecta a una red wifi abierta por ejemplo.

Session hijacking: mediante por ejemplo un ataque *cross-side scripting* un atacante es capaz de obtener la *cookie* de sesión de un usuario, ganando d esta forma acceso a dicha sesión para robar datos.

2.4. Calidad de los IDS[24].

Es bastante complejo conseguir establecer satisfactoriamente la eficiencia de un sistema de detección de intrusiones, y más difícil aún comparar los diferentes modelos existentes. Esto es debido en gran parte a la más que dudosa calidad de los conjuntos de datos disponibles a utilizar en dichas comparaciones. En muchos casos los datos ya están anticuados, no se parecen remotamente a lo que se podría considerar como tráfico real o básicamente han sido generados concretamente para un proyecto. En casos más extremos se han visto IDS que se habían ajustado exclusivamente al conjunto de datos considerado, sin pensar mucho en la variabilidad que una situación real puede tener.

En cualquier caso, un IDS siempre debe de intentar alcanzar un ratio de detección lo más elevado que sea humanamente posible y en consecuencia un ratio de falsos positivos extremadamente bajo, entendiendo por tales ratios[1]:

$$Ratio\ de\ detección = \frac{Verdaderos\ positivos}{Verdaderos\ positivos + Falsos\ negativos}$$

$$Ratio\ de\ falsos\ positivos = \frac{Falsos\ positivos}{Falsos\ positivos + Verdaderos\ negativos}$$

Estas afirmaciones serían válidas para una red comercial, donde un IDS rechazaría el tráfico legítimo en un falso positivo. Sin embargo, para una red casera, el que debería de ser mínimo es el ratio de falsos negativos, aún a costa de incrementar la cantidad de falsos positivos detectados, ya que lo que se busca es maximizar la protección sin consideraciones comerciales.

$$Ratio\ de\ falsos\ negativos = \frac{Falsos\ negativos}{Verdaderos\ positivos + Falsos\ negativos}$$

2.5. Crítica al estado del arte y propuesta

Con todo lo discutido anteriormente, resulta obvio que la situación actual no es perfecta en lo que respeta a seguridad de sistemas. Algunos de sus mayores problemas surgen del hecho de que a menudo las compañías son mucho más descuidadas en las actualizaciones de lo que debieran, o simplemente no se preocupan o no contemplado la posibilidad de actualizar el hardware que les suministran a sus clientes.

A la hora de hablar de sistemas de detección de intrusiones la cosa cambia, pero eso no quiere decir que dejen de existir problemas. Uno de sus mayores problemas, objeto principal de este proyecto, es el avanzado conocimiento requerido para administrar y mantener un IDS en funcionamiento. Es objetivo del proyecto,

proporcionar un IDS cuya configuración y mantenimiento requiera un bajo nivel de conocimiento previo, para que esté al alcance del mayor número posible de usuarios.

Algunos otros problemas que un IDS puede tener que se intentarán corregir en este proyecto son: el problema de los falsos positivos y de direcciones falsas, en este proyecto en particular direcciones MAC. Con todo esto en mente, a continuación, se plantean formas que en este proyecto se usarán para evitar caer en estos mismos problemas.

Sobre las direcciones falsas, se usará software que priorice la cantidad de paquetes recibidos antes que su origen, de forma que, si en un *log* de *tcpdump* aparecen miles de paquetes ARP en cuestión de segundos, todos ellos con direcciones falsas; se siga detectando y avisando del ataque.

Por el otro lado, con el tema de los falsos positivos y la experiencia previa requerida, se ha tenido que alcanzar un compromiso. Esto se debe a que, pese a que el proyecto es fácil de entender para cualquier persona y que en las pruebas realizadas apenas se han detectado falsos positivos, siempre es posible que se haya tenido algún desliz y alguna posibilidad no se haya contemplado. Por esto, se ha querido reducir lo máximo posible la posibilidad de que un usuario tenga dudas serias de si está siendo víctima de un ataque o no.

La forma empleada de hacer esto es con mensajes de aviso en ciertas situaciones, avisando de que tal vez sería conveniente darle un repaso rápido a los *logs* generados para asegurarse de que todo va bien. De todos modos, esto solo sucede a veces, ya que hay ciertas situaciones en las que simplemente no puede caber cualquier clase de duda, no se puede dudar de que se está siendo atacado si uno recibe docenas de miles de *pings* en meros segundos. Tal vez para *Google* esto sea algo cotidiano, pero para una persona normal que solo quiere aumentar la seguridad de su red desde luego que no.

3. Análisis del problema.

La mayoría de los negocios o particulares prefieren simplemente instalarse un antivirus gratuito o contratar los servicios de terceros para garantizar su seguridad, y en muchos casos no es una idea tan terrible. Es cuando empiezan a aparecer casos como la mencionada anteriormente *bot-net Mirai* que uno empieza a cuestionarse la verdadera validez de poner una fe ciega en productos ofrecidos por terceros.

Es este el problema que se trata de solucionar, la falta de un software gratuito, completo y fácil de aprender que pueda desarrollado y ampliado fácilmente, sin un conocimiento a fondo del software. Todo con el propósito de no depender en ningún momento de compañías que ocultan el cómo funcionan sus productos de software. Por esto mismo se ha decidido usar como ayuda software gratuito exclusivamente.

Para lograr esto se establecerán una serie de criterios mínimos:

- Tiene que ser un IDS sencillo de usar y configurar.
- Tiene que ser barato, usando exclusivamente software libre.
- Tiene que al menos detectar los siguientes ataques:
    - Denegación de servicio.
    - Escaneado de puertos.
    - *Man-in-the-middle.*

Teniendo en mente los problemas de fiabilidad de algunas compañías que hasta la fecha parecían completamente fiables, así como el desconocimiento por parte del usuario medio a la hora de defenderse contra ciberataques, se ha considerado que éste sería un buen campo para desarrollar el proyecto, y con el mismo proporcionar a usuarios no expertos de una herramienta que les permita al menos detectar un ciberataque.

3.1. Identificación y análisis de soluciones posibles

El siguiente paso que dar debe ser el de crear uno mismo una forma de reducir las probabilidades de ser atacado con éxito sin darse cuenta, ya que la mayoría de los ataques exitosos no se descubren hasta varias semanas o incluso meses de que ocurran. Una notable excepción de esto son los casos de ataques DOS.

Una de las formas iniciales a las que se llegaron para el cómo hacer el software del proyecto es hacer programas complejos que detectaran posibles ataques analizando paquetes potencialmente maliciosos con todo detalle. Lo conseguirían investigando todos y cada uno de los campos de dicho paquete hasta encontrar algo que llamara la atención; algo que fuera inusual y que podría señalar que se está siendo atacado. Esta opción fue descartada porque no sería eficiente en lo más mínimo ir mirando todos y cada uno de los paquetes que recibiera una máquina para decidir si ese en concreto está infectado o no.

Después de esto se pensó en desarrollar primero una interfaz en C#, por ejemplo, como las vistas a lo largo de la carrera; para que el usuario tuviera la máxima cantidad posible de control a la hora de decidir cómo iba a usar las herramientas y que es lo que quería hacer con ellas. Esta idea también se vio descartada cuando se pensó que el propósito del proyecto es hacer las cosas lo más simples posibles, a la vez que efectivas, y esta interfaz lo único que conseguiría sería aumentar innecesariamente la complejidad del proyecto. Aún sí se decidió que de esta propuesta se iba a sacar la idea de la interfaz, sólo que se haría de forma bastante más simple a la pensada originalmente.

La última opción planteada, que resultó ser la elegida, consiste en lo siguiente: Hacer una serie de programas usando el lenguaje de programación C. Dichos programas deben entonces de usar herramientas no demasiado difíciles de comprender ni usar para determinar si el usuario está siendo atacado. Como complemento a esto se decidió crear una interfaz más simple y básica, que usando el propio terminal desde el que se ejecuta ayudara al usuario a usar la herramienta, ofreciendo funcionalidades básicas como la modificación de varios parámetros.

3.2. Solución propuesta

La solución que finalmente se ha determinado como la más adecuada para el problema planteado, y la que finalmente se acabará usando, consiste en programar en C una serie de herramientas o programas, que funcionarán en conjunto con otras herramientas utilizadas en el mundillo de la ciberseguridad, aumentando de forma considerable el índice de seguridad de la red o sistema en el que se instale.

Las fases de desarrollo serán las siguientes:

1. Diseñar y montar una red con máquinas virtuales en VirtualBox para hacer todas las pruebas necesarias.
2. Plantear que tipos de ataques se van a simular y cómo prevenirlos.
3. Estudiar dichos ataques para saber cómo funcionan y como se podrían simular.
4. Hacer un “boceto” o plan inicial de cómo van a estar estructurados los programas, como van a funcionar y cómo van a interactuar entre ellos.
5. Ponerse a programar.
6. Hacer las pruebas pertinentes.

Con esto se espera conseguir un pequeño sistema de detección de intrusiones eficiente que consiga detectar exitosamente ataques e intrusiones en un sistema.

3.3. Plan de trabajo

Primeramente, era preciso obtener información de cómo funcionaban otros sistemas de detección de intrusiones, para tener un punto de partida. Esto era antes siquiera de que se tuviera una idea clara y concisa de qué se iba a hacer exactamente, o como iba a hacerse. De esta forma, se calculó erróneamente lo que se iba a tardar en hacer una investigación a fondo previa al comienzo del desarrollo.

Desde bastante al principio se supo que los primeros pasos necesarios para asentar las bases del proyecto, es decir, establecer cómo iban a estar configuradas las máquinas virtuales; no iba a llevar mucho tiempo, ya que a lo largo de la carrera se había visto en bastante profundidad el funcionamiento de estas máquinas. Esto fue lo único en lo que se acertó en las predicciones iniciales de cuanto iba a tardar en hacerse el proyecto.

Ahora se tenía otro problema en frente, y es que no se tenía claro siquiera en que lenguaje se iba a programar. Mientras se decidía esto, era preciso que, aparte de analizar algunos sistemas de detección de intrusiones para tener una idea de cómo hacer este, también se analizaran para verificar el estado del arte actual del mundillo de la ciberseguridad. Esto fue algo que también resulto muy escaso en las predicciones temporales iniciales.

Pese a estos problemas predictivos y con grandes retrasos, el proyecto siguió adelante poco a poco, hasta que llegó la hora de ponerse a programar el sistema en sí. Fue ahora cuando empezaron a surgir los problemas reales, ya que esto resultó ser bastante más complejo de lo que se imaginó inicialmente.

Pese a todos estos problemas y retrasos, el proyecto fue avanzando progresivamente, y finalmente, se decidió que se iba a hacer un nuevo plan de trabajo, este con los tiempos actualizados. Los tiempos están en días, teniendo en cuenta que se trabajaban unas cuantas horas al día, claramente no las 24:

1. Recaudación de información.	Unos 25 días.
2. Estudio del estado del arte.	Unos 30 días.
3. Preparación de máquinas virtuales.	3 días.
4. Conceptos y esquemas iniciales de los programas.	3 días.
5. Desarrollo del módulo de denegación de servicio.	16 días.
6. Desarrollo del módulo de escaneo de puertos.	14 días.
7. Desarrollo del módulo de *man-in-the-middle.*	11 días.
8. Desarrollo de los scripts.	14 días.
9. Mejoras y optimización de código.	9 días.
10. Pruebas iniciales y *beta testing*.	2 días.
11. Correcciones al código.	7 días.
12. Pruebas finales.	3 días.
13. Redactar la memoria.	38 días

3.4. Presupuesto

Considerando continuamente que se está trabajando en un proyecto de desarrollo, el presupuesto va a ser superior al de un trabajo convencional. Se ha aproximado un coste de 30€ por hora. En un entorno en el que participa uno a tiempo completo, contratado por la empresa, quiere decir que serían unos 36.000€ anuales de gastos en personal. Razonablemente se le tiene que sumar a esta cifra el coste monetario que tendrá comprar un equipo potente para hacer unas pruebas. Esto si se quiere hacer usando exclusivamente máquinas virtuales en un solo equipo, lo cual no es recomendable a la hora de escalar. Lo razonable sería comprar varios dispositivos, *raspberry* por ejemplo, para simular entonces una red y hacer las pruebas.

Teniendo en cuenta comentarios anteriores, el presupuesto se quedaría en algo similar a la tabla siguiente:

Tarea	Horas	Coste
1. Recaudación de información.	75 horas	2.250€
2. Estudio del estado del arte.	90 horas	2.700€
3. Preparación de máquinas virtuales.	9 horas	270€
4. Conceptos y esquemas iniciales de los programas.	9 horas	270€
5. Desarrollo del módulo de denegación de servicio.	48 horas	1.440€
6. Desarrollo del módulo de escaneo de puertos.	42 horas	1.260€
7. Desarrollo del módulo de *man-in-the-middle.*	33 horas	990€
8. Desarrollo de los scripts.	42 horas	1.260€
9. Mejoras y optimización de código.	18 horas	540€
10. Pruebas iniciales y *beta testing*.	6 horas	180€
11. Correcciones al código.	21 horas	630€
12. Pruebas finales.	9 horas	270€
13. Tres máquinas *raspberry* para simular el entorno.	-	100€
Total		9.910€

4. Diseño de la solución.

4.1. Arquitectura del sistema

El proyecto va a presentar una serie de módulos que se usarán en conjunto para crear el IDS. Los módulos se describen a continuación:

Módulo de soporte de máquinas virtuales: se ha implementado un entorno virtual usando el software VirtualBox, este entorno virtual se considera el primer gran bloque de desarrollo, ya que supone la base a partir de la cual se ha logrado desarrollar con éxito las simulaciones de los ataques, y las diversas formas logradas para detener dichos ataques. El entorno tiene creada una red NAT interna para que las máquinas virtuales se comuniquen entre ellas. El rango de direcciones que se ha configurado (habilitando DHCP) es 10.0.2.0/24.

Módulo de denegación de servicio: se crean dos programas en C que se encargarán de verificar que el sistema no está siendo atacado por ataques DOS. Esto lo harán mediante el uso de *tcpdump.* Capturarán paquetes que se muevan por la red, los analizarán y a partir de ahí determinarán si se está bajo ataque o no. Un script lo complementará con una interfaz e interacción con el usuario.

Módulo de escaneo de puertos: un único programa en C que esta vez empleará *snort* para detectar si saltan las respectivas alertas de ataques de escaneo de puertos que vienen con el programa. Una interfaz también lo complementará mediante un script con interacción en el usuario para modificar ciertos campos.

Módulo de *man-in-the-middle:* un único *script* que se ejecutará pidiendo a la máquina deseada cuál es su dirección del *Gateway*. Una vez obtenida se comparará con un valor que habrá sido introducido por el usuario. Este valor se meterá en la interfaz diseñada, que actuará de forma complementaria.

Los componentes esenciales para estos bloques vienen en forma de una serie de archivos programados en C, así como de múltiples scripts que se aseguran de ejecutar todos los comandos necesarios, recabando la información imprescindible que los programas a continuación usarán con el fin de determinar si el sistema está siendo atacado. Se han de incluir en esta categoría necesariamente las herramientas que desde Kali se han usado para atacar al Ubuntu.

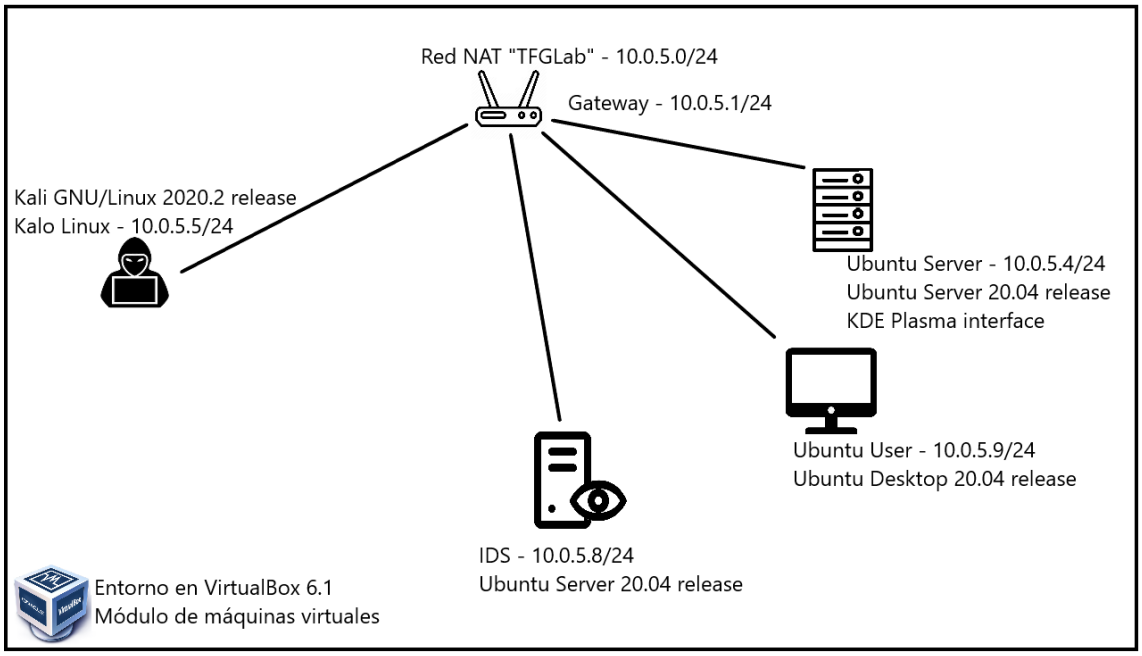

Figura 1: Arquitectura del módulo de soporte de máquinas virtuales

4.2. Tecnología utilizada.

Para empezar, hay que hablar de la base que se ha escogido para hacer todo el proyecto, el software de máquinas virtuales *VirtualBox*, esto se debe principalmente a su inexistente precio y el previo conocimiento a la hora de manejarlo. En *VirtualBox* se han creado dos máquinas, una ejecutando *Kali Linux* y la otra *Ubuntu Server.*

Entre la máquina *Kali* y la *Ubuntu* se han llegado a usar múltiples de las docenas de programas y comandos disponibles, además de los propios sistemas operativos y la base para las máquinas virtuales que los ejecutarán. Todo esto se pasa a detallar a continuación:

- *VirtualBox:* software usado como base para todo el proyecto, sobre este software se instalarán las máquinas virtuales. Escogido por su disponibilidad, facilidad de uso y pasada experiencia con él.

- *Kali Linux:* sistema operativo usado muy extensivamente en *pentesting* en todo el planeta. Es una distribución de *Linux* que ha sido diseñada para tener la mayor cantidad de programas y herramientas que podrían llegar a usarse para cualquier necesidad de *pentesting.*

- *Ubuntu Server*: sistema operativo que es una derivación de *Ubuntu,* que a su vez es una distribución de *Linux.* Tiene toda clase de funciones y capacidades que podrían estar disponibles para un servidor y se consideró apropiada para hacer las pruebas. Además, si no se instalan ninguna de las funcionalidades de servidor, se obtiene un sistema muy básico que consume muy pocos recursos. Idóneo para ordenadores de bajo coste.

- *Ubuntu desktop*: como para *Ubuntu Server,* es un sistema operativo gratuito usado a lo largo y ancho de todo el planeta, que servirá para hacer tanto de usuario de la red como de máquina en la que se instalará el IDS.

- *hping3:* Usado para hacer una inundación de pings a la máquina servidor como parte del módulo de detección de ataques tipo denegación de servicio. Es capaz de mandar docenas de miles de pings en segundos con las opciones adecuadas y demostró ser una de las formas más sencillas y efectivas encontradas. Como añadido final, consume poca cantidad de memoria.

- *macof:* Este comando, parte del módulo de detección de ataques tipo denegación de servicio; es similar al anterior en que envía una cantidad enorme de paquetes, pero en este caso envía gran cantidad de direcciones MAC, con el objetivo de colapsar las tablas CAM (*Content Addresssable Memory*) de un *switch.* Cumple los mismos requisitos que la herramienta anterior para que haya sido escogida.

- *nmap:* un comando bastante conocido y utilizado, *nmap* te ofrece una gran variedad para "mapear" una red entera, aunque en este proyecto se han usado exclusivamente sus capacidades para encontrar puertos abiertos en un solo ordenador. Fue escogida entre otras razones por su rapidez, facilidad de uso y su versatilidad; no tienes disponible exclusivamente una forma de hacer el escaneo, tienes múltiples: TCP y UDP siendo las más comunes, pero también existen otras, como la "XMAS"

- *ettercap:* programa usado para simular el ataque *man-in-the-middle,* este software es el único usado que tiene interfaz gráfica y no depende exclusivamente de una consola, lo cual ya le da puntos a la hora de ser usado, pero también, al igual que *nmap,* ofrece una gran cantidad de formas diferentes de realizar el ataque, aunque en este proyecto sólo se use el *ARP poisoning.*

- *tcpdump* y *snort:* ambos ofrecen una gran flexibilidad a la hora de hacer capturas, sobre todo *snort* si incluimos las reglas añadidas por la comunidad. Ambas opciones son relativamente fáciles de usar, aunque hay que resaltar que se necesita tener al menos algo de conocimiento sobre la red en la que se está trabajando y lo que ocurre en ella, si no se quiere llevar algún susto innecesario con falsos positivos.

Adicionalmente se ha usado C como lenguaje de programación, dada su gran versatilidad y flexibilidad ante la mayoría de los problemas. Además, ya existen gran cantidad de librerías para ayudar con los pequeños detalles. Otro lenguaje que se tuvo en consideración fue Python, pero finalmente fue descartado ante la realización de que no se tenían los conocimientos necesarios para usarlo efectivamente y que sería poco más que una carga.

Como punto final cabe decir que se han usado comandos como *grep* y *arp* para conseguir información más precisa de la red y de que exactamente se espera encontrar para determinar si se está sufriendo un ataque.

4.3. Diseño detallado.

Módulo de soporte de máquinas virtuales: El entorno virtual en su lugar estará formado por cuatro máquinas virtuales, una que actuará de atacante, utilizando un sistema operativo Kali Linux, mientras que otras dos actuarán como máquinas atacadas, a la que les corresponden sistemas operativo Ubuntu Server y Ubuntu Desktop respectivamente. Las dos máquinas se han creado siguiendo la configuración predeterminada. Ha de decirse que, en el caso de Kali Linux, se tomó la decisión de instalar lo que en la fase de configuración denominan como "herramientas opcionales".

Esto se debe a que una de las ideas originales era conseguir simular ataques con un cierto grado de realismo, lo cual conlleva una variabilidad que sería más fácil de conseguir teniendo más herramientas posibles. ¿Por qué no se decidió ir instalándolas sobre la marcha? La experiencia y discusiones vistas en foros y secciones de *troubleshooting* desaconsejan el uso de herramientas como *apt-get*, bajo riesgo de que no puedas volver a abrir la máquina virtual.

La última máquina virtual será sobre la que se haga el IDS, de esta forma el IDS puede estar en funcionamiento de forma paralela a las otras, sin quitarles rendimiento. Se ha basado esta máquina en un sistema operativo Ubuntu Server, sin ninguna de las funciones que los servidores suelen tener instaladas, para reducir su carga lo máximo posible y de esta forma poder ser introducida en la red usando hardware barato como una *Raspberry.*

Otro aspecto que puede ser considerado como relevante es el hecho de que se les ha dado 4 gigabytes de memoria RAM y dos núcleos de CPU a las máquinas virtuales Kali y Ubuntu Server y 2 gigabytes y un único núcleo a Ubuntu User. Esto se debe principalmente al coste adicional que tienen los entornos gráficos empleados. La máquina IDS por el otro lado al no tener interfaz gráfica se le ha dado solo 512 megabytes, esto es para emular lo que podría ser una *Raspberry.* Para ver cómo queda la arquitectura del módulo véase la figura 1 en el punto 4.

Módulo de denegación de servicio (DOS): Desde este bloque se espera conseguir detectar y avisar exitosamente al usuario de que está siendo atacado por un ataque de denegación de servicio, ya sea por *PING flood*, o por *MAC flood*, por ejemplo.

Para este fin, se ha creado inicialmente un programa en C llamado *ping_flood.c*, que se dedica a leer los *logs* de *tcpdump*, además de un script llamado *rundos.sh*. Este *script* es ejecutado inicialmente para recoger datos del usuario, que serán entonces usados en el programa. La tarea más importante del *script* es ejecutar *tcpdump*, que generará un archivo de salida a partir del comando:

```
sudo tcpdump -Qin -n --immediate-mode -l -ttttt
```

Este comando se lanza en *pipeline* en el propio programa si se decide ejecutar directamente desde ahí, sin pasar por el *script*, que es la acción recomendada. En el *script* se ejecuta el siguiente comando:

sudo timeout --preserve-status “${_tout}s” tcpdump -Qin -n -l -ttttt -i “$_ifce” > “$_pf_capture”

Las variables *_tout, _ifce* y *_pf_capture* se usan para almacenar los valores del timeout, interfaz y ruta destino de la salida de *tcpdump.* Estos son los valores que el usuario va a poder introducir manualmente en la interfaz creada en el terminal de la aplicación. Es después de la ejecución de *tcpdump* que se ejecuta el programa *ping_flood,* pasándole como argumentos la interfaz y archivo de salida. Estos campos si no son introducidos por el usuario recibirán un valor por defecto y serán completamente necesarios si el usuario decide ejecutar el programa directamente, sin pasar por el *script*. Si se han rellenado todos los campos, el comando que se usa para lanzar *ping_flood* es:

ping_flood -i $_ifce -ip $_ip -f $_pf_capture

En esta orden las variables se usan para lo mismo que en el caso anterior, con la opción añadida de escuchar en una dirección concreta; si así se desea.

El script también pasará a ejecutar el programa “mac_flood”, para detectar ataques de denegación de servicio por paquetes ARP. Esto lo hace a su vez después de ejecutar *tcpdump*, ya que este también recoge paquetes ARP recibidos. El programa se ejecutará desde el *script* con la siguiente orden:

mac_flood -c 200 -f $_af_capture

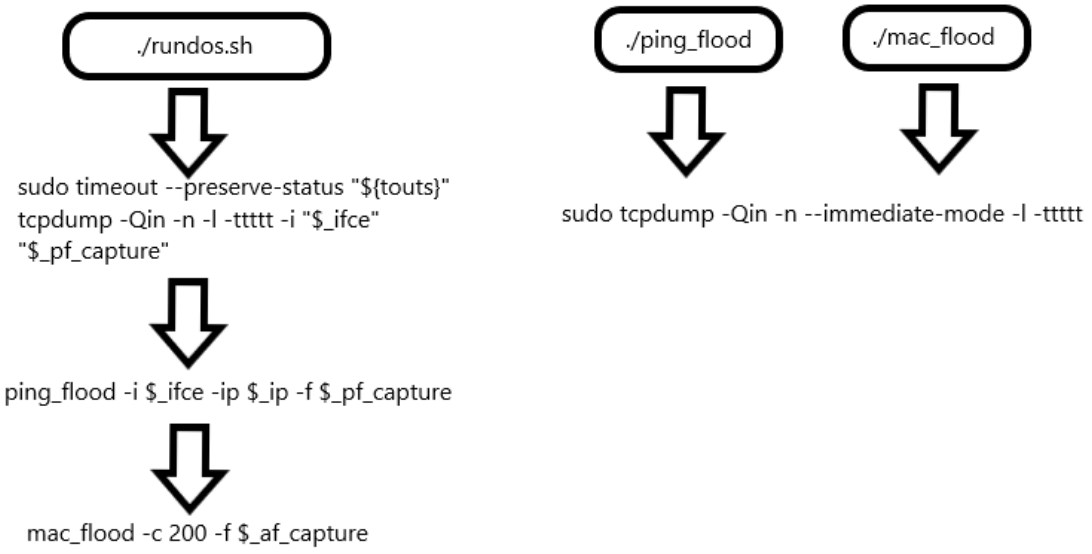

Figura 2: Arquitectura del módulo de denegación de servicio

Módulo de detección de escaneo de puertos: En este bloque se espera detectar si se está siendo víctima de un escaneo de puertos efectuado por el programa NMAP. Para conseguir esto, se han creado múltiples módulos en el programa "nmap_counter.c"; que se usarán para contrarrestar los tipos de escaneos más comunes, véase un ataque XMAS, escaneo UDP o TCP. Para este fin se ha decidido usar el software

snort, ya que es gratis, bastante efectivo y tiene una dedicada comunidad que progresivamente ha ido mejorando y actualizando el programa.

Se lanzará snort desde el script “runportscan.sh”, que tiene una funcionalidad casi idéntica a “rundos.sh” con la orden:

sudo snort -A console -q -c /etc/snort/snort.conf -i $_iface -K ascii > $_sf_capture

En la orden *$_iface* y *$_sf_capture* son para rellenar los campos de interfaz y de ubicación que tendrá el archivo de salida. Estos campos se autorrellenarán con valores por defecto automáticamente si el usuario los deja en blanco. De esta forma nos aseguramos de que el resultado será legible y pueda ser utilizado a continuación por el ya mencionado programa *nmap_counter*. Este programa se ejecutará a continuación con el siguiente comando:

nmap_counter -f $_sf_capture

La alternativa, al igual que en el caso anterior, es ejecutar directamente el programa sin pasar por la interfaz, mediante un *pipeline.* Esto en consecuencia ejecutará el siguiente comando, donde “argv[i + 1]” es la interfaz sobre la que se lanzará el comando:

sudo snort -A console -q -c /etc/snort/snort.conf -i “argv[i + 1]” -K ascii

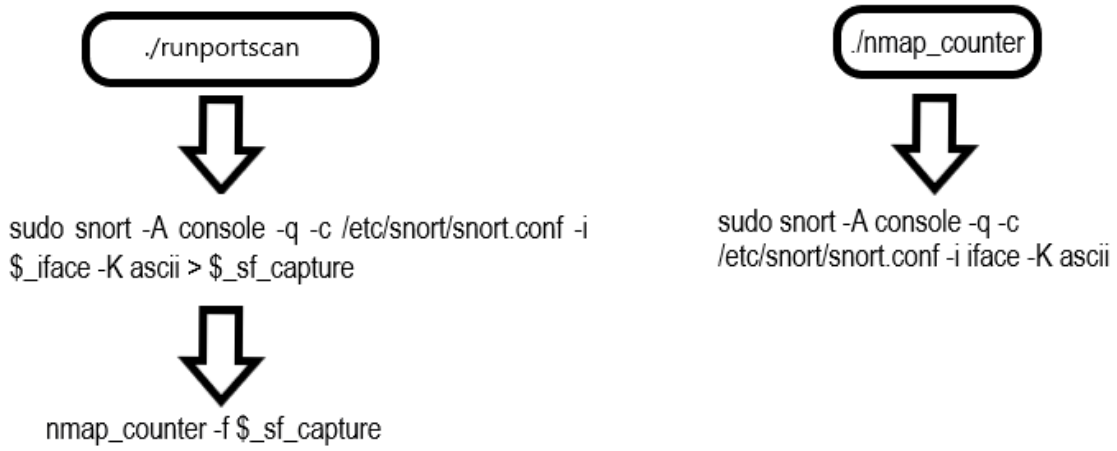

Figura 3: Arquitectura de comandos del módulo de escaneo de puertos

Módulo de detección de intrusiones *man-in-the-middle* (MITM): A continuación, se espera detectar si el equipo en cuestión está sufriendo un ataque MITM, esto se espera conseguir escaneando las tablas ARP de la máquina, para ver si alguna dirección MAC no corresponde con la que debería. La forma en la que funciona una intrusión como la planteada ahora es que el atacante consigue ponerse entre dos máquinas en una red (generalmente el servidor y el *gateway*) para poder escuchar toda clase de tráfico que se mueva hacia y desde el servidor.

Para este módulo se ha decidido que no será necesario hacer un programa, y que bastará con hacer un solo script que funcione como ampliación a las funcionalidades establecidas en los otros dos. Con esta mentalidad, se necesitará ejecutar el siguiente comando para averiguar las tablas ARP:

```
ssh "$_RUSER"@"$_SERVER" arp -an -D "$_exp_ip" |cut -d" " -f4
```

Donde *$_RUSER* y *$_SERVER* son el nombre de usuario y dirección IP de la máquina que se va a comprobar y *$_exp_ip* es la IP del *Gateway* del sistema.

Como se puede ver es un comando SSH, eso es para que se ejecute el arp -an en la máquina destino deseada. A continuación, bastará con leer un archivo en el que se encuentren las direcciones MAC de las máquinas de la red, para poder compararlas con las actuales y ver si hay alguna que no coincida.

5. Desarrollo de la solución propuesta.

Desde el primer momento, se tenían absolutamente claros algunos de los aspectos que se iban a tratar en este proyecto, con esto se quiere poner de ejemplo al hecho de que iba a tratar de ciberseguridad, y sobre cómo aumentar la seguridad de un sistema. Otros conceptos como los tipos de ataques que se iban a contemplar por ejemplo no estaban tan claros y requirieron unas recomendaciones por parte del tutor. Por desgracia hubo algunas otras cosas que fueron dudas serias y requirieron un periodo más prolongado para llegar a una conclusión satisfactoria, el mejor ejemplo de esto es el lenguaje que se iba a utilizar para programar.

Comenzando por el principio, la idea del proyecto se cementó de una conversación entre el alumno y el tutor poco después de aceptar el último la petición del primero de ser su el tutor para esta propuesta. Dicha idea consistía en desarrollar e intentar implementar un sistema de detección de intrusiones. Esto fue ampliándose con el paso del tiempo y a medida que iban apareciendo preguntas del estilo de cuál iba a ser la magnitud que iba a tener el proyecto, hasta acabar en la idea actual de que iba a ser una propuesta pensada para otros usuarios que puedan no tener mucho conocimiento de ciberseguridad o de cómo protegerse.

Una vez tomada la decisión de que se iba a intentar implementar un pequeño IDS, fue necesario decidir qué hardware iba a ser usado. Ante esta cuestión existían dos posibilidades: usar múltiples máquinas *raspberry* proporcionadas por el tutor, o simular una red con máquinas virtuales. Se decidió la segunda opción, dado el hecho de que el alumno ya contaba con conocimientos previos aprendidos a lo largo de la carrera para ayudarse en esto y porque el ordenador en el que se iba a realizar toda esta simulación tenía la potencia más que suficiente para funcionar bajo cualquier estrés que las máquinas pudieran provocar. Adicionalmente no se tenían máquinas *Raspberry* a disposición.

5.1. Software a usar.

Llegado este punto, la siguiente idea consistía claramente en decidir qué sistemas operativos se iban a emplear. Uno de ellos estuvo bien claro desde el principio: *Kali Linux.* Esto se debe a que era un sistema operativo con el que ya se había trabajado recientemente y con el que se tenía un cierto grado de familiaridad, a parte del hecho de que es posiblemente una de las mejores herramientas de *pentesting* gratuito que existen y que tanto el tutor como otras personas recomendaron. Cabe destacar que se escogió la opción completa de instalación para *Kali,* lo cual conllevó más espacio de disco duro ocupado por la máquina y el hecho de que no hiciera falta descargar ninguna herramienta adicional.

Los otros sistemas operativos con los que se decidió trabajar son *Ubuntu Server* y *Ubuntu Desktop. Ubuntu* es algo con lo que se lleva trabajando toda la carrera, o si no, con alguna otra distribución de *Linux*. Se consideró que al ser las bases las mismas, fuera de la parte de configuración del servidor, en la que se dejaron mayoritariamente todos los valores por defecto para la máquina de servidor; esta opción no supondría ninguna clase de dificultad de emplear. Más aún en la máquina IDS, que es un *Ubuntu Server* sin ninguna clase de función de servidor; en la que solo se usan las funcionalidades más básicas. El objetivo aquí es que el IDS se pueda implementar en una máquina *Raspberry.*

El siguiente paso por emprender suponía diseñar la red en la que el entorno virtual existiría. Originalmente se usó un adaptador puente, debido a su inmensa sencillez y al hecho de que no había que modificar nada de nada para que todo funcionara. Esto cambió rápidamente al caer en cuenta de que empleando esta estrategia se estaba integrando completamente a las dos máquinas en la red local del hogar, y esto podía causar problemas a la hora de hacer algunas de las pruebas. Esto fue corregido cambiando la configuración de red a una red *NAT.* Una vez se habían acabado todas las preparaciones previas aquí mencionadas se dio comienzo al desarrollo en sí del IDS. Se puede ver un diagrama de la red interna en el punto 4.1.

5.2. Denegación de servicio.

El primer elemento que se decidió desarrollar fue un programa capaz de detectar y avisar al usuario de que se estaba sufriendo un ataque de denegación de servicio. Para esto primeramente se decidió que una de las mejores herramientas gratuitas disponibles que se adaptaran a las condiciones requeridas, sin salirse demasiado de los límites establecidos era *tcpdump.* Fue ahora cuando uno de los mayores obstáculos se hizo presentes: Qué lenguaje iba a ser empleado para programar.

Fue en este momento cuando se hizo un análisis de los lenguajes que se habían estudiado a lo largo de la carrera y si serían adecuados para el tipo de tarea que se quería desarrollar. El primero en el que se pensó fue Java, pero esta idea se vio descartada rápidamente al llegar a la conclusión de que se desconocía lo útil que sería utilizarlo para las facetas más orientadas a redes del proyecto. La siguiente posibilidad era C, que resultó ser una opción mucho más preferible, ya que era un lenguaje que había sido usado más recientemente por el alumno, además de que se sabía que existían todas las facultades necesarias para desarrollar adecuadamente el proyecto. Se ha de destacar que también se analizó la posibilidad de usar Python a recomendación de terceros, pero se descartó la posibilidad principalmente porque era un lenguaje completamente desconocido para el alumno.

Con todo esto, se hizo un boceto inicial bastante simple de cómo iba a ser el programa y que iba a hacer y se comenzó a trabajar. Esto claramente fue aumentando de proporción cuando se consideró la posibilidad de usar scripts para lanzar los programas. En su versión final el funcionamiento de todo el módulo, incluyendo script es el siguiente:

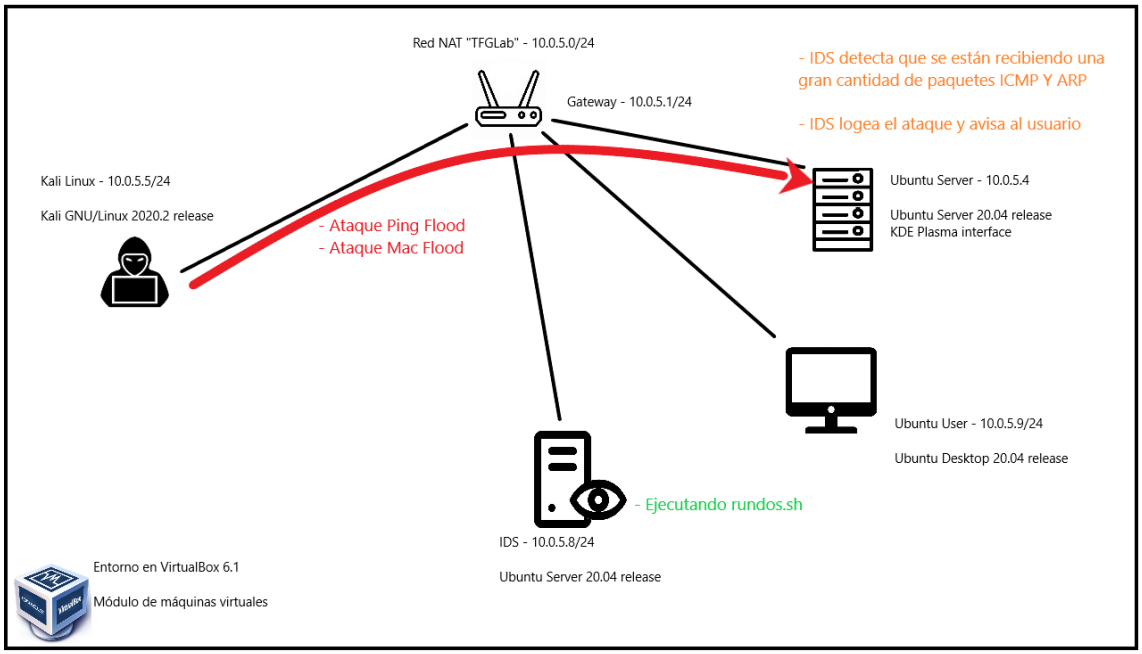

Figura 5: Estructura de la red cuando se produce un ataque DOS

Inicialmente sólo se pensó en hacer un programa *ping_flood.c* que sería ejecutado sin argumentos por un script. Este programa se limitaría a obtener la IP de la máquina local y usarla para compararla con las IP que aparecerían en los resultados de las capturas realizadas con *tcpdump*. Esto lo haría porque sólo interesan los paquetes *ICMP* que vayan a ir dirigidos a la máquina local. A continuación, se almacenaría la IP de la máquina que ha enviado dicho paquete *ICMP* en un *array* y se sumaría 1 a un contador que será correspondiente a dicha *IP,* si se cuentan unos 200 *pings* en menos de un segundo por parte de la misma IP se considera que se está siendo atacado.

Esta solución con el tiempo pasó a ser insatisfactoria, de forma que se decidieron añadir más elementos al programa y modificar algunos de los elementos hechos hasta el momento. Las modificaciones ocurrieron en un intervalo de aproximadamente dos semanas entre actualizaciones. Estos elementos añadidos consistían en:

1. Múltiples opciones que tendría el programa como seleccionar la interfaz que va a ser elegida, la *IP* que va a ser monitoreada, por si no se desea ver solo los paquetes que le llegan a la *IP* local. Estas opciones se las pasará el *script* al programa.
2. La dirección en la que se ubicarán los *logs* de *tcpdump.*
3. Usar una estructura personalizada para guardar las direcciones *IP* en lugar de un *array*.
4. Añadir la posibilidad de que la orden *tcpdump* necesaria para la ejecución del programa se lance mediante *pipelining,* de forma que en vez de tener que leer los *logs* de *tcpdump* se lee a tiempo real la salida del programa.

De forma paralela a *ping_flood* se estaba desarrollando otro programa que tendría una función similar, pero para otro tipo de ataque de denegación de servicio: *mac_flood.* Este programa tiene la función de detectar si un atacante está intentando ejecutar un ataque de denegación de servicio de inundación de tablas ARP.

La otra gran idea que se tuvo para ampliar esta parte del proyecto es darle funcionalidades reales al script que se iba a usar, más allá de que simplemente ejecute un par de comandos y ya está. La lista de funcionalidades que tiene el *script* es la siguiente:

1. La opción de recibir múltiples opciones para la interfaz que se va a observar, el archivo de registro de la salida, si se quiere que la interfaz en el terminal tenga colores o no, la *IP* que se va a observar y los archivos de captura para *ping_flood.c* y *arp_flood.c*.
2. Introducir valores predeterminados para todos los campos anteriormente mencionados en caso de que no se rellenen.
3. Comprobar si el sistema tiene instalados todos los paquetes necesarios para que funcione bien el programa.
4. Asegurarse de que *ping_flood* y *mac_flood* están en el *PATH* del sistema, de lo contrario el programa no funcionará.
5. Una pequeña interfaz usando el terminal del sistema para poder hacer modificaciones a los valores del programa mencionados previamente.

Para más detalles sobre cómo se ha logrado todo esto por favor véase el anexo 1.

5.3. Escaneo de puertos.

Para esta parte, el procedimiento fue bastante similar al del módulo de denegación de servicio. Esto quiere decir que las ideas iniciales de cómo debería de ser el programa y sus funcionalidades iban a ser bastante básicas, y con el tiempo se decidió añadir múltiples opciones de calidad de vida como la interfaz básica en el terminal o las múltiples opciones de lanzamiento que se pueden elegir.

Todo este módulo consiste esencialmente en leer los *logs* que resultan de lanzar el famoso *snort* y quedarse sólo con las partes que podrían estar relacionadas con un ataque de escaneo de puertos, habitualmente ejecutado con *nmap.* Se hizo, al igual que los otros; usando el lenguaje C, por las mismas razones descritas anteriormente, y de la misma forma se acabó haciendo un *script* que serviría para ejecutar el programa, que acabaría llamándose *nmap_counter.c*. Todo esto se amplió haciendo un nuevo y mejorado *script* que seguiría la misma línea que el anterior, con una pequeña interfaz gráfica y todo para acompañar.

Antes incluso de ponerse a programar esta idea, se tenía pensado usar otra herramienta llamada *psad.* Esta herramienta estaba especializada en detección de ataques de escaneo de puertos, más concretamente *nmap.* La idea fue descartada finalmente, ya que este era un software del cual el alumno no había oído hablar jamás y las últimas personas que en Internet hablaban de *psad* lo hicieron hace como mínimo 4 años.

En su versión más inicial el programa se lanzaría tras ejecutar *snort,* leyendo los logs de un archivo de salida en una localización predeterminada del sistema y analizando los resultados en busca de los códigos de mensajes de aviso, por ejemplo, *snort* le asigna un código determinado a una alerta si se está produciendo un ataque

XMAS con *nmap,* uno de los ataques más agresivos que puedes realizar con dicha herramienta; la idea era simplemente leer este código y avisar al usuario de que posiblemente alguien estaba intentando analizar los puertos de su máquina.

Esto se acabó expandiendo de forma similar al anteriormente descrito módulo de detección de ataques de denegación de servicio, implementando una interfaz bastante similar a la anterior que garantizaba al usuario la capacidad de alterar algunos campos a su elección, como el destino en el que se escribirá la salida del comando *snort* (que es obligatorio de introducir) o el tiempo que va a estar dicho comando ejecutándose.

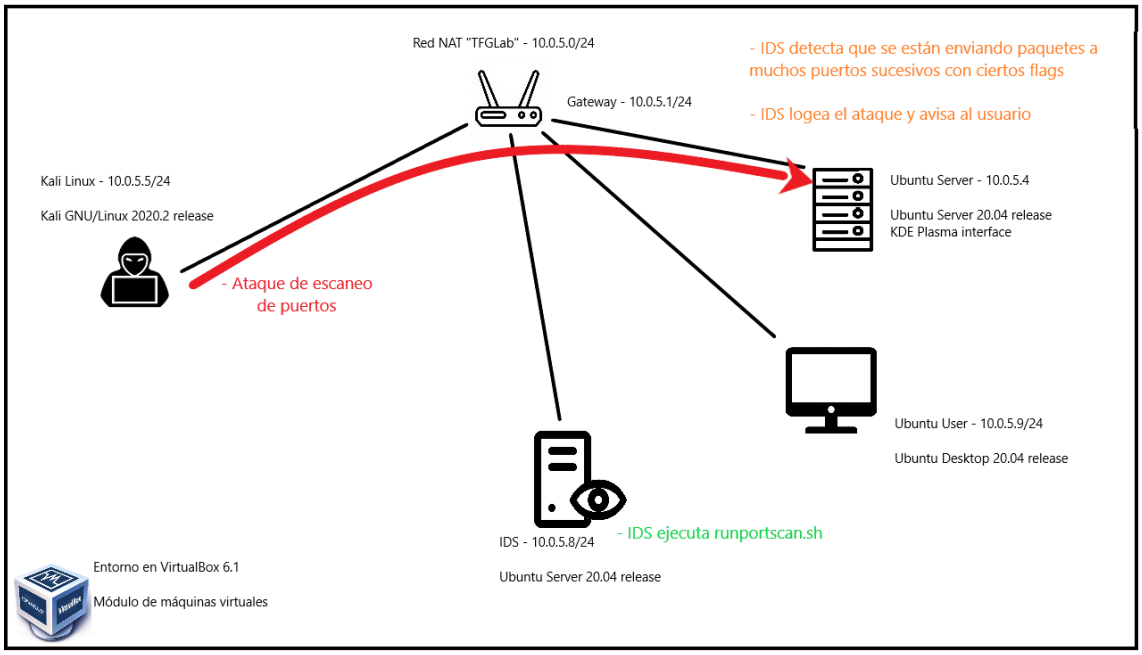

Figura 6: Estructura de un ataque de escaneo de puertos sobre la red simulada

5.4. *Man-in-the-middle*.

La traducción del término inglés *man-in-the-middle* (MITM) es literalmente “hombre en medio”, lo cual sirve para hacerse una idea relativamente clara de en qué puede consistir este tipo de ataque. De lo contrario, en el punto 2.3 del proyecto se explica con más detalle.

Para poder simular un ataque como este, se ha empleado una programa que ya venía por defecto en la instalación completa de *Kali*: *Ettercap. Ettercap* es un programa que te ayuda a hacer ataques de este estilo, entre otros; de muchas formas diferentes, la preferida y más usada en este proyecto ha sido por envenenamiento ARP. Esto lo que implica es que el programa manda paquetes ARP maliciosos a los dos extremos de la conexión, siendo el servidor uno y el *router* otro; para hacerles creer que se ha modificado la dirección MAC del otro extremo. En verdad, lo que ha ocurrido es que la máquina del atacante se ha introducido entre las otras dos máquinas, y está ahora escuchando todo el tráfico que se mandan entre ellas.

No es necesario debatir demasiado sobre el porqué esto es potencialmente muy peligroso, ya que el atacante puede ahora tranquilamente guardarse toda clase de información sobre clientes o empleados de la empresa usando este método.

Una vez se ha ejecutado el ataque con éxito, es sorprendentemente fácil darse cuenta de que se está siendo atacado, pero para eso necesitas tener un conocimiento avanzado de la red y sus componentes, además de que hay que ir a buscar este tipo de brecha en concreto.

Si se miran las tablas ARP de la máquina en cuestión, se puede comparar cual es la MAC del *Gateway,* a continuación, se tiene que comparar dicha MAC con la correspondiente MAC real del *router*. Si coinciden, enhorabuena, todo está yendo bien en la red de la empresa. En caso de que no lo hagan, hay un atacante en la red que ha estado escuchando todo el tráfico que se movía desde y hacia el router por una cantidad de tiempo desconocida.

La forma que se ha elegido para contrarrestar esto, es mediante un script; ya que la solución planteada consiste en que el usuario guarde en un archivo una lista de direcciones MAC que sean correspondientes a la red y, en caso de ver que falta alguna de dichas direcciones y que en su lugar hay una nueva dirección previamente desconocida, avisar al usuario.

Algo así es relativamente fácil de hacer en *bash,* así que se decidió que se emplearía esto para la última parte del proyecto. Como punto extra, también apoya la idea del proyecto de que cualquier persona contribuir a aumentar la seguridad de su sistema. La experiencia en este aspecto no es excesivamente relevante, ya que se ha conseguido crear un sistema que usa herramientas de fácil uso para usuarios inexpertos. Para concluir, cabe destacar que se sigue con la idea de insertar una interfaz gráfica en el terminal del sistema para ayudar al usuario a interactuar con la aplicación.

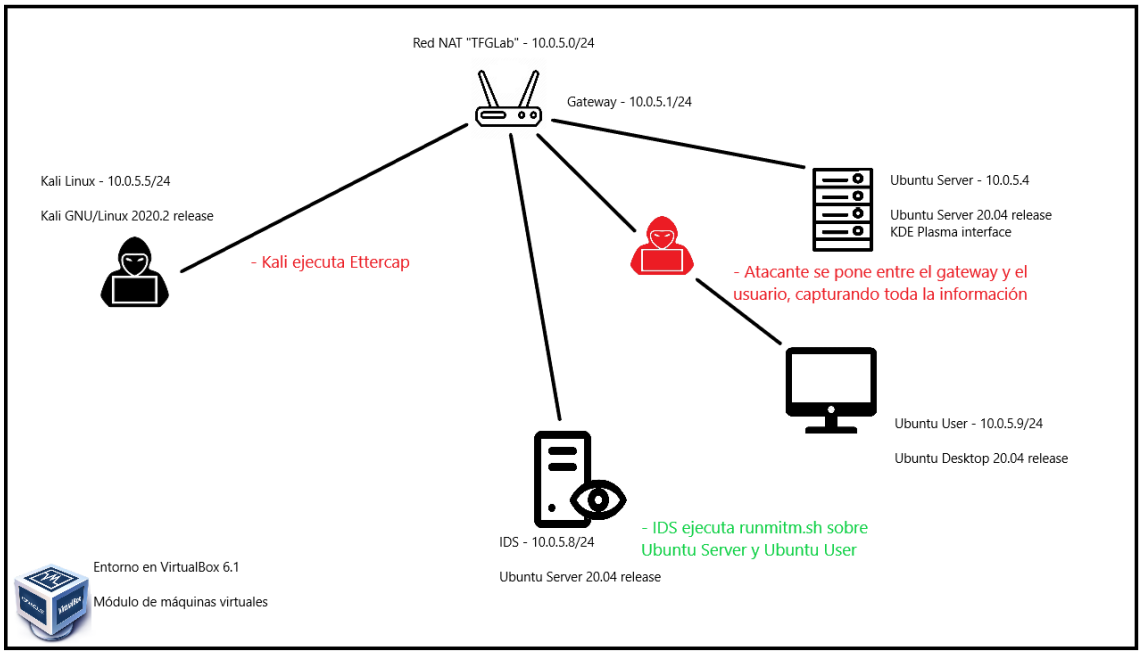

Figura 7: Estructura de un ataque MITM sobre la red simulada

6. Implantación

Una vez preparadas y estudiadas las bases de lo que se acabaría convirtiendo el proyecto, tocaba empezar a trabajar y a desarrollar dicho proyecto. El primer paso para conseguir eso fue instalar VirtualBox, software gratuito de creación y manejo de máquinas virtuales, muy empleado durante toda la carrera.

Acto seguido, hubo que crear las máquinas virtuales e instalar los sistemas operativos correspondientes, siguiendo los procedimientos habituales; con algunas particularidades, como que se instaló la versión más completa posible de Kali, o que se instaló una interfaz gráfica para Ubuntu Server, con el fin de ayudar en la programación. Las máquinas del IDS y del Ubuntu Desktop se instalaron en sus versiones por defecto, la máquina del IDS no tiene siquiera ninguna de las funcionalidades que suelen estar presentes en los *Ubuntu Server.* Para más detalles véase el capítulo *4.1 Arquitectura del sistema*.

Con esto hecho, no hizo falta añadir nada a la máquina *Kali,* pero en el caso del *Ubuntu Server* de la máquina IDS, donde se va a desarrollar y ejecutar la gran parte del proyecto, la cosa cambia ligeramente. En caso de que falte algo crítico para la ejecución de los programas, se le avisará al usuario. De todos modos, hay que asegurarse de que el equipo tiene instalados los siguientes elementos:

1. Un compilador de C, por ejemplo: *gcc.*
2. Elementos que generalmente vienen instalados por defecto en la mayoría de las distribuciones de *Linux: coreutils, dialog, ncurses-bin, útil-linux, arp, grep.*
3. *Tcpdump* y *snort,* para capturar paquetes en la red.
4. *Xterm,* para las interfaces en terminal.

Una vez se cumplen todos los requisitos mencionados anteriormente, se obtienen los programas y *scripts* desarrollados, y es necesario compilar los programas, de forma que los archivos de salida del compilador deben llamarse exactamente igual que los programas, sin la extensión al final. Para el caso de *ping_flood.c,* el archivo de salida se llamará *ping_flood.*

Adicionalmente, también es completamente necesario que la ruta en la que se encuentren estos archivos compilados esté en el *PATH* del sistema. Esto es preciso porque los scripts van a buscar los archivos al *PATH,* garantizando de esta forma, que los scripts y los programas no necesiten estar en el mismo fichero para poder ser ejecutados. Como último detalle, habrá que asegurarse de que los *scripts* y los programas tengan derechos de lectura, y ejecución. Ambas cosas se explican a continuación:

Para añadir una ruta al *PATH* tienes 2 opciones

- Una primera opción que es temporal, al reiniciar el ordenador habrá que hacerlo de nuevo: export PATH=”$PATH:$HOME/*la ruta deseada*”
- Una opción permanente:
    - Editar el archivo ~/.profile

- Introducir esto al final: PATH="$PATH:$HOME/*la ruta deseada*"

Una vez se ha hecho todo esto, solo queda ejecutar los *scripts.* Al hacerlo, lo primero que se ve es una pequeña lista con los requisitos para poder ejecutarlos. Esta lista no será visible si se cumplen todos los requisitos, de forma que, para demostrarlo, se eliminará a propósito uno de los archivos imprescindibles para forzar el error:

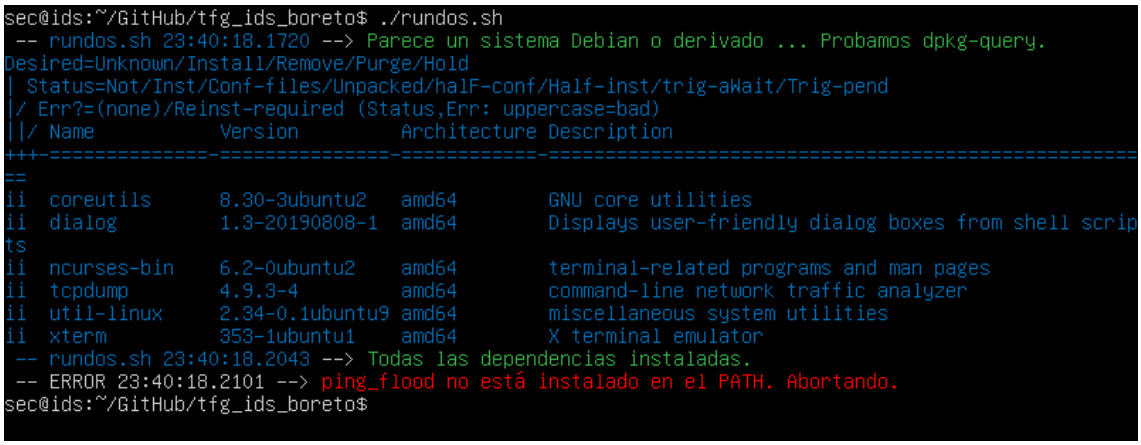

Figura 8: comprobación de dependencias en rundos.sh

En caso de que todo haya ido bien, se habrá pasado a ver la interfaz del programa, ahora sólo queda seguir las instrucciones. Para más dudas véase el siguiente punto, para las pruebas.

7. Pruebas

Para hacer las pruebas necesarias en *VirtualBox* que verificarán si el proyecto y sus partes desarrolladas han tenido éxito, será necesario ejecutar las cuatro máquinas virtuales instaladas al mismo tiempo. Esto en pocas palabras quiere decir que el ordenador en el que se hagan las pruebas debe tener unos componentes decentes capaces de soportar esta carga, de lo contrario las máquinas presentarán serios problemas en la fluidez de su ejecución. Es incluso posible que se "cuelguen" las máquinas, lo que quiere decir que dejan de responder y se tienen que reiniciar.

Una vez dicho esto, se dividirá en partes este apartado para facilitar su navegación. Se comenzará por el módulo de denegación de servicio, seguido del módulo de escaneo de puertos y finalmente, el módulo *man-in-the-middle.*

Antes de empezar hay que subrayar una función que tienen los *scripts* y es la de detectar si falta algún elemento crítico para la ejecución de estos. Por ejemplo, en el caso de que falte añadir la ruta donde se encuentran los programas al *PATH* saltará el mensaje visto al final del punto 6.

7.1. Denegación de servicio.

Una vez estén las cuatro máquinas en marcha, es todo cuestión de lanzar el *script rundos.sh* para empezar a escuchar ataques.

Con motivo de hacer estas pruebas, se rellenan los campos de la interfaz de forma que los archivos en los que se guardarán los *logs* del programa van a estar en el escritorio. Esto se ha hecho simplemente por facilidad para encontrar dichos archivos y mostrarlos en pantalla. Habitualmente se querría tener estos *logs* en una carpeta temporal, de hecho, los valores que se asignan por defecto en caso de que se dejen vacíos los campos para las direcciones están en */tmp.*

Al mismo tiempo, hay que preparar un terminal en *Kali* para lanzar los comandos que realizarán los ataques de denegación de servicio. Estos comandos van a ser *hping3* y *macof.* Primero se hará una prueba lanzando solo *hping3*. Cabe destacar que se recomienda que no se pongan tiempos muy altos, ya que los programas *ping_flood* y *mac_flood* no se van a ejecutar hasta que no acabe de ejecutarse *tcpdump,* y puede estar ocurriendo un ataque mientras se espera a que acabe *tcpdump* y no se sabrá.

Figura 9: comandos para el ataque de denegación de servicio

Hay varias opciones que pueden ser modificadas en el *script* para adaptarlo al gusto del usuario, pero hay ciertos campos que no se pueden dejar en blanco o empezarán a surgir problemas. Se ven a continuación los scripts con y sin valores en los campos no relevantes.

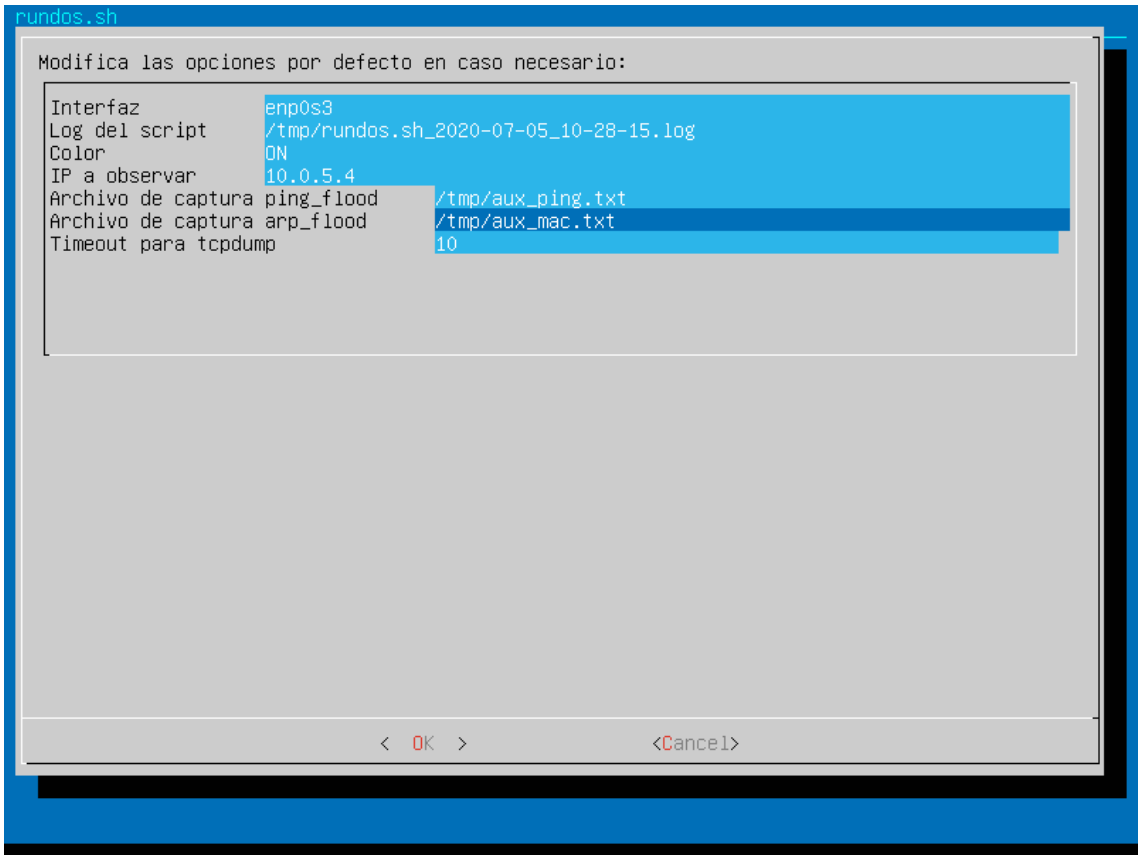

Figura 10: rundos.sh con todos los valores introducidos para una monitorización de la red

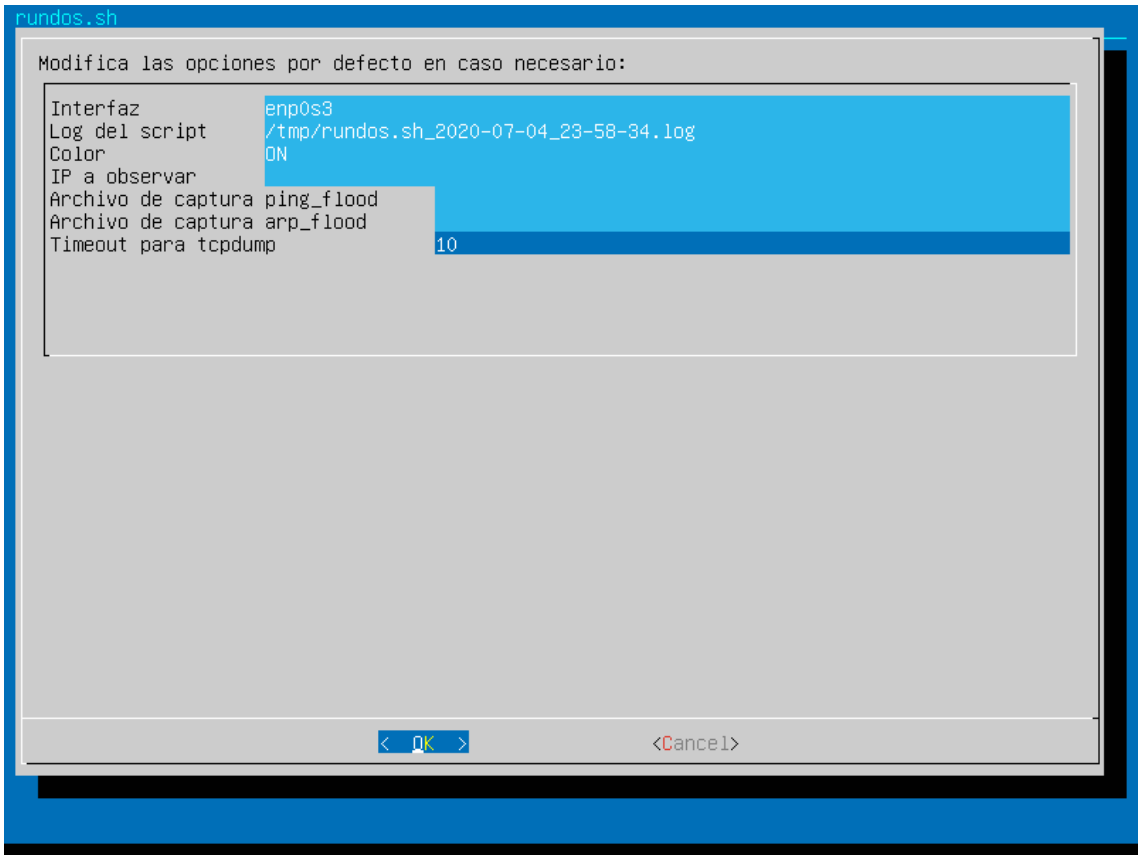

Figura 11: rundos.sh con solo los valores necesarios

```
 -- rundos.sh 10:40:32.7403 --> Comenzamos ...
 -- rundos.sh 10:40:32.0128 --> Ejecutando tcpdump sobre enp0s3 durante 10 segundos
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on enp0s3, link-type EN10MB (Ethernet), capture size 262144 bytes
25313 packets captured
388711 packets received by filter
350480 packets dropped by kernel
 -- rundos.sh 10:40:44.2762 --> Ejecutando ping_flood sobre /tmp/aux_ping.txt
DIRECCION IP OBSERVADA: 10.0.5.4
AVISO:  Posible ataque desde 10.0.5.5 --> 200 pings en menos de 1 segundo
Se han encontrado un total de 24467 pings de la IP 10.0.5.5
 -- rundos.sh 10:40:46.0322 --> Ejecutando nuevamente tcpdump para obtener /tmp/aux_mac.txt
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on enp0s3, link-type EN10MB (Ethernet), capture size 262144 bytes
28157 packets captured
396654 packets received by filter
354975 packets dropped by kernel
 -- rundos.sh 10:40:57.6259 --> Ejecutando mac_flood sobre /tmp/aux_mac.txt
AVISO:  [0:0.14977] Posible ataque mac_flood --> 200 pings en menos de 1 segundo
Se han encontrado un total de 8035 paquetes
sec@ids:~/GitHub/tfg_ids_boreto$
```

Figura 12: la salida de la ejecución del script rundos.sh

Una última forma de ejecutar los programas es directamente, sin pasar por la interfaz. Los programas funcionan a la perfección igualmente porque lanzan *tcpdump* en un *pipe.* Todos los programas tienen la opción de “--help”, que indica por pantalla como exactamente se deben de ejecutar. Esta opción es la que se debería de usar si quieres rápidamente detectar si está ocurriendo un ataque, ya que te dice en tiempo real si se está produciendo o no.

```
sec@ids:~/GitHub/tfg_ids_boreto$ ping_flood --help
Opciones:
     -i interfaz:       Indica la interfaz que queremos observar (enp0s3 por defecto).
     -f archivo:        Archivo del que obtener los datos (capture.txt por defecto).
     -ip IP:            Indica la dirección IP a monitorear (si no queremos que sea la
                        de la máquina. En este caso -i no es de ninguna utilidad)
     --help:            Show this help
sec@ids:~/GitHub/tfg_ids_boreto$
sec@ids:~/GitHub/tfg_ids_boreto$
sec@ids:~/GitHub/tfg_ids_boreto$
sec@ids:~/GitHub/tfg_ids_boreto$ mac_flood --help
        Parámetros:
        -c N:           Fija el valor máximo del contador (por defecto 200).
        -f file:        Fija el archivo de entrada a 'file' (por defecto se ejecuta en tubería
                        lanzando un comando tcpdump).
sec@ids:~/GitHub/tfg_ids_boreto$
```

Figura 13: opciones de ayuda de los programas para ejecución directa

Ahora se ven los resultados de ejecutar los módulos individualmente en caso de ataque.

```
sec@ids:~/GitHub/tfg_ids_boreto$ ./ping_flood -f pipe -ip 10.0.5.4
DIRECCION IP OBSERVADA: 10.0.5.4
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on enp0s3, link-type EN10MB (Ethernet), capture size 262144 bytes
AVISO:  Posible ataque desde 10.0.5.5 --> 200 pings en menos de 1 segundo

^C
sec@ids:~/GitHub/tfg_ids_boreto$ 16329 packets captured
179904 packets received by filter
163241 packets dropped by kernel

sec@ids:~/GitHub/tfg_ids_boreto$ ./mac_flood
No se ha fijado archivo de entrada.
No se ha fijado Nº máximo de paquetes, usando 200
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on enp0s3, link-type EN10MB (Ethernet), capture size 262144 bytes
AVISO:  [0:0.18724] Posible ataque mac_flood --> 200 pings en menos de 1 segundo
^C
sec@ids:~/GitHub/tfg_ids_boreto$ 15831 packets captured
20814 packets received by filter
4983 packets dropped by kernel

sec@ids:~/GitHub/tfg_ids_boreto$ _
```

Figura 14: ejecución de ping_flood y mac_flood en modo pipeline

Como cabía de esperar no se han tenido problemas al ejecutar los dos comandos concurrentemente. Con esto se da por finalizado con éxito el módulo de denegación de servicio.

7.2. Escaneo de puertos.

Los primeros pasos son los mismos que en el módulo anterior: iniciar las máquinas, preparar *Kali* con los comandos pertinentes y rellenar los campos deseados en la interfaz del script correspondiente, en este caso, *runportscan.sh.*

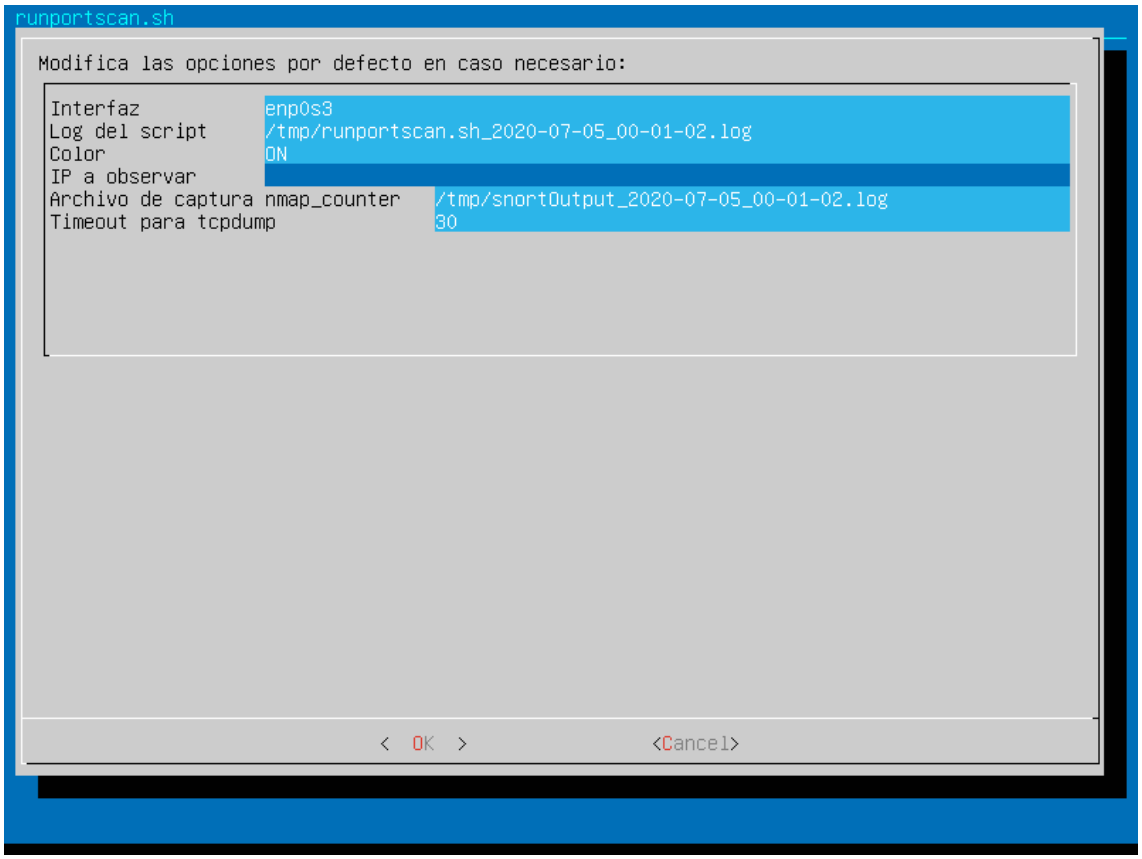

Figura 15: la interfaz visual de runportscan.sh

Se le sigue aplicando a este programa la misma norma que al caso anterior de no añadirle demasiado tiempo, solo que esta vez hay que tener en cuenta un aspecto extra.

Se ejecuta a continuación *nmap,* con la opción "-sX" para lanzar un ataque *XMAS* y se pone en marcha el módulo detector. Se ve claramente que ha funcionado y se ha detectado exitosamente el escaneo.

```
 -- runportscan.sh 16:55:03.1945 --> Comenzamos ...
 -- runportscan.sh 16:55:03.2019 --> Ejecutando snort sobre enp0s3 durante 30 segundos
 -- runportscan.sh 16:55:33.2510 --> Ejecutando nmap_counter sobre /tmp/snortOutput_2020-07-05_16-52
-00.log
Se esta recibiendo un escaneo XMAS de 10.0.5.5
 -- runportscan.sh 16:55:33.2693 --> Terminado correctamente
sec@ids:~/GitHub/tfg_ids_boreto$
```

Figura 16: detección de un escaneo XMAS por runportscan.sh

Al igual que en el módulo de denegación de servicio, es posible ejecutar directamente *nmap_counter* sin necesidad de pasar por la interfaz. Se ve cómo es posible a continuación:

```
sec@ids:~/GitHub/tfg_ids_boreto$ ./nmap_counter -f pipe -i enp0s3
Se esta recibiendo un escaneo XMAS de 10.0.5.5
^C
sec@ids:~/GitHub/tfg_ids_boreto$ *** Caught Int-Signal
```

Figura 17: resultado de lanzar el programa en modo pipeline

Ahora para finalizar, siguiendo con la idea del módulo anterior, se deciden lanzar varios ataques simultáneos para ver si el programa es capaz de soportarlo. Estos han sido ataques TCP, UDP, FIN y XMAS. Esta vez se ha decidido atacar a varios objetivos simultáneamente para ver si los detectaba todos.

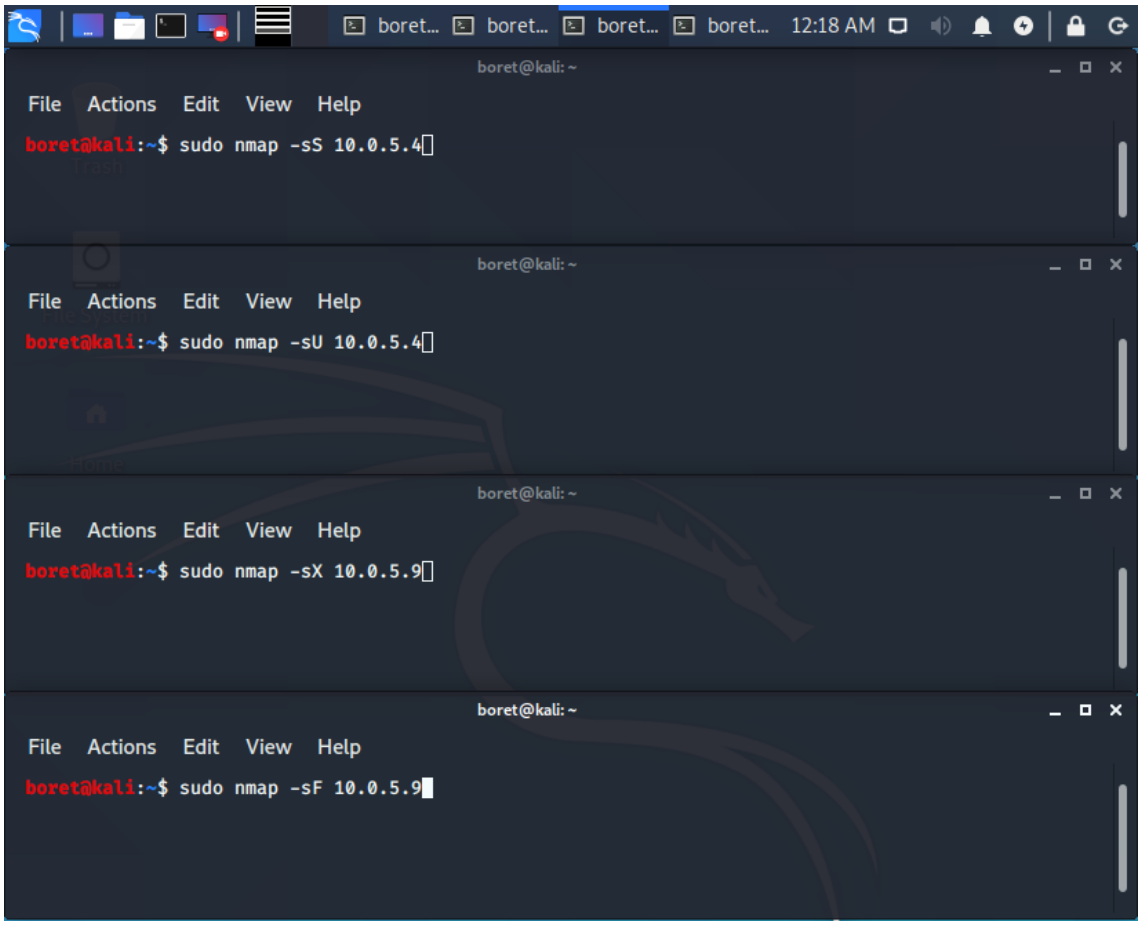

Figura 18: comandos para atacar con nmap

```
 -- runportscan.sh 22:12:06.5224 --> Comenzamos ...
 -- runportscan.sh 22:12:06.5297 --> Ejecutando snort sobre enp0s3 durante 300 segundos
*** Caught Term-Signal
 -- runportscan.sh 22:17:11.5799 --> Ejecutando nmap_counter sobre /tmp/snortOutput_2020-07-05_22-11
-58.log
Se esta recibiendo un escaneo XMAS de 10.0.5.5
Se esta intentando hacer un escaneado de puertos por TCP de 10.0.5.5
Se esta intentando hacer un escaneado de puertos FIN de 10.0.5.5
Se esta intentando hacer un escaneado de puertos UDP de 10.0.5.5
 -- runportscan.sh 22:17:11.5912 --> Terminado correctamente
sec@ids:~/GitHub/tfg_ids_boreto$
```

Figura 19: resultados de la ejecución exitosa del script

7.3. Man-in-the-middle.

Antes de comenzar es preciso explicar que es necesario tener SSH instalado en todas las máquinas virtuales. Esto es preciso porque la máquina IDS se asegura de que no se están sufriendo ataques *man-in-the-middle* preguntando por el túnel SSH sobre la MAC de la máquina en cuestión. Adicionalmente, será necesario crear un fichero llamado *.mitm_detect* en la misma ubicación en la que esté *run_mitm.sh* en la que se introducirán tanto la dirección IP real del Gateway como su MAC, separados por un espacio.

Las pruebas en este caso son por un lado más simples que en los otros módulos, y por el otro lado un poco más complejas. Esto se debe a la forma en la que se da comienzo al ataque desde *Kali.* Se usa ahora un programa llamado *Ettercap.* Esto añade un nivel de facilidad a la hora de hacer las pruebas, pero al mismo tiempo se ha de decir que la interfaz no es suficientemente intuitiva.

Figura 20: pantalla inicial de Ettercap

Tenemos que hacer un escaneo de hosts para saber quién está presente en la red. De esta forma descubrimos a un equipo con la dirección 10.0.5.4, que es el equipo *Ubuntu,* y un host con la dirección 10.0.5.1, que naturalmente es el *Gateway.*

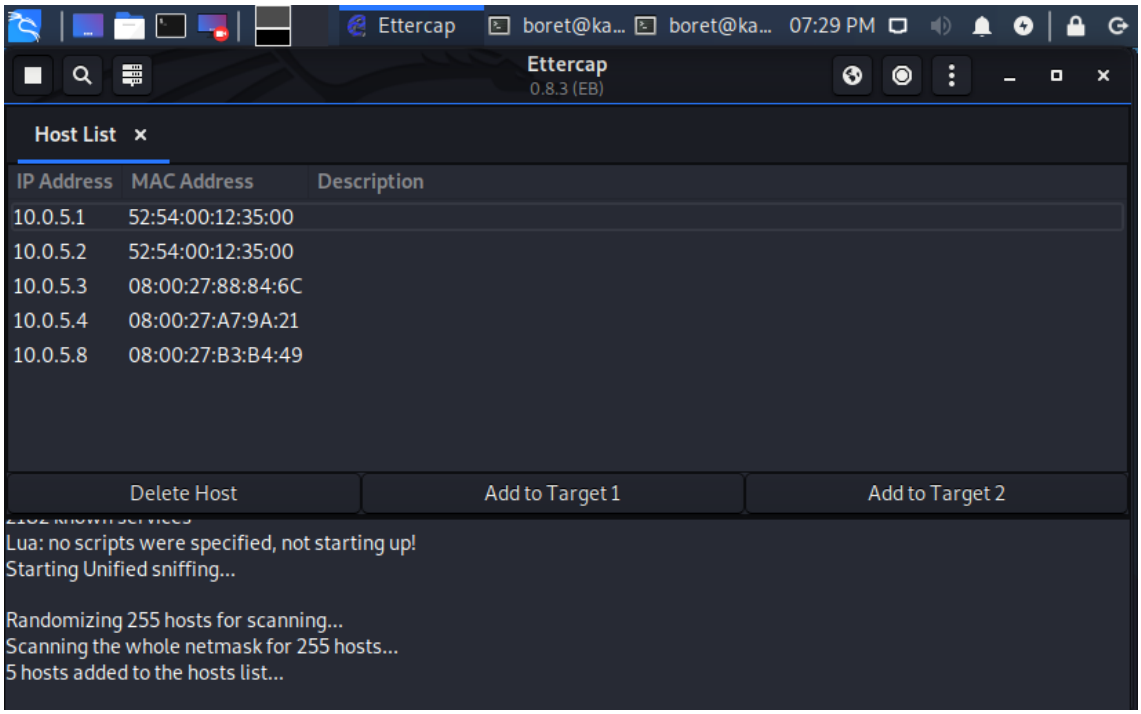

Figura 21: lista de hosts encontrados en la red

Se tienen que añadir uno de los dos hosts como *Target 1* y el otro como *Target 2*. En este caso se usarán 10.0.5.1 y 10.0.5.4. Una vez hecho esto, el siguiente paso es "envenenar" las tablas ARP de los dos hosts. Esto se hace desde el *MITM menú.*

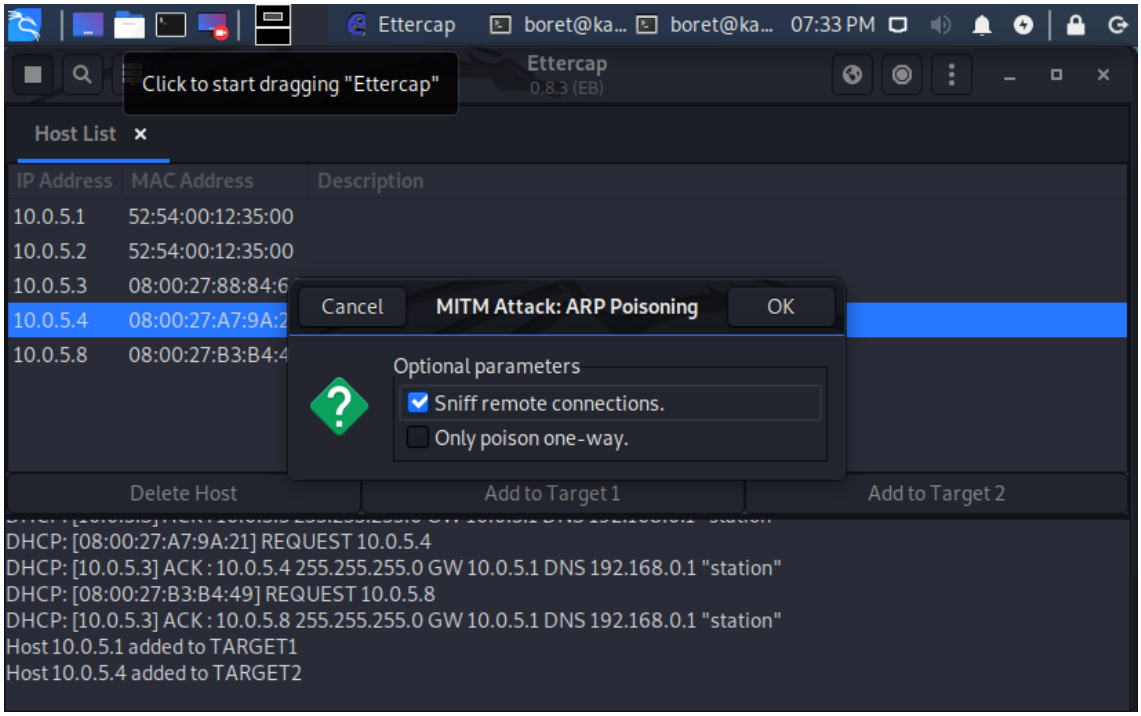

Figura 22: arp poisoning preparado

El momento en el que se pulsa sobre "*Accept*" en el *ARP poisoning* empieza el ataque, si se han seguido todos los pasos previos. Automáticamente el programa se pondrá entre el *Gateway* y el equipo *Ubuntu.* Debería de aparecer *Ettercap* de esta forma:

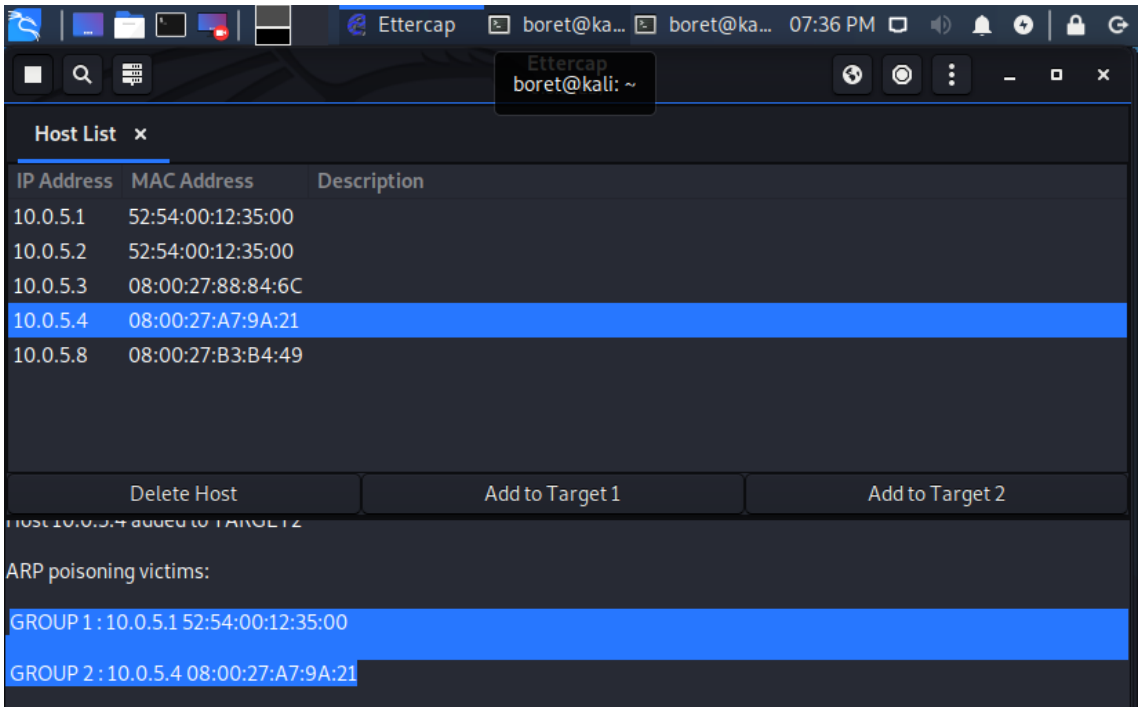

Figura 23: demostración de que el ataque está en proceso

Es ahora cuando toca ejecutar el módulo de detección de *man-in-the-middle* para verificar que se puede detectar este ataque. Al ejecutar el *script* lo primero que se ve es la interfaz del terminal preguntando en qué máquina se quiere comprobar que está sucediendo el ataque:

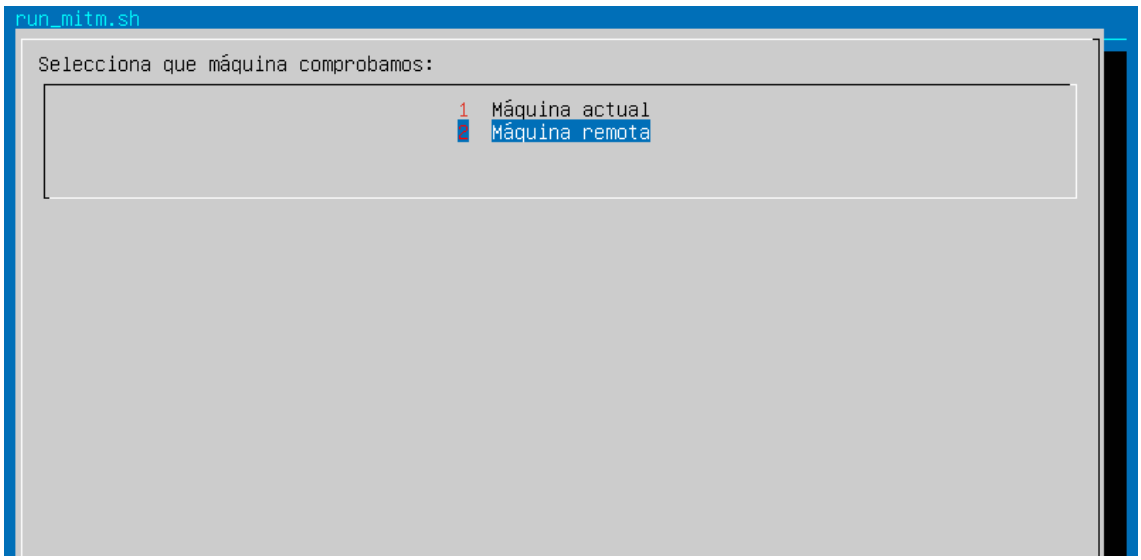

Figura 24: vista de la interfaz recién lanzada

Para el caso actual se selecciona la segunda opción. Después de esto aparece una nueva ventana en la que es necesario introducir la dirección de la máquina que se desea comprobar y cual es su usuario SSH.

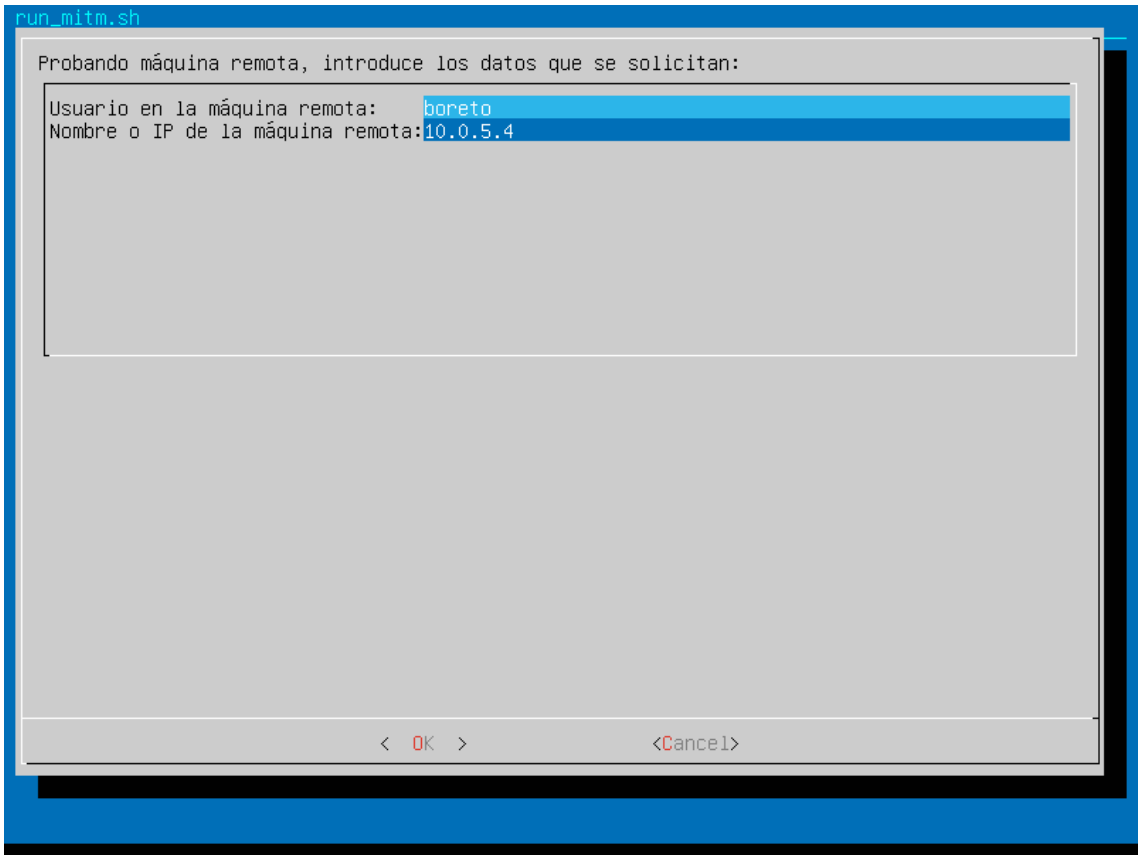

Figura 25: campos a rellenar para ejecutar la comprobación de si todo está yendo bien

A continuación, la siguiente ventana muestra lo que es la dirección MAC detectada en la máquina objetivo o la MAC que se ha detectado por el sistema. Ahora hay 2 opciones diferentes sobre cómo proceder.

- Se puede eliminar esa dirección ofrecida para comparar la dirección obtenida con la introducida manualmente en el fichero *.mitm_scan.*
- Se puede introducir manualmente la dirección MAC del Gateway en ese campo

Si se elige la primera opción se comparará la MAC actual de la máquina objetivo con las que haya en el archivo *.mitm_scan.* En el caso de que se elija la segunda se comparará la que aparezca en el cuadro de texto con la obtenida en la máquina destino. Para esta prueba se escoge la primera opción.

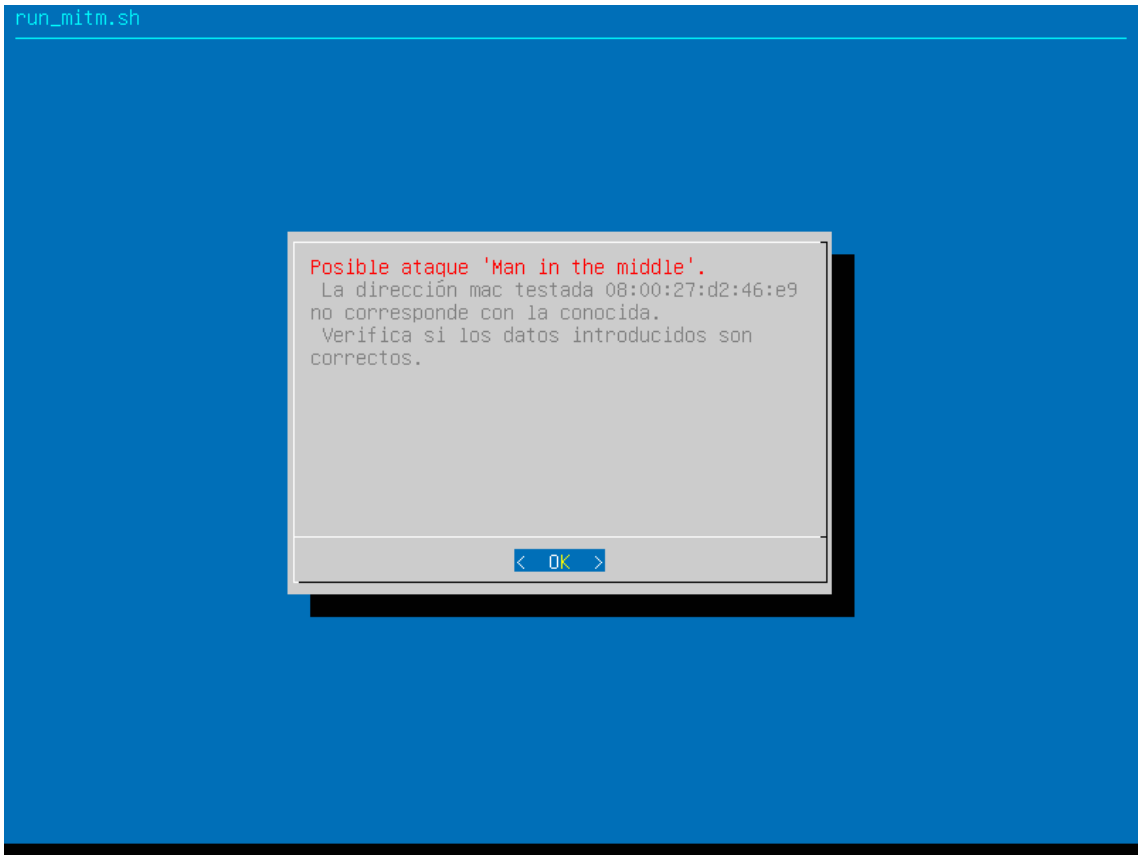

Figura 26: resultado final del programa sobre máquina comprometida

7.4. Pruebas de detección con otros programas.

Con el objetivo de demostrar que el IDS reacciona correctamente a ataques generados con diferentes herramientas, se realizaron pruebas adicionales. A continuación, se van a mostrar los resultados de otras pruebas con otros softwares diferentes.

El primer caso ha sido con el típico ping. Se lanzaban ataques desde 6 terminales diferentes de la siguiente forma:

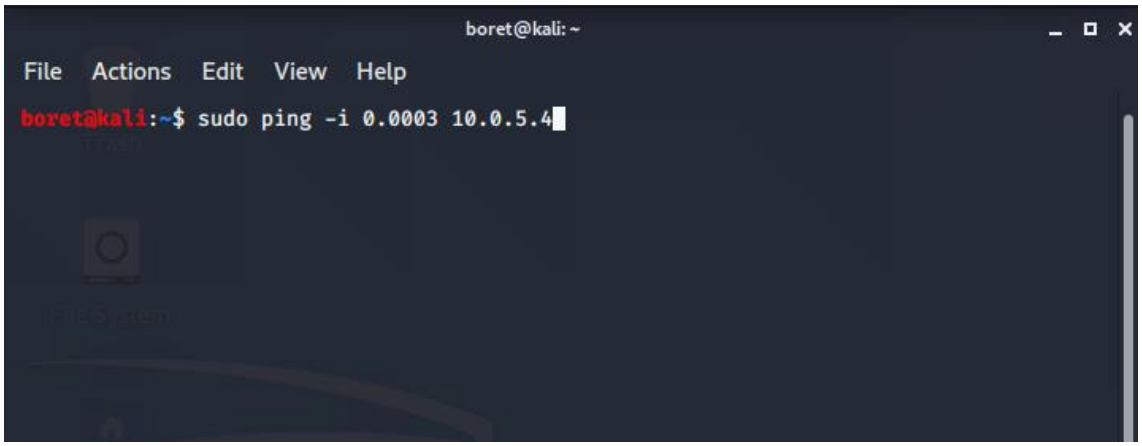

Figura 27: ping con el comando para hacer un ataque DOS

El resultado, exitoso, fue:

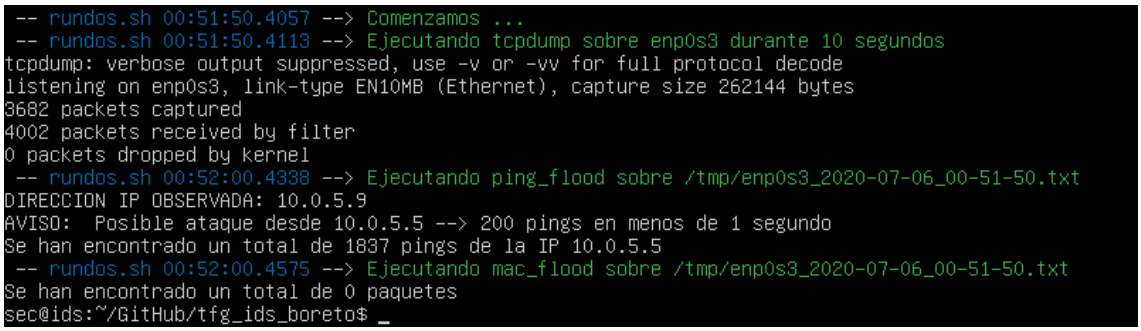

Figura 28: resultado de ejecutar rundos.sh durante el ataque con scapy

A continuación, se prueba a hacer un escaneo de puertos con *netcat* a ver si el módulo de detección de escaneos lo detecta. Se ejecuta netcat con el siguiente comando:

sudo nc -znvv 10.0.5.9 1-1000

El módulo de detección de escaneos lo detecta y este es el resultado:

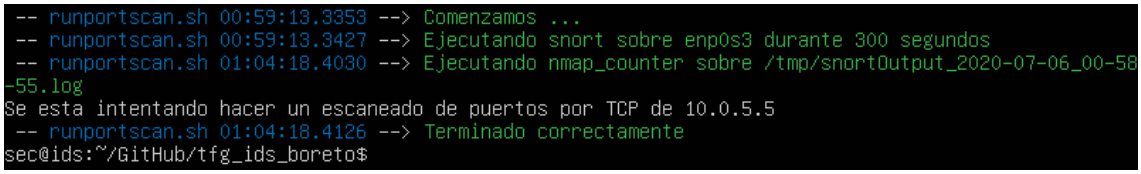

Figura 29: resultado de ejecutar runportscan.sh durante el escaneo de netcat

La última prueba que se va a hacer ahora es la de realizar un ataque *man-in-the-middle* usando *arpspoof.* El comando tiene que lanzarse en las dos direcciones de forma similar:

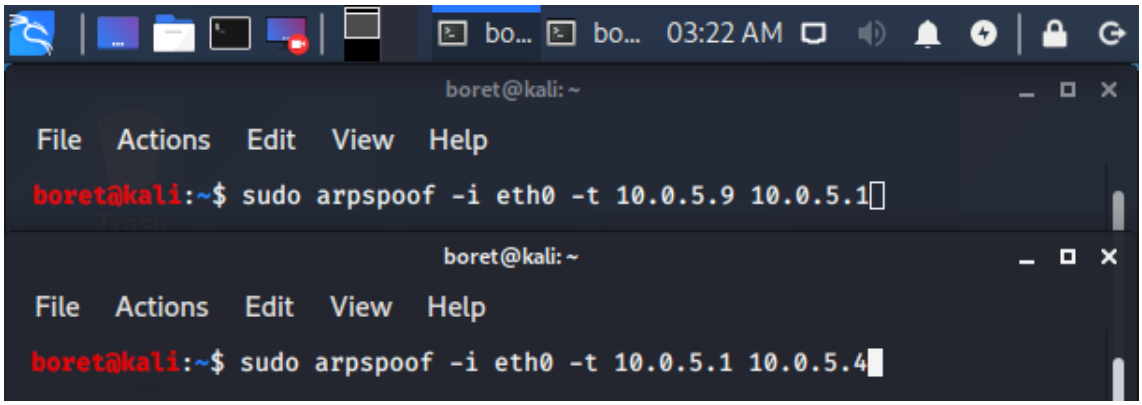

Figura 30: arspoof en las dos direcciones para monitorizar todo el tráfico

El resultado, donde 08:00:27:d2:46:e9 es la MAC de la máquina *Kali.*

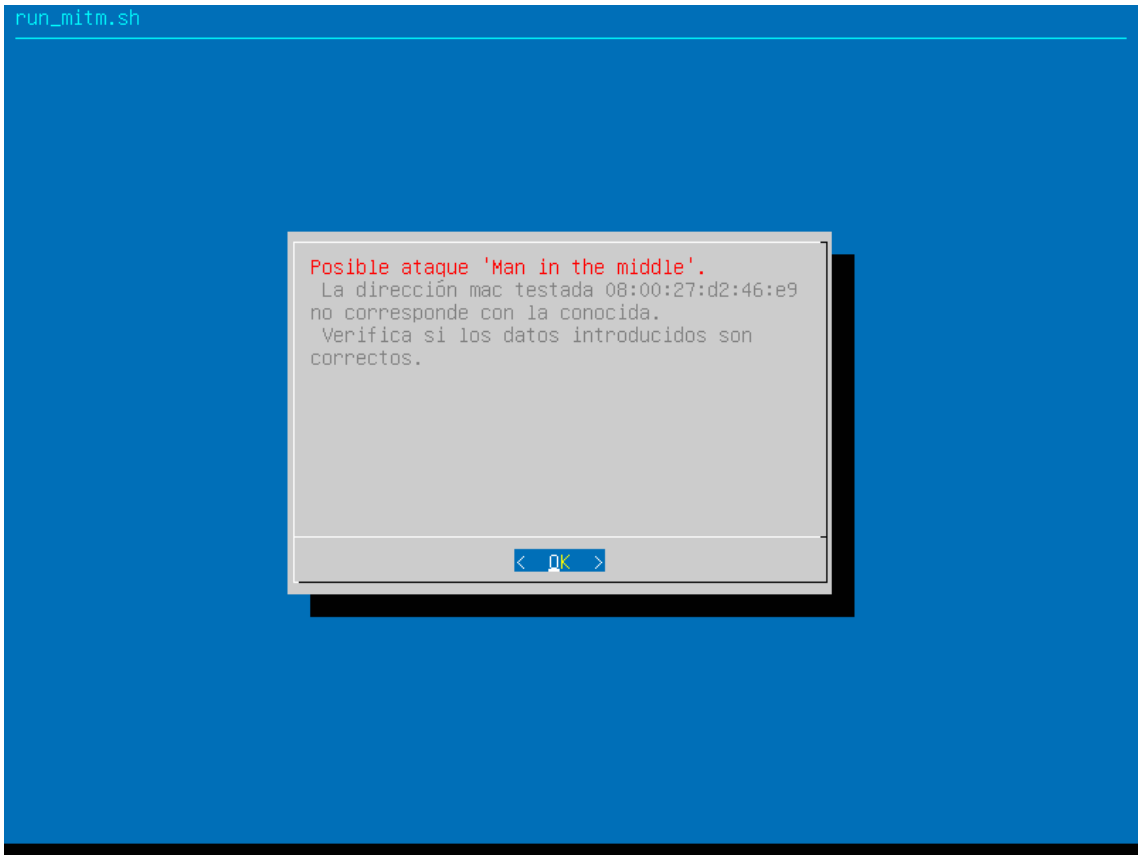

Figura 31: script run_mitm.sh detectando el ataque con arpspoof

8. Conclusión

Una vez acabadas las pruebas finales, solo queda hacer una pequeña reflexión sobre lo mucho que se ha avanzado en el proyecto, desde sus versiones más iniciales hasta la actualidad. Desde el momento en el que los objetivos planteados parecían una montaña infranqueable a la actualidad. Y es en eso en lo que se va a centrar este último apartado, en si se han resuelto los objetivos definidos en la introducción o no y en el por qué, además de hablar sobre los conocimientos adquiridos en el proceso de desarrollo y su relación con la materia vista a lo largo de la carrera.

Para empezar, se ha conseguido implementar y simular un sistema de detección de intrusiones básico que detecta algunos de los ataques más comunes que se pueden encontrar en el mercado. Esto lo ha hecho utilizando exclusivamente software gratuito, tanto el software empleado para instalar y ejecutar las máquinas virtuales, como los sistemas operativos en sí y los programas y herramientas usados para su funcionamiento. Con esto ya se puede decir que se ha cumplido el objetivo principal del proyecto, además de dos objetivos secundarios.

Se han cumplido además todos los objetivos secundarios. Se han investigado las técnicas de detección existentes usadas por otros softwares, se han simulado ataques exitosamente y se ha automatizado la detección de ataques. Se ha conseguido implementar el IDS y verificado su correcto funcionamiento. Tal vez con el tiempo se consiga a su vez concienciar a la gente sobre los riesgos de depender de compañías de pago para la ciberseguridad.

Pese a esto, hay que subrayar que al usar máquinas virtuales en un solo sistema resultó imposible replicar con total precisión un ataque de denegación de servicio distribuido; ya que hacerlo supondría el uso concurrente de una gran cantidad de máquinas virtuales, así como un consumo excesivo de recursos del ordenador.

Pero no se llegó hasta el final sin pasar por el principio, y es que previo al desarrollo del proyecto hubo una fase de investigación, en la que se intentó estudiar primeramente cómo funcionaba el software usado por grandes empresas del mercado e intentar replicarlo. Esto resultó ser un fracaso y finalmente se decidió que la mejor opción disponible sería pasarse al software gratuito e intentar usarlo en vez de replicarlo. El cambio de perspectiva resultó ser un gran punto positivo para el desarrollo que permitió establecer unas metas realistas y contribuyó enormemente al éxito de la fase de desarrollo.

El hecho de poder usar y depender de software gratuito ha ayudado mucho más a entender nuevo software que no se había usado previamente de lo que se esperaba. Se han tenido que usar programas y herramientas de las cuales previamente apenas se sabía nada. Una excepción podría ser *snort* ya que en una asignatura del último curso se vio un poco por encima, pero fue un grupo diferente al que el alumno participó el que lo llegó a desarrollar.

Ejemplos de programas que resultaron novedosos para el alumno son todas las herramientas usadas de *Kali* como *nmap* o *Ettercap.* Incluso algunos de los programas como *tcpdump* también eran nuevos y se tuvo que aprender exactamente qué hacían, cómo funcionaban y en qué forma se podían usar para desarrollar el proyecto.

Aprender a usar nuevas herramientas o analizar un problema de la magnitud del planteado no supuso un exceso de dificultad, ya que a lo largo de la carrera se han visto una multitud de programas diferentes, de los cuales se han tenido que aprender a usar todos, de forma que ya se estaba acostumbrado.

Hay que destacar, al mismo tiempo, que se cometieron errores a lo largo del desarrollo. A modo de ejemplo al principio en vez de usar *snort* para detectar ataques de escaneo de puertos se había decidido usar una herramienta mucho menos conocida: *psad.* Esta herramienta era mucho más rudimentaria que *snort* y llevaba sin soporte alguno años, eso combinado con el hecho de que dio problemas a la hora de acceder a los *logs* y que a veces incluso no detectaba los ataques acabó llevando a que fuera desestimada para la tarea y se cambiara a *snort,* lo que acabó resultando ser la elección superior. Esto fue posible porque se determinó que no era rentable ser cabezota y centrarse en un programa sin soporte.

Finalmente, a modo de resumen, se ha aprendido a usar herramientas de las cuales algunas ya eran conocidas, pero aún no habían sido usadas: *tcpdump, nmap* y *snort.* Otras más fueron completamente nuevas: *ettercap, macof* y *hping3.* Sobre los lenguajes de programación hay que destacar que no se había programado nunca un *script* a mano para *bash* en *Linux,* ya que en la rama de Tecnologías de la Información no se ha dado.

8.1. Relación del trabajo desarrollado con los estudios cursados.

Sobre este punto, hay que empezar diciendo que se considera que si no se hubiera estudiado la carrera no se habría podido hacer esto. Con esto lo que se quiere decir es que han sido los conocimientos aprendidos a lo largo de las últimos 4 cursos lo que han facilitado por ejemplo tanto los conocimientos necesarios para programar en C la mayoría de los módulos, como los necesarios para no ir completamente perdido a la hora de empezar a programar en *bash.*

La rama escogida por el alumno fue Tecnologías de la Información, en la cual se dan principalmente asignaturas relacionadas con redes informáticas, adicionalmente, en el cuatrimestre 4B se escogieron asignaturas que estaban generalmente centradas en la ciberseguridad, así que no es sorpresa que se haya escogido desarrollar un trabajo de fin de grado sobre la ciberseguridad en las redes. Todo esto no quiere decir que otras asignaturas no hayan influido en el contenido de este proyecto. Por ejemplo, la decisión de programar en C se vio enormemente influenciada por el hecho de que hacía relativamente poco en las prácticas de la asignatura redes multimedia se estuvo programando en C sobre métodos de compresión de imágenes para ser enviadas a

través de Internet. Esto puso la semilla de la duda ante la concepción que se tenía sobre este lenguaje y sirvió para darse cuenta de que se podía adaptar, directa o indirectamente a un proyecto como el presentado.

Hablando de lenguajes de programación, cuando se tuvo que empezar a programar en *bash,* o a aprender a usar otras herramientas no vistas antes; el alumno tuvo que empezar desde 0, lo que podría haber supuesto un problema para otros, pero aquí no fue el caso. A lo largo de la carrera se han enseñado una multitud de lenguajes, formas de programar y herramientas y programas diferentes, de los cuales la mayoría resultaban completamente nuevos para el alumno; esto ha servido como preparación para lo que se venía al principio de este proyecto, ha servido para acostumbrar a aprender cosas nuevas rápida y efectivamente si guardan similitud con lo visto anteriormente. Sin esto habría sido considerablemente más difícil realizar el proyecto exitosamente.

9. Referencias

[1] GitHub - February 28th DDoS Incident Report. Accedido el 20/03/2020. https://github.blog/2018-03-01-ddos-incident-report/

[2] Dyn – Dyn statement on 10/21/2016 DDoS attack. Accedido el 24/03/2020. https://dyn.com/blog/dyn-statement-on-10212016-ddos-attack/

[3] Akamai Research, State of the Internet / Security | 2019 – A Year in Review (Volume 5, Issue 6) | Akamai

[4] Akamai Research, State of the Internet / Security Report | Financial Services – Hostile Takeover Attempts (Volume 6, Issue 1) | Akamai

[5] Antar Abdul-Qawy, Srinivasulu Tadisetty, "The Internet of Things (IoT): An Overview", Diciembre de 2015

[6] Nickolaos Koroniotis, Nour Moustafa, Elena Sitnikova, Benjamin Turnbull, "Towards the Development of Realistic Botnet Dataset in the Internet of Things for Network Forensic Analytics: Bot-IoT Dataset", 2018.

[7] Roger Hallman, Josiah Bryan, Geancarlo Palavicini Jr, Jose Romero-Mariona, "IoDDoS — The Internet of Distributed Denial of Service Attacks: A Case Study of the Mirai Malware and IoT-Based Botnets", Aril de 2017

[8] David J. Day, Benjamin M. Burns, "A Performance Analysis of Snort and Suricata Network Intrusion Detection and Prevention Engines", Febrero de 2011

[9] Suricata – Suricata User Guide. Accedido el 14/04/2020 https://suricata.readthedocs.io/en/suricata-5.0.3/

[10] Security Onion – Securiry Onion Documentation. Accedido el 09/06/2020 https://securityonion.readthedocs.io/en/latest/

[11] Sagan – Sagan User Guide. Accedido el 12/06/2020 https://sagan.readthedocs.io/en/latest/

[12] Zeek – Zeek User Manual. Accedido el 12/06/2020 https://docs.zeek.org/en/current/

[13] Solarwinds – Network Performance Monitor Administrator Guide. Accedido el 17/04/2020 https://documentation.solarwinds.com/en/success_center/NPM/content/npm_administrator_guide.htm

[14] Bace Rebecca, Mell Peter. "Intrusion Detection Systems", Nist Special Publication on Intrusion Detection Systems, Noviembre de 2001

[15] Ansam Khraisat, Iqbal Gondal, Peter Vamplew and Joarder Kamruzzaman, "Survey of intrusion detection systems: techniques, datasets and challenges", Cybersecurity (2019)

[16] Robert Mitchell, Ing-Ray Chen. “A survey of intrusion detection techniques for cyber-physical systems”. ACM Comput. Surv. 46, 4, Article 55. Abril de 2014

[17] C.Kolias G.Kambourakis M.Maragoudakis, "Swarm intelligence in intrusion detection: A survey" Computers & Security, Volume 30, Issue 8, Pages 625-642, November 2011

[18] Khloud Al Jallad, Mohamad Aljnidi, Mohamad Said Desouki, "Big Data Analysis and Distributed Deep Learning For Next-Generation Intrusion Detection System Optimization", Journal of Big Data, 2019

[19] Wang K., Stolfo S.J, “Anomalous Payload-Based Network Intrusion Detection”. In: Jonsson E., Valdes A., Almgren M. (eds) Recent Advances in Intrusion Detection. RAID 2004. Lecture Notes in Computer Science, vol 3224. 2004

[20] Khaled M. Elleithy, Drazen Blagovic, Wang Cheng, Paul Sideleau. “Denial of Service Attack Techniques: Analysis, Implementation and Comparison”, Enero de 2006.

[21] Monowar H Bhuyan, D. K. Bhattacharyya, J. K. Kalita. “Surveying Port Scans and Their Detection Methodologies”, Octubre de 2011

[22] Pushpendra Kimar Pateriya, Srijith S. Kumar, “Analysis on Man in the Middle Attack on SSL”, in International Journal of Computer Applications in Technology. Mayo de 2012

[23] Mario Surco, “Mitigación del ataque ‘Hombre en el Medio’ en las Redes Locales”, en Revista I+i Investigación aplicada e innovación. Volumen 6, 2012

[24] Zanero Stephano, "Flaws and frauds in the evaluation of IDS/IPS technologies", Marzo de 2013