Universidad de Valladolid

Trabajo de Fin de Grado

Sistema de desvío de intrusiones de red hacia honeynets dinámicas virtuales

---

Autora:
Helena Carbajo Olmedo

Tutor:
Federico Simmross Wattenberg

Departamento

Departamento de Teoría de la Señal y Comunicaciones e Ingeniería Telemática

13 de mayo de 2018

«*Que seas una personalidad no significa que tengas personalidad.*»

Señor Lobo *en la película* "Pulp Fiction"

UNIVERSIDAD DE VALLADOLID

Resumen

Escuela Técnica Superior de Ingenieros de Telecomunicación
Departamento de Teoría de la Señal y Comunicaciones e Ingeniería Telemática

Grado en Tecnologías Específicas de Telecomunicación. Mención en Telemática

Sistema de desvío de intrusiones de red hacia honeynets dinámicas virtuales

por Helena Carbajo Olmedo

Durante los últimos años, la creciente expansión de Internet en todos los ámbitos ha supuesto también un aumento de los ataques que se producen, tanto contra individuos como organizaciones. Desarrolladores y personal de seguridad buscan día a día soluciones que mejoren los sistemas actuales, haciéndolos más seguros y con menos vulnerabilidades. No obstante, este progreso tiene su réplica entre los atacantes, que también avanzan en las técnicas que emplean para perpetrar sus ataques. Ante esta situación, no basta con repeler los ataques, sino que es necesario estudiar a fondo cómo se realizan, para aprender de ellos y ganar cierta anticipación. Resulta, por lo tanto, fundamental disponer de algún sistema que nos permita observar el comportamiento de un atacante. Teniendo en mente este propósito, en el presente trabajo se ha empleado un sistema de detección de intrusiones, *Snort*, con el que se detectarán posibles ataques, para después desviarlos hacia una red virtual, *honeynet*, creada con la tecnología de virtualización KVM, de manera que el atacante no sea consciente de ello y su actividad pueda ser estudiada, al tiempo que se neutraliza el ataque.
El trabajo se evalúa mediante la realización de diversas pruebas tanto para detectar los ataques, como para desviarlos.

Agradecimientos

Quisiera reservar unas líneas para agradecer a todas aquellas personas que me han acompañado durante esta etapa de mi vida.
A mi tutor, Federico, por brindarme la oportunidad de realizar un trabajo tan interesante y guiarme en todo momento.
A mis amigos que, de una manera u otra, siempre están ahí cuando se les necesita. Gracias por animarme (y aguantarme) en los momentos de frustración.
A José, mi maestro, que no es solo eso, sino un amigo en quien puedo confiar y a quien recurrir cuando necesito consejo.
A Javi, mi gurú de los *scripts*, por compartir conmigo este largo recorrido y apoyarme cuando más lo necesito.
Y, finalmente, me gustaría agradecer especialmente a mis padres todo lo que hacen por mí. A mi madre, que ha ido leyendo toda esta memoria con entusiasmo para ayudarme con las erratas. A mi padre, que siempre tiene tiempo para echarme un cable. Quiero agradeceros que siempre hayáis confiado en mi criterio y en mis decisiones, aunque en ocasiones pudieran parecer algo disparatadas. Todo apunta a que, al final, esta fue la carrera definitiva. Pero lo que verdaderamente quiero agradeceros es lo geniales que sois. Sé que tengo mucha suerte de tener unos padres como vosotros, con los que puedo compartir tantas cosas, ya sea ir al cine o al teatro, entrenar o, simplemente, ver "Los Simpsons". No podríais ser mejores.

Índice general

Índice de figuras

Índice de Tablas

Lista de abreviaturas

CIA	**C**onfidenciality **I**ntigrity **A**vailability
TTD	**T**ime **T**o **D**etect
TTE	**T**ime **T**o **E**volve
NIST	**N**ational **I**nstitue (of) **S**tandards (and) **T**echnology
RFC	**R**equest **F**or **C**omments
ISOC	**I**nternet **S**ociety
IDS	**I**ntrusion **D**etection **S**ystem
TCP	**T**ransmission **C**ontrol **P**rotocol
UDP	**U**ser **D**atagram **P**rotocol
SSH	**S**ecure **SH**ell
HIDS	**H**ost (based) **I**ntrusion **D**etection **S**ystem
NIDS	**N**etwork (based) **I**ntrusion **D**etection **S**ystem
DMZ	**D**emilitarized **Z**one
IP	**I**nternet **P**rotocol
MTU	**M**aximum **T**ransfer **U**nit
OISF	**O**pen **I**nformation **S**ecurity **F**oundation
IDS	**I**ntrusion **P**revention **S**ystem
VRT	**V**ulnerability **R**esearch **T**eam
SDN	**S**oftware **D**esign **N**etwork
KVM	**K**ernel **V**irtualization **M**achine
API	**A**pplication **P**rogramming **I**nterface
GUI	**G**raphical **U**ser **I**nterface
CLI	**C**ommand **L**ine **I**nterface
XML	**E**xtensible **M**arkup **L**anguage
QCOW	**Q**EMU **C**opy **O**n **W**rite
VNC	**V**irtual **N**etwork **C**omputing
ICMP	**I**nternet **C**ontrol **M**essage **P**rotocol
NAT	**N**etwork **A**ddress **T**ranslation
GNU	**G**eneral **P**ublic **L**icense

Capítulo 1

Introducción

A lo largo de este primer capítulo se explica el contexto en el que se sitúa el presente TFG, justificándose sus aportaciones. También se exponen los objetivos a conseguir, la metodología y fases empleadas, así como la estructura del resto de la memoria.

1.1. Contextualización

Es posible encontrar múltiples definiciones de lo que se entiende por seguridad en Internet. Por ejemplo, Stallings [1] se refiere a ello como la protección que se proporciona a un sistema de la información para preservar la integridad, disponibilidad y confidencialidad de sus recursos, tanto software como hardware. Por otro lado, la empresa Cisco la define como la actividad destinada a proteger la usabilidad y la integridad de la red y datos. Al igual que la anterior, engloba medios software y hardware [2]. Estas son tan solo dos ejemplos de las muchas acepciones que existen, pero es posible observar que coinciden en gran medida en los aspectos que la seguridad, en términos de Internet, debería garantizar. Ambas también establecen los mismos objetivos a proteger: medios hardware y software. Es importante entender en que consisten la integridad, la disponibilidad y la confidencialidad, conjunto conocido como *CIA* (*Confidentiality, Integrity and Availability*). La confidencialidad asegura que los datos de carácter privado no sean accedidos por personas no autorizadas; la disponibilidad permite que datos o cualquier otro tipo de recurso pueda ser utilizado sin ningún tipo de impedimento y, finalmente, la integridad preserva el contenido de los datos o comportamiento de un sistema, de manera que estos no sean modificados por alguien desautorizado. Todos estos términos pueden aplicarse a un sistema aislado, pero cuando este sistema pasa a estar en una red de millones de nodos, las amenazas se múltiplican.

1.1.1. Tipos de atacantes

Se distinguen distintos tipos de atacantes: *hackers*, criminales y empleados. Los hackers suelen realizar ataques buscando la emoción de conseguir acceder a un sistema restringido y el reconocimiento del resto de la comunidad *hacker*. Son frecuentes los ataques de oportunidad que aprovechan alguna vulnerabilidad para acceder a información que luego comparten en la red. Los segundos atacantes, los criminales, constituyen bandas de *hackers* que se asocian para llevar a cabo ataques con fines lucrativos, generalmente contra servicios de comercio electrónico. Tratan de hacerse con datos bancarios y tarjetas de crédito que después utilizan a expensas de la víctima o venden en la red. Estos grupos, que se han expandido por toda la red [1], suponen una amenaza común para todos los sistemas basados en Internet, buscan objetivos concretos y, en ocasiones, son contratados por gobiernos u otras organizaciones. Por último,

los empleados son individuos que ya se encuentran dentro del sistema y conocen su estructura. Sus ataques pueden estar motivados por venganza contra la organización en la que trabajan o sencillamente, por un sentimiento de derecho. Resultan, por lo tanto, los ataques más difíciles de detectar y prevenir, y solamente políticas de acceso y monitorización dentro de la organización ayudan a evitarlos.

1.1.2. Ataques de seguridad y sus motivaciones

Las normas *X.800* [3] y *RFC 4949* [4] clasifican los ataques en dos categorías: pasivos y activos. Los ataques pasivos serían aquellos que extraen información de un sistema, pero no alteran en modo alguno a sus recursos. Un ejemplo sería la monitorización de las transmisiones realizadas entre dos sistemas, accediendo a esta información. Por otro lado, los ataques activos sí que afectan a los recursos de un sistema e incluso a su funcionamiento. Dentro de este tipo de ataques [1] se encuentran la suplantación, cuando un individuo u organización finge ser otra distinta; reenvio de información capturada previamente sin autorización; modificación de mensajes o la denegación de servicio, que impide el acceso normal a un servicio. En lo que respecta a las motivaciones, la gran mayoría de ataques están conducidos por el espionaje o un interés financiero. Otras motivaciones serían la diversión, el resentimiento o la ideología. No obstante, hay que tener en cuanto que muchos casos de extorsión no son reportados y confirmados, por lo que las cifras recogidas en las estadísticas no reflejan la totalidad de los ataques. Aún así, es posible contar con una referencia de los fines que persiguen algunos conocidos ataques[5]:

- Financieros: uso de credenciales robadas, uso de *backdoor*, *spyware*, *phising*, *malware* para exportar data, *c2*[1].
- Espionaje: *phising*, *c2*, uso de *backdoor*.
- Resto: abuso de privilegios.

1.1.3. Algunas cifras concretas

En relación a lo anterior, existen multitud de estudios e informes que tratan de recabar información acerca del estado de la seguridad en Internet partiendo de diversas fuentes, como encuestas o ataques sufridos. Pese a que gran parte de estos estudios están realizados por empresas privadas, resultan útiles para obtener una perspectiva global del problema que supone la seguridad en Internet.

DBIR 2017

El DBIR (*Data Breach Investigations Report*), un informe realizado por Verizon [5] en el que participan 65 organizaciones, analiza el estado de la ciberseguridad.Según este informe hubo 1616 ataques durante el año 2016, de los que 828 supusieron la revelación de datos confidenciales. Este informe también proporciona los tipos de ataques más conocidos, así como los actores que los perpetran y sus motivaciones. Plasmando en cifras lo referido en la anterior sección, el 66 % de los ataques tenían una motivación financiera y el 33 % de espionaje. Menos del 1 % de los ataques fueron motivados por ideología o diversión. Además, el 99 % de estos ataques los llevaron a cabo individuos u organizaciones externas. Otro dato interesantes que recoge el informe es que la táctica más empleada es la de *phising* y que la mayoría de estos

[1] *Command and control.*

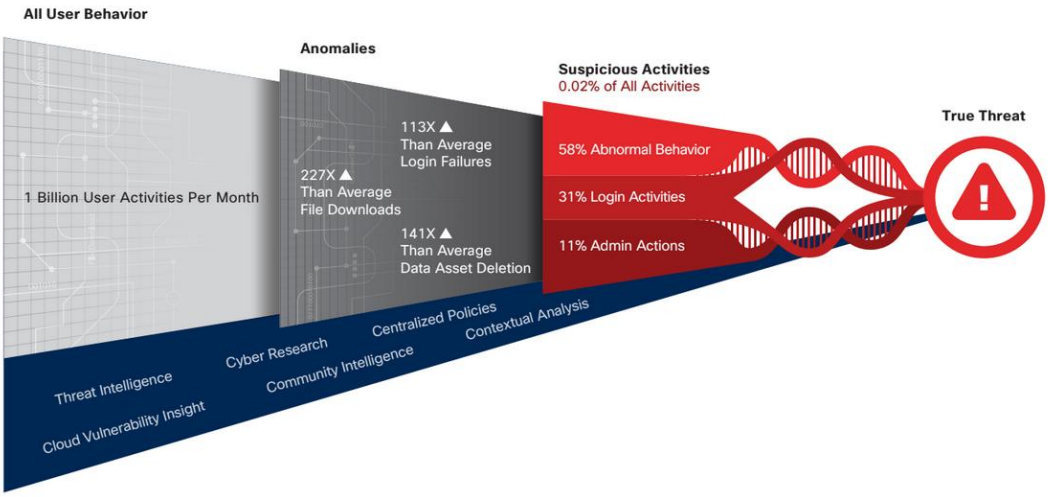

FIGURA 1.1: Identificación de patrones de comportamiento de un usuario normal [6].

ataques van seguidos por la instalación de algún tipo de *malware*. Finalmente, cabe mencionar que el 81 % de los ataques con éxito se produjeron debido a credenciales inseguras o robadas. Este informe además de analizar las estadísticas de los ataques *reportados*, proporciona algunos consejos basándose en los resultados para tratar de evitarlos. Destaca, sobre todo, la necesidad de concienciar y educar acerca de las amenazas y riesgos que existen. También pone el foco en la importancia que supone la detección temprana de un ataque y la localización de la fuente del ataque.

Cisco 2017 Annual Cybersecurity Report

El grupo de investigación de seguridad de Cisco [6] publica cada año este informe, para ayudar a las organizaciones a hacer frente a las amenazas y riesgos que surgen constantemente en la red. Entre los datos recogidos por el informe cabe destacar las razones que impiden la adopción de sistemas u otras medidas de seguridad en muchas empresas: el 35 % carecía de presupuesto, para el 28 % presentaba problemas de compatibilidad, el 25 % por la certificación y el 25 % restante por falta de talento. Por este motivo, apenas la mitad de las alertas de seguridad que se reciben son investigadas. Cabe mencionar también el hecho de que aquellas organizaciones que aún no han sufrido ninguna ruptura de seguridad están convencidas de que su red es segura, aunque esta seguridad parece cuestionable si se tiene en cuenta el grado de afectación que supone para cualquier empresa que su sistema se vea comprometido. Casi un cuarto de las empresas perdió alguna oportunidad de negocio al sufrir un ataque y una de cada cinco perdió clientes. Este estudio también muestra que muchas de las empresas recurren a las soluciones de seguridad de varias empresas especializadas, por lo general más de 5, con varios productos distintos también. Todo ello supone una complejidad extra que dificulta la automatización de tareas, algo fundamental a la hora de mejorar la seguridad de un sistema. Por ejemplo, distinguir un comportamiento anómalo y sospechoso del que, según los patrones, resulta normal requiere un proceso de varias etapas que solo puede lograrse con automatización, véase la figura 1.1.
En lo que se refiere a los ataques, los datos revelan que en la mayoría de ellos se distinguen las siguientes fases:

- Reconomiento: los atacantes investigan, identifican y seleccionan a sus víctimas.

- Armamento: generación de paquetes con *malware* que permite el acceso remoto aprovechando una vulnerabilidad.
- Distribución: la carga anterior se hace llegar mediante correo, ficheros adjuntos, etc.
- Instalación: en el objetivo, el *malware* genera un *backdoor* que permite el acceso permanente de los atacantes.

Frente a los ataques, una de las medidas que propone Cisco para conocer el progreso de las medidas de seguridad es el TTD (*Time To Detec*). Lo define como el intervalo de tiempo que transcurre desde que un sistema se ve comprometido hasta que la amenaza es detectada. Las amenazas y ataques evolucionan muy rápidamente y en ocasiones resulta difícil identificar un ataque, aunque este sea conocido en la comunidad. De la misma manera que los sistemas de seguridad trabajan en mejorar el TTD, los atacantes desarrollan nuevas técnicas y estrategias para evitar ser detectados y disponer así de más tiempo para perpetrar su ataque. Esta mejora en los ataques se puede medir con el TTE (*Time To Evolve*), el tiempo que tarda un atacante en modificar el modo en que cierto malware es distribuido o en cambiar de táctica. El hecho de que los ataques evolucionen con tanta rapidez denota, a su vez, las mejoras que experimentan los sistemas de seguridad.
Este progreso constante en ambos partes, ha supuesto un incremento en el personal dedicado exclusivamente a la seguridad en las empresas. Frente a las 25 personas que se registraron de media en cada organización durante el año 2015, el año 2016 esta cifra era de 33. El interés por combatir el progreso de las amenazas en Internet radica en el impacto que estos ataques tienen en una organización o empresa. La repercusión no se limita a cortes de servicio, con la consecuente pérdida de dinero, sino que afecta gravemente a la reputación. De hecho, el 33 % de las organizaciones encuestadas tuvieron que hacer frente a la publicación involuntaria de ataques que sufrieron.
El informe concluye que toda organización es susceptible de sufrir un ciberataque, siendo necesaria una constante mejora en los medios de seguridad teniendo en cuenta, además, las limitaciones de presupuesto y compatibilidad, entre otras.

1.1.4. Búsqueda de soluciones contra los ciberataques

Los informes comentados anteriormente reflejan tan solo una pequeña parte del total de informes elaborados por empresas y organizaciones. Pese a que ambos han sido llevados a cabo por empresas privadas con sus propios intereses y es necesario estudiarlos con una actitud crítica, sí que ponen de manifiesto la magnitud del problema que supone la seguridad en Internet. Como puede extraerse de los datos, en la red toda organización es susceptible de convertirse en víctima de un ataque y la necesidad de sistemas seguros se mantiene.
Se ha discutido anteriormente cómo son y en qué consisten algunos de los ataques más frecuentes de la red, pero ¿qué soluciones existen para hacerlos frente? Se han desarrollado multitud de estándares que abarcan desde técnicas de gestión de ataques, hasta arquitecturas recomendables[1]. Algunas de las instituciones más importantes que realizan esta tarea son el NIST(*National Institue of Standards and Technology*), una agencia federal de Estados Unidos; o la ISOC (*Internet Society*), que elabora RFCs(*Requests For Comments*)[1]. Además de estos estándares, hay diversos productos concretos que los expertos en ciberseguridad utilizan. La organización secTools proporciona una lista de las herramientas más populares[7]:

- **Wireshark**[8]: se trata de un analizador de protocolos, que permite estudiar el tráfico de red.
- **Metasploit**[9]: herramienta para probar y desarrollar código de *exploits*.
- **Nessus**[10]: permite realizar escáneres.
- **Aircrack**[11]: utilidad para romper algoritmos de encriptación y recuperar contraseñas para los protocolos 802.11a/b/g WEP and WPA.
- **Snort**[12]: es un IDS(*Intrusion Detection Sistem*), que tal y como su nombre indica, se emplea para detectar instrusiones.
- **Cain and Abel**[13]: herramienta para recuperar contraseñas en Windows.
- **BackTrack**[14]: distribución de Linux que engloba herramientas forenses y de seguridad.
- **Netcat**[15]: permite leer y escribir datos a través de conexiones TCP y UDP.
- **Tcpdump**[16]: al igual que Wireshark analiza el tráfico de red, pero se trata de una herramienta de línea de comandos.
- **John the Ripper**[17]: se utiliza para *crackear* contraseñas en sistemas Unix y Mac, permitiendo detectar contraseñas débiles.

Como puede comprobarse, gran parte de estas herramientas son empleadas tanto para auditoría, como para perpetrar ataques, puesto que a la hora de comprobar la eficacia de cualquier sistema de seguridad resulta fundamental ponerlo a prueba con ataques reales. Y es ahí donde entran en juego lo que se conoce como *honeypots*, de los que se hablará a continuación.

1.2. Motivación

En el contexto que se plantea surge la posibilidad de desviar ataques, en lugar de cortarlos, hacia lo que se conoce como *honeypots* y de los que se hablará a continuación.

1.2.1. Honeypots

Los *honeypots* son sistemas aislados en la red que actúan como señuelos de cara a posibles atacantes, evitando así que accedan a información crítica. Por este motivo, han de simular eficazmente un sistema productivo, pero carente de datos reales o sensibles, de manera que el atacante no sea consciente del engaño y tenga una falsa sensación de seguridad. Un *honeypot* está diseñado con el objetivo de retener al atacante el mayor tiempo posible, siendo éste uno de sus mayores atractivos, pues permite estudiar su comportamiento. Tal y como se ha mencionado anteriormente, resulta de vital importancia entender cómo piensa y actúa un *hacker* para diseñar mejores estrategias de defensa[1].
Existen diferentes tipos de *honeypots*. Si se atiende a su funcionamiento y ámbito de actuación, tradicionalmente han actuado de forma pasiva en el lado del servidor, esperando intrusiones y monitorizando los ataques que se produjeran. Un ejemplo de este tipo sería *Kippo*, un *honeypot* para SSH, que permite levantar puertos que esperen conexiones SSH de manera que los atacantes puedan acceder al sistema e interactuar con un sistema de ficheros ficticios, mientras toda esta actividad queda registrada[18].

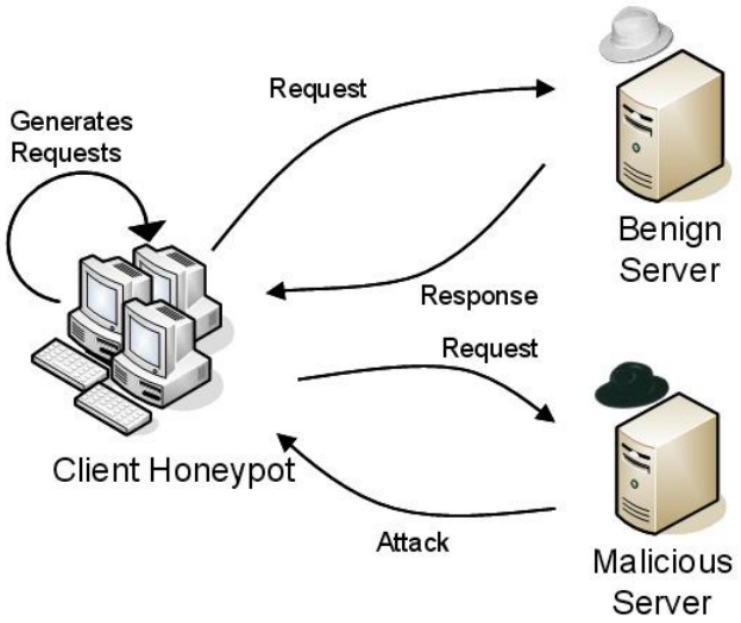

FIGURA 1.2: Funcionamiento de un cliente *honeypot* [18].

Frente a este modelo pasivo se encuentran los clientes *honeypot*[19], que se encuentran en el lado del cliente e investigan posibles ataques desde servidores de manera activa. Cuando se detecta una posible amenaza o intrusión, el sistema realiza una serie de peticiones para detectar servidores malignos, tal y como se ilustra en la figura 1.2.

Dentro de este tipo de *honeypots* se encuentra *Shelia*, que emula el comportamiento de un usuario que accediese a los enlaces que incluyen los correos de spam, detectando así qué servidores suponen un peligro para la seguridad del sistema[20].
Partiendo del concepto de *honeypot*, referido a una única máquina, es posible ampliarlo a todo una red, *honeynet*, de manera que sean varias máquinas las que reciban los ataques, simulando una red real.

1.2.2. ¿Cómo desarrollar un *honeynet*?

Con el concepto de *honeynet* en mente, teniendo presente lo expuesto hasta este punto y lo relativamente sencillo que resulta ser víctima de un ciberataque, se pretende desarrollar un sistema capaz de detectar una hipotética intrusión para, a continuación, desviarla a una red, que actuaría como *honeynet*. Con esta noción aparece un nuevo concepto: la detección. Y es que, para que un sistema sea seguro es de vital importancia que perciba un ataque en el caso de que éste llegue a producirse. A día de hoy existe ya este tipo de tecnología que implementan los denominados IDS (*Intrusion Detection System*) y de los que se hablará en el capítulo 2. Por otro lado, en lo que respecta a la *honeynet*, no resulta factible contar con todo un conjunto de máquinas físicas infrautilizadas o inactivas, dedicadas a simular una red real. Frente a esta opción, existen diversas soluciones de virtualización que permiten ejecutar varias máquinas virtuales sobre un mismo *hardware*(véase el capítulo 3). De esta manera, integrando la capacidad de detección de un IDS, junto con herramientas de virtualización, sería posible lograr un sistema con el comportamiento esperado en una *honeynet*.

1.3. Objetivos

El presente Trabajo de Fin de Grado tiene por objetivo desarrollar un sistema de desvío de intrusiones hacia *honeynets* que, en primer lugar, sea capaz de detectar una intrusión o ciberataque y, a continuación, emplee máquinas virtuales creadas dinámicamente para gestionarlos y desviarlos. Para ello, aunará la capacidad de detección de un IDS con tecnologías de virtualización de manera que el resultado sea un sistema integrado. Este sistema ha de tener en cuenta las limitaciones de capacidad del *hardware* empleados, así como el origen de los ataques, de forma que no se empleen nuevos recursos en ataques ya gestionados.
Por otro lado, teniendo en cuenta el estudio realizado en el contexto, que refleja las restricciones presupuestarias que existen en el ámbito de la seguridad, se buscarán soluciones gratuitas y de código abierto, tanto para el IDS, como para el *software* de virtualización.

1.3.1. Objetivos específicos

- Estudiar y comparar algunos de los IDS existentes y seleccionar uno para la implementación del sistema.
- Detectar de manera eficaz los siguientes ataques: escaneo de puertos[2], ataque de fuerza bruta contra un servicio y un ataque de denegación de servicio.
- Analizar y estudiar diversas tecnologías de virtualización y seleccionar una de ellas para implementar el sistema.
- Integrar el IDS con la tecnología de virtualización elegida.
- Evaluar el funcionamiento del sistema propuesto.

1.4. Metodología

Para realizar las tareas del presente trabajo se han establecido dos fases:

- **Fase de detección**: esta etapa inicial engloba todos aquellos aspectos relacionados con la detección de ataques e intrusiones, es decir, con el IDS. Incluye una serie de tareas de documentación e investigación, así como la configuración y elaboración de pruebas para los distintos IDS.
- **Fase de desvío**: esta segunda fase se centra en las tecnologías de virtualización incluyendo, al igual que en la primera, tanto tareas teóricas como prácticas. Así mismo, también se ha añadido a esta etapa todo lo referente a la integración entre las tecnologías del IDS y de virtualización. Por último, esta fase también comprende la configuración de otros servicios como *syslog* o *netfilter*.

1.5. Estructura del resto de la memoria

A continuación se indica el resto del contenido de los distintos capítulos de la memoria:

[2]No es un ataque en sí, sino un estudio del objetivo. No obstante, suele realizarse antes de cualquier otro ataque.

- **Capítulo 2 - Sistemas de Detección de Intrusiones**: se explica qué es un IDS, cuáles son sus características y qué requisitos se buscan para el empleado en este TFG. A continuación, se analizan dos de los más empleados, discutiéndose cuál es el más adecuado para el presente TFG.
- **Capítulo 3 - Virtualización**: de manera similar al capítulo 2, se describe en qué consiste la virtualización y se comparan distintas soluciones. A partir de esta comparativa se discute cuál es la idónea para el trabajo.
- **Capítulo 4 - Solución planteada**: en este capítulo se describe el funcionamiento del sistema propuesto, explicando los distintos componentes, así como su integración entre ellos.
- **Capítulo 5 - Resultados y evaluación del sistema**: se muestran los resultados obtenidos a partir de las pruebas realizadas con los IDS y se evalúa el funcionamiento del sistema completo.
- **Capítulo 6 - Conclusiones y líneas futuras de investigación**: en este último capítulo se extraen las conclusiones extraídas de la realización del TFG y se plantean investigaciones futuras que pudieran derivar de él.
- **Anexo A - Ficheros de configuración y *scripts* del sistema**: en este anexo se detallan las configuraciones de los distintos elementos empleados, así como los *scripts* que permiten conectarlos entre sí.

Capítulo 2

Sistemas de Detección de Intrusiones

En este capítulo se explica qué es un sistema de detección de intrusiones, se describen dos de las utilizados a día de hoy, para finalmente compararlos y discutir cuál de ellos se adapta mejor a los requisitos necesarios para la elaboración del TFG.

2.1. Qué son los IDS

Tal y como se ha visto en el capítulo anterior, la llegada de Internet y su extensión en todo tipo de ámbitos ha supuesto, además de un gran avance, la introducción de ciertos riesgos y vulnerabilidades, antes inexistentes. A día de hoy resulta inconcebible que una empresa de cierto tamaño no cuente con su propia red o utilice Internet para llevar a cabo gran parte de su actividad. Las facilidades que esta apertura al exterior puede proporcionar se contraponen con los ataques que estas organizaciones son susceptibles de sufrir. Pese a que existen soluciones que tratan de garantizar la seguridad y que únicamente los usuarios autorizados accedan a los recursos, dichas soluciones no resultan infalibles y dependen, además, de tareas de mantenimiento que están sujetas a fallos u olvidos. Es ahí donde entran en juego los IDS o sistemas de detección de intrusiones[21]. Una vez que el atacante ha traspasado las medidas de prevención resulta esencial detectarlo por las siguientes razones:

- Cuanto antes se localice la intrusión, antes se pueden tomar medidas al respecto y, por lo tanto, menor será el daño causado por el ataque.
- De la misma manera que un sistema de alarmas instalado en una casa puede disuadir a ladrones a la hora de perpetrar un robo, un IDS también puede frenar posibles ataques.
- La detección de ataques proporciona una gran cantidad de información sobre las estrategias empleadas por los atacantes, y contribuyen a solventar vulnerabilidades del sistema de prevención.

En un paso previo a la descripción de los sistemas de detección de intrusiones, a continuación se explican los riesgos y los distintos ataques que puede sufrir un sistema. Dejando al margen los virus, que junto con las intrusiones representan los ataques más comunes, podrían distinguirse distintos tipos de intrusos[1]:

- **Impostor**: en este caso el intruso entra en el sistema bajo la identidad de un usuario legitimo y hace uso de los recursos de dicha cuenta.
- **Usuario negligente**: un usuario legitimo del sistema utiliza de manera errónea o abusiva los recursos a los que tiene acceso, ya sean datos o programas.

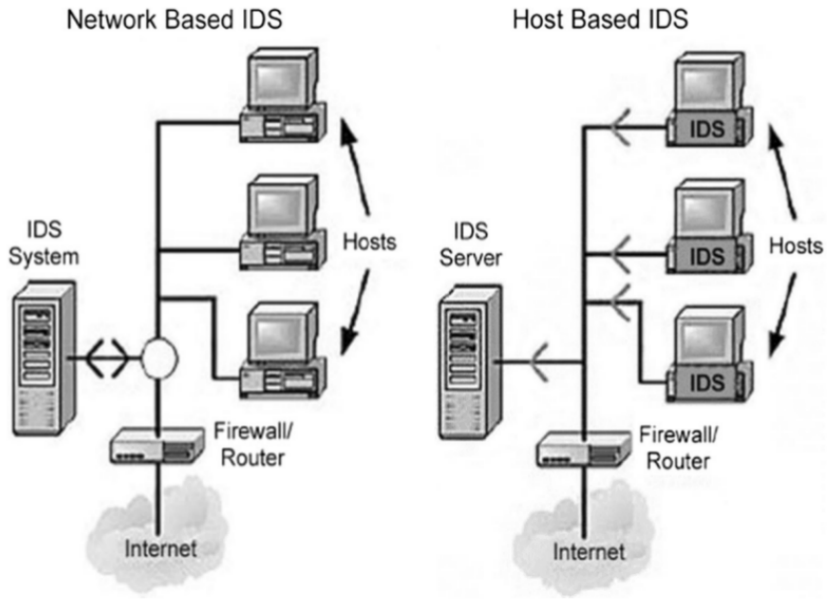

FIGURA 2.1: Comparación entre un HIDS y un NIDS [22].

- **Usuario clandestino**. Se trata de un atacante que intenta apropiarse de los privilegios del administrador o superusuario. Bajo la identidad del administrador el atacante puede, además, eludir controles de acceso o el registro de sus actividades.

2.1.1. Tipos de IDS

Existen dos criterios para clasificar los IDS: según dónde se coloquen o según la técnica que empleen. Atendiendo al punto de la red en que se encuentre, hay:

- **IDS basados en *host* o máquina (HIDS)**: se localizan en una única máquina, corriendo como aplicaciones, y analizan ficheros de log para detectar intrusiones.
- **IDS basados en red (NIDS)**: este tipo de IDS se encuentra en puntos concretos de la red, ya sea a la entrada o en el área desmilitarizada *DMZ*. En este caso se analiza todo el tráfico de la red y no solamente el que llega a cierta máquina.
- **IDS híbridos**: en este caso se combinan las dos modalidades previamente mencionadas.

Por otro, en función de la técnica que se utilice para detectar la intrusión, se encuentran los siguientes IDS:

- **Detección por uso inadecuado o basada en conocimiento**: se buscan trazas de ataques, patrones en el tráfico de red o registros en *logs* que puedan denotar un comportamiento sospechoso. Este tipo de sistema reconocería como sospechoso un número elevado de intentos fallidos de acceso.
- **Detección por firma**: en este caso, se monitorizan los paquetes de la red, comparándolos con patrones predeterminados y reconocidos, que se conocen como firmas. Ante la detección de un nuevo ataque, expertos de todo el mundo lo analizan y extraen características que se traducen en firmas y permitan su detección en futuras ocasiones. Este tipo de sistemas presenta como inconveniente

el lapso de tiempo que transcurre entre que se detecta un ataque y que la firma está disponible.

- **Detección por anomalía**: la instrusión es detectada cuando se identifica un comportamiento anómalo para determinado perfil o se superan los umbrales fijados según análisis estadísticos. Algunos ejemplos de los ataques que se pueden detectar serían suplantación de la identidad, si se diera el caso en que un usuario comenzara a realizar acciones inusuales o tratara de ganar privilegios de acceso cuando no suele ser lo habitual en dicho usuario. Otro ejemplo sería un ataque de denegación de servicio [23].

2.1.2. Características del IDS ideal

A la hora de utilizar un IDS es importante ser consciente de que se trata de una herramienta muy potente, pero pese a ello, presenta ciertas limitaciones. Debería ser capaz, sin embargo de:

- Registrar la actividad de los usuarios.
- Comprobar las alteraciones y modificaciones que se producen en ciertos datos.
- Actualizarse para responder ante los últimos ataques.
- Detectar si el sistema está sufriendo un ataque.
- Localizar errores de configuración.
- Facilitar la creación de políticas de seguridad por parte del administrador.
- Realizar gestión de la seguridad.

Además, debería presentar un diseño dedicado a infraestructura, lo que supone una ventaja para implantarlo en un sistema. No obstante, tal y como se ha mencionado, resulta importante ser consciente de ciertos inconvenientes, como la falta de opciones ante mecanismos de autenticación inseguros, la necesidad de una persona que investigue los posibles ataques detectados o los problemas que podría presentar a la hora de analizar el tráfico en una red, cuando este es muy elevado [23].

2.2. Requisitos del IDS buscado

Teniendo en cuenta que el propósito del presente trabajo es la creación de un sistema que, a partir de la detección de un ataque, sea capaz de desviarla a un entorno virtual, resulta crucial que la fase de detección sea eficaz. Es, por lo tanto, necesario establecer unos requisitos mínimos para el IDS elegido. Éste ha de ser:

- Capaz de detectar diversos tipos de ataques.
- Integrable con otros sistemas y servicios, a través de módulos de salida.
- Disponible para Unix.
- Open source

A partir de los dos últimos criterios y atendiendo a su popularidad en el mundo de la ciberseguridad, se han seleccionado *Snort* [12] y *Suricata* [24] como los IDS candidatos para llevar a cabo el trabajo.
A continuación en las siguientes secciones se describirán estos dos IDS para, posteriormente discutir la ideonidad de cada uno de ellos en lo referente a este TFG.

2.3. Sistemas de detección de intrusiones existentes

En los siguientes puntos se describirán la arquitectura y demás características de dos de los numerosos IDS existentes actualmente: Snort y Suricata.

2.3.1. Snort

Snort es un *software* gratuito y de código abierto desarrollado en 1998 por Martin Roesch. Desde su aparición, ha ido mejorando y extendiéndose, encontrándose consolidado como uno de los IDSs *open source* más potentes y eficaces. En la actualidad el proyecto se encuentra a cargo de la empresa *Sourcefire*, creada también por Martin Roesch, y que fue comprada por *Cisco* en 2013. Pese al desarrollo de una versión comercial, *Sourcefire 3D System*, Snort sigue adelante respaldado por una amplia comunidad.
En lo que respecta a las funcionalidades de Snort, es posible distinguir tres modos de funcionamiento distinto:

- ***Sniffer***: en este modo, Snort lee los paquetes que atraviesan la red y los muestra por pantalla.
- ***Logger* de paquetes**: loguea o registra los paquetes capturados en disco.
- ***Network Intrusion Detection System***: Cuando funciona en este modo, Snort lleva a cabo tareas de análisis y detección del tráfico de la red. Este modo ofrece multitud de opciones de configuración, lo que puede suponer cierta dificultad[25].

Arquitectura de Snort

Las distintas funcionalidades descritas anteriormente se implementan a través de una serie de *plugins* que presenta la arquitectura modular del sistema y puede verse en la figura 2.2. Snort consta de un núcleo o *core* con 4 módulos [26]:

- ***Sniffer* de paquetes**: este plugin permite escuchar el tráfico que viaja a través de la red, siendo incluso posible almacenar los paquetes capturados para su posterior lectura
- **Decodificador**: identifica los protocolos que encapsula el paquete desde el nivel de capa de enlace hasta los niveles TCP/IP. Los paquetes atraviesan por lo tanto una cascada de decodificadores hasta que su contenido queda guardado en estructuras de datos según sus correspondientes campos. De esta manera el contenido de los paquetes queda preparado para ser tratado por los preprocesadores.
- **Preprocesador**: el preprocesador de Snort engloba una serie de *plugins* o módulos encargados de diversas tareas que facilitan y aceleran la detección en el siguiente módulo (motor de detección) al realizarse el *matching* con las reglas.

Se puede variar el número de preprocesadores que los paquetes han de atravesar, variando con ello también el tiempo total de procesamiento, lo que resulta fundamental a la hora de determinar la eficacia de snort.
Entre las funciones que pueden realizar los preprocesadores se encuentran [25]:

- *Detección de anomalías.* Consiste en determinar si el contenido de un paquete se ajusta a lo que corresponde con los protocolos que lo encapsulan.
- *Agregación de sesiones TCP.* Este preprocesador recoge los datos de una sesión TCP, agrupándolos de manera que posteriormente sean evaluados y analizados en su conjunto. Esto se debe a que gran parte de los ataques suelen llegar en distintos fragmentos, de manera que serían indetectables si fuesen estudiados por separado.
- *Ensamblado de fragmentos IP.* De manera similar a lo que ocurría con las sesiones TCP, los paquetes IP pueden sufrir fragmentaciones debido a las limitaciones de la red, en concreto al MTU (*Maximum Transfer Unit*) que determina el tamaño máximo de un paquete para que pueda atravesar un enlace. De esta manera resulta posible que un ataque quede enmascarado en varios fragmentos y no genere ninguna alerta.
- *Detección de escaneo de puertos.* Resulta muy difícil detectar un escaneo de puertos haciendo uso únicamente de reglas, pues hay que tener en cuenta que para realizarlo se envían paquetes a distintos *hosts* y puertos, en conexiones distintas. Por otro lado, existen ciertos paquetes que no cumplen las especificaciones y denotan que se esta llevando a cabo este tipo de ataque. Es el caso de un paquete *NULL*.

- **Motor de detección**. Este módulo se encarga de procesar los paquetes procedentes del preprocesador. Para ello utiliza una serie de reglas contra las que analiza los paquetes. En caso de encajar, son enviados al procesador de alertas. El sistema de reglas empleado por Snort se basa en la detección mediante firmas o signatures. Este método se basa en comparar los datos de los paquetes, como cadenas, con ciertos patrones conocidos de ataques. A esta comparativa las reglas de Snort añaden la posiblidad de generar expresiones de manera que se produzca una coincidencia solo bajo determinadas circunstancias. Con todo ello, este sistema resulta muy rápido (gracias a la detección mediante patrones) y fiable, pues el hecho de fijar determinados parámetros permite reducir el número de falsos positivos (alertas generadas por paquetes que no constituyen ningún ataque).

- **Módulo de alertas y *logs***. Este componente se encarga de gestionar los paquetes que hayan coincidido con alguna regla. Existen multitud de posibilidades a la hora de tratar las alertas y logs generados. Por ejemplo, se pueden guardar las alertas en ficheros en máquinas remotas haciendo uso de sockets *UNIX* o mostrar la información referente a los *logs* en interfaces *web*. Todo ello requiere el empleo de *plugins* adicionales, al igual que ocurría con los preprocesadores.

Reglas de Snort

Las reglas de Snort constituyen la clave del sistema, encontrándose agrupadas en función del tipo de ataque que detectan. Existen ya numerosas reglas en el proyecto, que pueden añadirse a Snort con una simple directriz *include* en el fichero de configuración. Pese a ello, la sintaxis simple y descriptiva del lenguaje que emplean estas

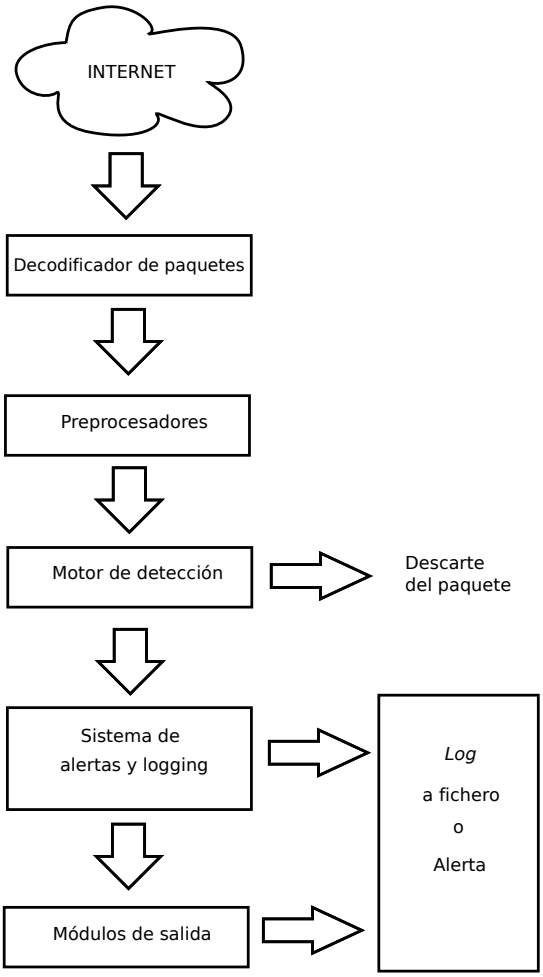

FIGURA 2.2: Arquitectura de Snort.

reglas permite que cualquier usuario pueda escribir sus propias reglas, una vez se haya familiarizado con esta.
Las reglas constan de dos partes [27]:

- ***Header* o cabecera**. Esta primera parte contiene la acción a realizar en caso de que la regla coincida, así como, las direcciones IP y puertos, tanto de origen como destino.
- **Opciones**. Esta segunda parte incluye el mensaje que ha de mostrarse en la alerta y otros parámetros adicionales, como las partes del paquete que han de evaluarse para determinar si se cumple la regla o no.

2.3.2. Suricata

Al igual que ocurre con *Snort*, *Suricata* se caracteriza por ser a su vez un sistema *open source*, de alto rendimiento, dirigido a la detección de intrusiones mediante el empleo de firmas de ataques ya conocidos. El proyecto *Suricata* surgió en 2009, impulsado por la OISF (*Open Information Security Foundation*), una organización sin ánimo de lucro fundada con el objetivo de crear lo que se conoce como IDS e IPS(*Intrusion Prevention System*) de próxima generación. Esta organización, que aglutina tanto desarrolladores como empresas expertas en ciberseguridad, trata de diseñar sistemas que satisfagan los requisitos que demanda la comunidad *open source* de seguridad.

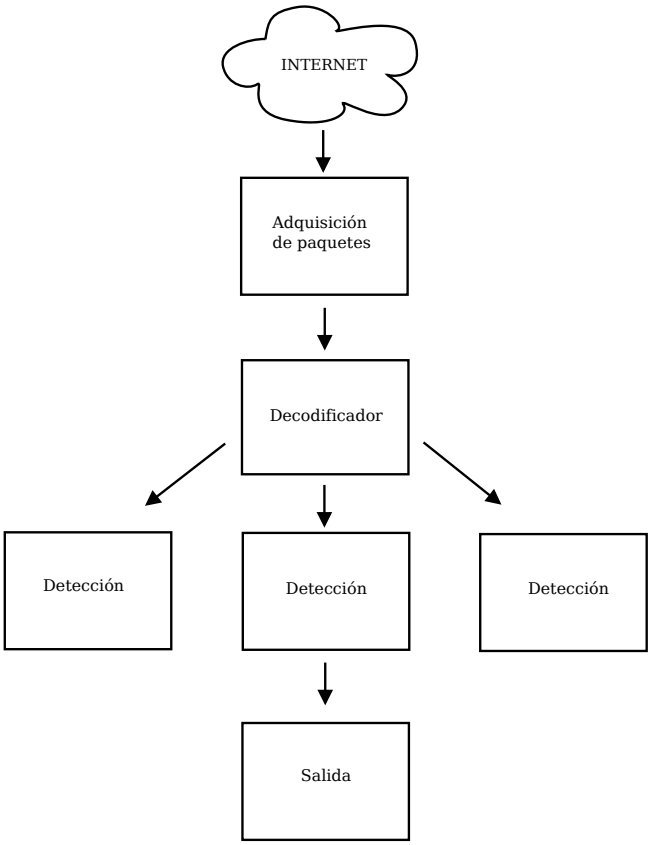

FIGURA 2.3: Arquitectura de Suricata.

Suricata presenta, por tanto, las funcionalidades de un IDS, un IPS y un motor de monitorización de red [24].
Entre las características de *Suricata* se encuentran [28]:

- Alto rendimiento y escalabilidad gracias a su arquitectura multi-hilo.
- Detección automática de protocolos de red.
- Seguimiento y reensamblado de segmentos TCP.
- Análisis de los protocolos HTTP, SMTP, DNS y TLS.
- Unificación de la salida en formato *JSON*.
- Integración con *Lua*, un lenguaje de programación que permite elaborar *scripts* para la detección.

Arquitectura de Suricata

La arquitectura de *Suricata* es bastante similar a la de *Snort*. Se trata de un sistema modular en el que los paquetes analizados atraviesan diferentes módulos en función del modo en que se esté ejecutando. El modo por defecto es el de detección, que es el que más recursos consume y se expone a continuación. Suricata presenta los siguientes módulos[29]:

- ***Packet Acquisition***: este módulo se encarga de capturar los paquetes procedentes de la interfaz de red y los envía al siguiente módulo para que puedan ser decodificados.

- ***Decoder***: En este módulo se determina el tipo de enlace (estando soportados los siguientes: LINKTYPE_LINUX_SLL, LINKTYPE_ETHERNET, LINKTYPE_PPP, LINKTYPE_RAW) y se procesan los paquetes de manera que sean compatibles con el resto de módulos.
- ***Stream module***: su principal cometido es identificar sesiones en el flujo de datos y reorganizar los paquetes de forma que su orden sea el correcto. Además de actuar a nivel de transporte como sobre el protocolo TCP, es capaz de tratar datos a nivel de aplicación, como por ejemplo tráfico HTTP.
- ***Detection module***: presenta varias tareas, de entre las que destaca la comprobación de las reglas y la generación de alertas en el caso de existan coincidencias. Las alertas son enviadas al módulo output.
- ***Output module***: la salida de Suricata suele presentarse al usuario en formato *JSON*, aunque también permite la integración con otros servicios como *syslog*.

Una de diferencias más importantes de la arquitectura de *Suricata* con respecto a la de *Snort* es el *multithreading*, lo que permite que la CPU procese varios paquetes de manera concurrente, es decir, en paralelo, tal y como puede verse en la figura 2.3.

Reglas de Suricata

Suricata permite hacer uso de reglas predefinidas, como las proporcionadas por *Talos* (antes VRT, *Vulnerability Research Team*), un grupo de expertos dedicado a la seguridad, que investiga nuevas amenzas y ataques en la red. Sin embargo, también presenta su propia sintaxis, muy similar a la de Snort, de forma que cualquiera pueda implementar sus propias reglas. Éstas presentan la siguiente estructura [24]:

- **Acción**. Determina qué ocurrirá cuando se identifique alguna firma concreta. Las posibles acciones son dejar pasar el paquete, descartarlo, rechazarlo con un mensaje al origen o generar una alerta.
- **Cabecera**. Indica el protocolo, origen y destino de la regla, así como su sentido.
- **Opciones**. Permite definir cómo se generará una cierta alerta, *meta-settings*, qué contenido hay que buscar en los paquetes, *payload keywords* o qué parametros han de encajar para un determinado protocolo, *HTTP keywords* o *DNS keywords*, entre otros.

2.4. Discusión

En la tabla 2.1 se exponen las diferencias más significativas entre los IDS descritos previamente. Tal y como se puede comprobar, ambas herramientas son muy similares, siendo el procesamiento multi-hilo de Suricata una de las diferencias más notables. Si bien es cierto que ante situaciones en las que la red se ve sometida a una carga de tráfico muy elevada, Suricata presenta un rendimiento más elevado y es más eficaz en el número de amenazas que es capaz de detectar, el largo recorrido de Snort y su documentación facilitan en gran medida su configuración. Esta circunstancia ha resultado de gran relevancia a la hora de seleccionar el IDS a emplear durante el proyecto, puesto que durante las pruebas que se llevaron a cabo con ambos sistemas (véase el capítulo 5), en todos los casos resultó más sencillo averiguar la configuración adecuada para un determinado ataque en Snort.

Por otro lado, las condiciones del entorno de prueba, donde no se producirían situaciones de congestión de red o tendría que analizarse gran cantidad de tráfico, redujeron la ventaja que aportaba Suricata en este aspecto.
Finalmente, otro factor que se ha tenido en cuenta para la elección del IDS ha sido la carga computacional y el consumo de recursos, pues el sistema ha sido implementado en un ordenador portátil con recursos limitados. En este sentido, la carga de CPU y memoria que requiere Suricata es considerablemente mayor que la de Snort.
En definitiva, teniendo en cuenta los requisitos expuestos al inicio del capítulo, tanto Snort como Suricata resultaban IDS válidos. No obstante, algunos factores relacionados con el entorno de implementación, junto con la documentación disponible, han decantado la elección en favor de Snort.

Característica	**Snort**	**Suricata**
Desarrollador	Sourcefire, Inc	OISF
Año de lanzamiento	1998	2009
Plataformas	Windows, MacOS, Unix	Windows, MacOS, Unix
IPS	opcional	opcional
Versión estable	2.9.11	4.0.4
Soporte IPv6	opcional con la opción –enable-ipv6	completamente soportado
Procesamiento	mono-hilo	multi-hilo
Licencia	GNU GPL v2	GNU GPL v2
Lenguaje	C	C
Escalabilidad	Medio	Alto
Reglas soportadas	VRT Emerging Threats rules SO rules	VRT Emerging Threats rules
Acelaración para la captura de tráfico	Extensión no incluida por defecto	Integración con CUDA y soporte de PF_RING
Output	Formato Unified2	Formato Unified2, YAML o JSON
Instalación	código fuente rmp	código fuente git apt-get(Ubuntu)
Uso de recursos CPU y memoria	Medio	Elevado
Documentación	Abundante: oficial foros	Escasa

Tabla 2.1: Comparativa entre Snort y Suricata.

Capítulo 3

Virtualización

En este capítulo se introducirá el concepto de virtualización y se presentarán algunas de las tecnologías disponibles a día de hoy para, finalmente compararlas y discutir cuál de ellas se adecua mejor a las necesidades del presente TFG.

3.1. ¿Qué es la virtualización?

Para la segunda parte del *proyecto* se hará uso de herramientas de virtualización, por lo que este punto estará dedicado a explicar en qué consiste. En primer lugar, el término virtualización hace referencia [30] a aquel *software*, por lo general sistemas operativos, que se ejecuta de manera concurrente y aislada con respecto a otros, sobre un único sistema *hardware*. La virtualización permite, por lo tanto, la compartición de los recursos *hardware*, incrementando su utilización y reduciendo el coste de operación.
Esta compartición de recursos es posible gracias a una capa *software* que se encuentra entre la máquina física, *host*, y los elementos virtuales, *guests*. Este *software* se denomina hipervisor y se encarga de gestionar y repartir los recursos del *hardware* entre los diferentes *guests*. Cada máquina virtual necesita su propio procesador y memoria que son, en realidad los recursos de la máquina física que han sido asignados por el hipervisor. Esta estructura puede verse en la figura 3.1.
En función de donde se sitúe, existen dos tipos de hipervisores:

- **Tipo 1, nativos o *bare-metal*** Se ejecuta directamente sobre el *hardware* de la máquina anfitriona. Éste es el hipervisor empleado por *VMware ESX* [31], *kvm* [32] o *Microsoft Hyper-V hypervisor* [33], entre otros.
- **Tipo 2 o embebidos** Al contrario que los anteriores, este tipo de hipervisores corren como una aplicación más sobre el sistema operativo del anfitrión. Algunos ejemplos son *VirtualBox* [34] o *VMware Workstation* [35].

Si se comparan ambos tipos, los primeros resultan más rápidos y eficientes, pues no tienen ninguna capa intermedia sobre el *hardware* que los ralentice. Además también resultan más seguros. Por el contrario, los hipervisores embebidos son más sencillos de configurar [36].

3.1.1. Orígenes de la virtualización

Los inicios de la virtualización se remontan a la década de 1960, en la búsqueda de la reducción de los costes que suponía el empleo de los recursos de un sistema computacional y aumentar el número de usuarios que podían acceder a ellos. Con este propósito, IBM desarrolló una solución de tiempo compartido, lo que redujo considerablemente los costes de uso que suponía un ordenador. No obstante, a pesar de

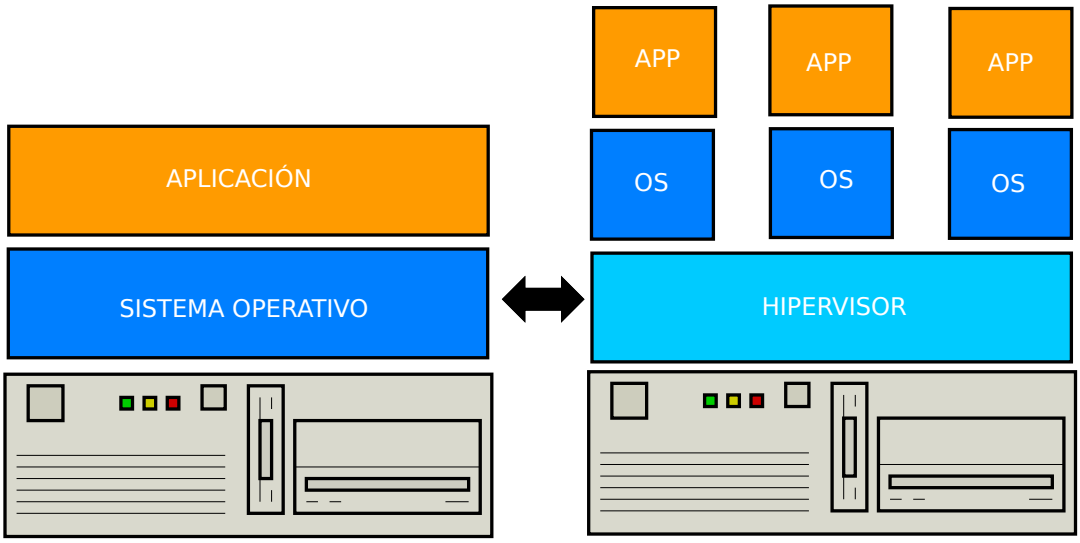

FIGURA 3.1: Esquema de una máquina con virtualización frente a una sin ella

la existencia la tecnología de los hipervisores, no fue hasta los años 1960 cuando se produjo el auge de la virtualización. Este despliegue fue impulsado con la introducción en las empresas de nuevos proveedores, lo que suponía en muchos casos incompatibilidades de *hardware* con aplicaciones de herederas. La solución pasaba por adquirir nuevos servidores (ya más baratos), específicos para cada proveedor, de manera que quedaban infrautilizados. Frente a este hecho, la virtualización proporcionaba una solución eficaz al permitir dividir los recursos de los servidores y ejecutar varios sistemas operativos [37].
A día de hoy han cambiando los usos de la virtualización, así como las tecnologías empleadas. No obstante, la idea principal de ejecutar múltiples sistemas independientes sobre un mismo entorno se mantiene [38].

3.1.2. Ventajas de la virtualización

En el punto anterior se han explicado algunos de los beneficios que impulsaron el uso de la virtualización. A continuación se profundizará en este aspecto.
La virtualización permite [39]:

- **Reducción de costes a largo plazo**. Aunque pueda suponer una inversión inicial, con el tiempo se produce un considerable ahorro en energía, mantenimiento y espacio. Además de prolongar el uso de *software* que haya quedado obsoleto.

- **Alto rendimiento**. Las tecnologías de virtualización actuales permiten alcanzar un rendimiento similar al que se obtendría al ejecutar el *software* sobre el propio *hardware*. De hecho, de acuerdo con el *SAP Sales and Distribution (SD) Standard Application Benchmark*, una districución *Red Hat Enterprise Linux 6.2* ejecutada sobre kvm, presentaba una eficiencia del 85 % comparada con el mismo sistema corriendo sobre el *hardware*.

- **Migración**. La migración consiste en trasladar una máquina virtual o *guest* de una máquina física a otra. Esto permite, a su vez, balancear la carga que

soportan los *hosts*, actualizarlos sin que el *guest* se vea afectado o la migración física de máquinas virtuales.

- **Recuperación de desastres**. Ante un problema o incidencia grave, la recuperación de la máquina virtual es mucho más rápida que una máquina física. Por otro lado, el hecho de que sean entornos aislados impide que el problema afecte a otros sistemas.

3.1.3. Tipos de virtualización

Aunque en los puntos anteriores se ha utilizado la virtualización de sistemas operativos principalmente para explicar cómo funciona, éste no es el único nivel de virtualización que puede encontrarse [40]:

- **Virtualización de escritorio**: consiste en la separación del escritorio, así como los programas y datos de usuario, de la máquina física en la que se ejecuta. Este escritorio virtual es almacenado en un servidor central que puede ser accedido remotamente. Un ejemplo de este caso es VMware Horizon [41].
- **Virtualización *hardware* o de servidor**: permite la abstracción de los recursos *hardware* de una máquina para el *software* que lo utiliza. Dentro de la virtualización *hardware* se distinguen los siguientes tipos:
  - *Virtualización completa*. Emplea el sistema operativo del *guest* inalterado. El *guest* se comunica directamente con la CPU del *host* a través de un canal creado por el hipervisor.
  - *Paravirtualización*. Requiere una versión modificada del sistema operativo del *guest*, que utiliza el hipervisor como intermediario para realizar llamadas a la CPU.
  - *Virtualización software o emulación*. El hipervisor traduce las llamadas desde el *guest* a un formato comprensible por el sistema del *host*. Toda esta traducción supone tiempos mayores.
- **Virtualización software**: en este tipo de virtualización, el *host* crea una plataforma con todos los requisitos y funcionalidades necesarios para que el *virtualizado* funcione. Con esta tecnología se pueden solucionar problemas de compatibilidad y rendimiento. Pueden distinguirse tres tipos distintos de virtualización *software*: de aplicación, de sistema operativo y de servicio.
- **Virtualización de almacenamiento**: permite la unión de multitud de dispositivos de almacenamiento bajo la apariencia de un único dispositivo. Supone una mejora de la gestión de los recursos de almacenamiento, así como de su usabilidad y flexibilidad.
- **Virtualización de red**: asociado al concepto SDN (*Software Design Networks*) consiste en trasladar las funcionalidades que tradicionalmente han residido en elementos *hardware*, como *switches* o *routers* a componentes lógicos que puedan ser ejecutados sobre *hardware* genérico. Entre las ventajas que proporciona esta virtualización se encuentran un aprovisionamiento más rápido, automatización del mantenimiento y soporte o implementación no disruptiva.

Para la realización de la segunda parte del *proyecto*, donde será necesario deplegar de manera dinámica una red de máquinas virtuales se contemplarán a continuación algunas de las muchas soluciones existentes.

3.2. Sistemas y tecnologías de virtualización

A lo largo del siguiente punto se estudiarán algunas de las diferentes tecnologías de virtualización que es posible encontrar a día de hoy, para poder compararlas y elegir una para la realización del trabajo.

3.2.1. KVM Kernel Virtual Machine

KVM se trata de una solución de virtualización *hardware* completa para arquitecturas x86 y amd64. Es de código abierto y se distribuye bajo la licencia GNU GPL. Fue creado en el año 2006 por *Avi Kivity* y publicado como parte del *kernel* de Linux 2.6.2. Cuando fue introducido, *Xen* era el hipervisor por excelencia, utilizaba un *kernel* modificado del *guest* para arrancar máquinas virtuales y limitaba el *kernel* del *host* a la gestión de los elementos de entrada y salida. Por el contrario, la llegada de *KVM* coincidió con la introducción de las extensiones de virtualización para la CPU, por parte de Intel y de AMD. KVM fue desarrollado con el propósito de actuar como *driver* que gestionara las instrucciones de virtualización expuestas por el hardware y, de esta manera, aprovechar al máximo la funcionalidad ya existente de Linux. Las actualizaciones y mejoras introducidas en el kernel podían ser aprovechadas también por KVM. En resumen, la principal diferencia entre *Xen* y KVM es que el primero es un hipervisor externo, mientras que el segundo forma parte del kernel de Linux [42].

Componentes de KVM

La funcionalidad completa de KVM se consigue gracias a la integración de distintas partes [32][42]:

- **kvm.ko**: se trata de un módulo del kernel cargable. Se encarga de proporcionar las funcionalidades de KVM que son independientes de la arquitectura.
- **kvm-intel.ko/kvm-amd.ko**: son los módulos específicos de procesador.
- **Virtualizador**: parte del espacio de usuario encargada de simular el *hardware* de la máquina virtual sobre el que corre el sistema operativo del *guest*. Para esta función se emplea QEMU [43], un emulador que recrea CPU mediante traducción binaria[1], pero que también es capaz de funcionar como virtualizador. KVM emplea la siguiente lógica: al ejecutar código nativo de la CPU del *host* se utilizan las llamadas al sistema del módulo del kernel, mientras que para el resto, cuando se requieren *drivers* de I/O por ejemplo, se emplea QEMU.
- ***Guest additons***: se trata de software encargado de mejorar la integración entre el *guest* y la máquina anfitriona.

Características de KVM

Algunas de las principales características que ofrece KVM son las siguientes [44]:

- **QMP**: *QEMU Machine Protocol* es un protocolo basado en *JSON* que permite controlar una instancia QEMU.
- **Canal VM**: permite la creación de un canal de comunicación entre el *host* y los *guests*.

[1] La traducción binaria consiste en traducir las instrucciones del kernel de la máquina *guest* de manera que puedan ejecutarse en el *host*

- **Migración**: es posible migrar una máquina virtual, tanto de forma *offline*, estando ésta apagada, como *live*, encendida. Esto permite realizar mantenimiento o actualizaciones, sin alterar el servicio.
- ***Nested Guests* o *Guests* anidados**: consiste en ejecutar un *guest* KVM sobre otro *guest*. Se trata de una característica experimental.
- ***Virtio Devices***: marco común para la virtualización de los dispositivos de entrada y salida. En el *host* se implementa a través de QEMU.
- **ABI de guest estable**: permite a las máquinas virtuales emplear el mismo ABI (*Application Binary Interface*) de cara a actualizaciones del QEMU.

3.2.2. Docker

Docker es una plataforma que permite la creación de contenedores Linux para desarrollar, desplegar y ejecutar aplicaciones. Aunque el proyecto Docker surgió dentro de *dotCLoud* de la mano de *Salomón Hykes*, en el año 2013 fue liberado como un proyecto de código abierto. Hoy en día el proyecto es mantenido por *Docker Inc.* y en él colaboran empresas como *Red Hat, IBM, Cisco Systems o Google*. A pesar de que los contenedores Linux aparecieron mucho antes, Docker introduce un nuevo concepto de uso al emplearlos para el desarrollo de aplicaciones. Aunque conviene recordar que un contenedor proporciona principalmente aislamiento de procesos y no presenta las características de una máquina completa, los contenedores *Docker* podrían verse como máquinas virtuales compuestas de módulos y muy ligeras. Por otro lado, Docker permite además reducir el tiempo que hay normalmente entre que se escribe el código de una aplicación hasta que se corre en producción.
Para la creación de los contenedores Docker emplea una plantilla con instrucciones, que es lo que se conoce como *imagen*. Cada nueva instrucción supone una nueva capa, de manera que al construir una imagen, si ya había una creada, solamente se aplican los cambios de las capas modificadas [45]. Es este aspecto lo que proporciona la ligereza a Docker frente a otras tecnologías, como LXC (*Linux Container*) [46].

Arquitectura de Docker

Docker presenta una arquitectura cliente-servidor, de manera que el cliente se comunica con el servidor y es éste el que se encarga de levantar y ejecutar los contenedores tal y como puede verse en la figura 3.2 [47].
Docker consta de los siguiente componentes:

- **Servidor Docker (*dockerd*)**: servicio que se encuentra escuchando llamadas a la API de Docker y, a partir de éstas gestiona imágenes, contenedores y volúmenes.
- **Cliente Docker (*docker*)**: cliente por línea de comandos que dirige directrices hacia el servidor por medio de una API REST, que proporciona una define una serie de interfaces sobre *sockets* Unix o interfaces de red.
- **Registros**: constituyen otro elemento clave de la plataforma pues almacenan imágenes que pueden ser utilizadas directamente o por otros usuarios o como base para nuevos contenedores. *Docker Hub Docker Cloud* son repositorios públicos y, de hecho, el primero de ellos se encuentra configurado por defecto para la búsqueda de imágenes. Existen, no obstante, repositorios privados.

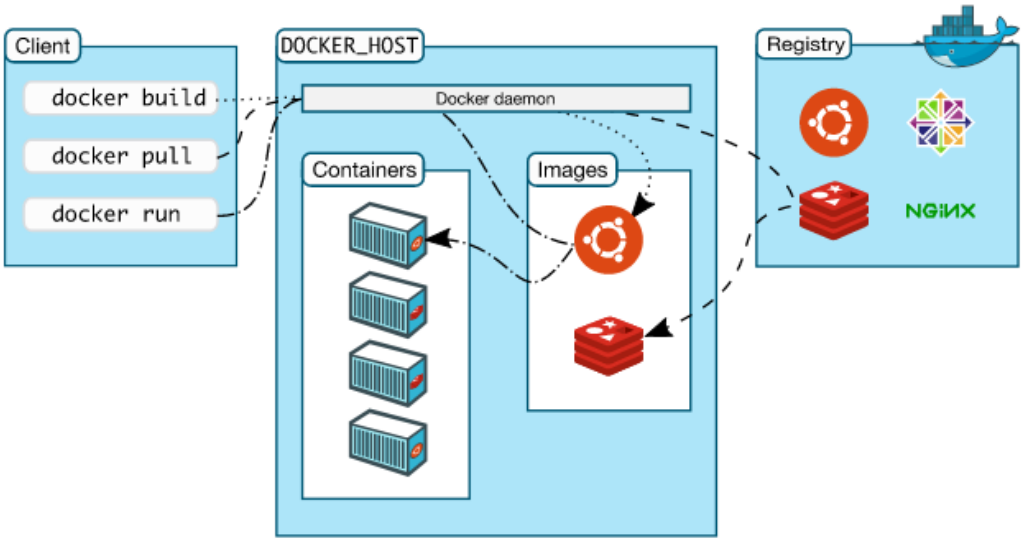

FIGURA 3.2: Arquitectura de Docker [48].

¿Qué aportan los contenedores?

El uso de contenedores Docker supone una serie de beneficios [45]:

- **Modularidad**: Docker proporciona una visión de la aplicación como microservicios, de manera que esta se puede dividir, facilitando cambios o actualizaciones.
- **Capas y control de versión de imagen**: tal y como se ha explicado en el punto anterior las capas que configuran las imágenes permiten su reutilización. Las capas incluyen de manera implícita un registro de cambios o versiones.
- **Restauración**: es posible recuperar versiones anteriores de las imágenes en caso, por ejemplo, de que se produzca un error o la imagen resultante no fuera la deseada.
- **Implementación rápida**: al abordar el proceso de desarrollo desde una perspectiva más granular, que permite construir y desechar contenedores de manera sencilla, Docker reduce considerablemente los tiempos de implementación[2].

3.2.3. OpenStack

OpenStack es una plataforma que integra una serie de herramientas *open source* para la gestión de todo tipo de recursos para las denominadas *clouds*, tanto públicas como privadas. Es preciso realizar un inciso para aclarar el concepto *cloud computing* o computación en la nube: este término hace referencia a un paradigma de acceso a los recursos de computación, tales como memoria o capacidad de proceso. Siguiendo este principio, los servidores se agrupan de manera que su capacidad conjunta sea utilizada por los usuarios finales de forma transparente y remota. Generalmente, se distinguen tres tipos de servicio que pueden ser ofrecidos a través de *cloud computing* [49]:

[2]Un sistema podría estar formado por varios contenedores, donde cada uno de ellos ejecutara un proceso e implementase una funcionalidad. Si, por ejemplo, hubiera que realizar cambios en una de estas funcionalidades, únicamente sería necesario reconstruir el contenedor en cuestión y no todo el sistema.

- ***Software as a Service* (SaaS)**: en este caso se ofrece la utilización de software que se encuentra corriendo sobre alguna plataforma e infraestructura.
- ***Platform as a Service* (PaaS)**: plataforma de desarrollo y despliegue de aplicaciones para desarrolladores. Suele incluir *software* de infraestructura, bases de datos, *middleware* y herramientas de desarrollo.
- ***Infrastructure as a Service* (IaaS)**: ofrece *hardware* (servidores, almacenamiento y redes), así como tecnologías de virtualización de sistemas operativos.

OpenStack surge en 2010, desarrollado conjuntamente por *Rackspace Hosting* y NASA, aunque desde el año 2016 es mantenido por *OpenStack Foundation*, una organización sin ánimo de lucro creada para promover el *software* de OpenStack, así como su comunidad. Como proyecto *Open Source*, múltiples usuarios y compañías colaboran para desarrollar nuevos componentes. OpenStack dispone de un conjunto de API que potencian la abstración que proporciona de por sí la virtualización y permite la creación de *clouds* que cumplen con el estándar establecido por el *NIST*, según el cual ésta ha de tener: red, agrupación de recursos, interfaz de usuario, proporcionar capacidades y asignar recursos de manera automática [50].

Componentes de OpenStack

Aunque existen multitud de elementos dentro de la plataforma gracias a las contribuciones de la comunidad, cabe mencionar los siguientes [50][51], que también se muestran en la figura 3.3:

- **Nova**: constituye el núcleo de OpenStack, encargándose de la creación y gestión de las instancias de máquinas virtuales.
- **Neutron**: es el encargado de las redes que interconectan los instancias y demás servicios. Garantiza que éstos se comuniquen de manera rápida y eficaz.
- ***Swift***: permite almacenar objetos y ficheros, localizables por medio de un identificador. Aporta escalabilidad y alta tolerancia a fallos.
- **Cinder**: proporciona almacenamiento de bloques de manera más tradicional para aquellos escenarios en los que el acceso a los datos sea de gran relevancia.
- **Keystone**: autentica y autoriza todos los servicios en OpenStack. Soporta distintos métodos de acceso, por lo que los usuarios no tienen que modificar los que usaban anteriormente.
- **Glance**: guarda imágenes de máquinas virtuales, siendo éstas copias de los discos duros.
- **Horizon**: se trata de una interfaz gráfica web, como alternativa a las APIs de OpenStack. Proporciona una visión global de todo el entorno.

3.2.4. OpenNebula

Una plataforma similar a OpenStack es OpenNebula, que al igual que ésta se encuentra en el ámbito del *cloud computing*, principalmente dentro del marco IaaS. Es también un proyecto de código libre que surgió en el año 2005 como un proyecto de investigación. Sus creadores, Ignacio M. Llorente y Rubén S. Montero buscaban

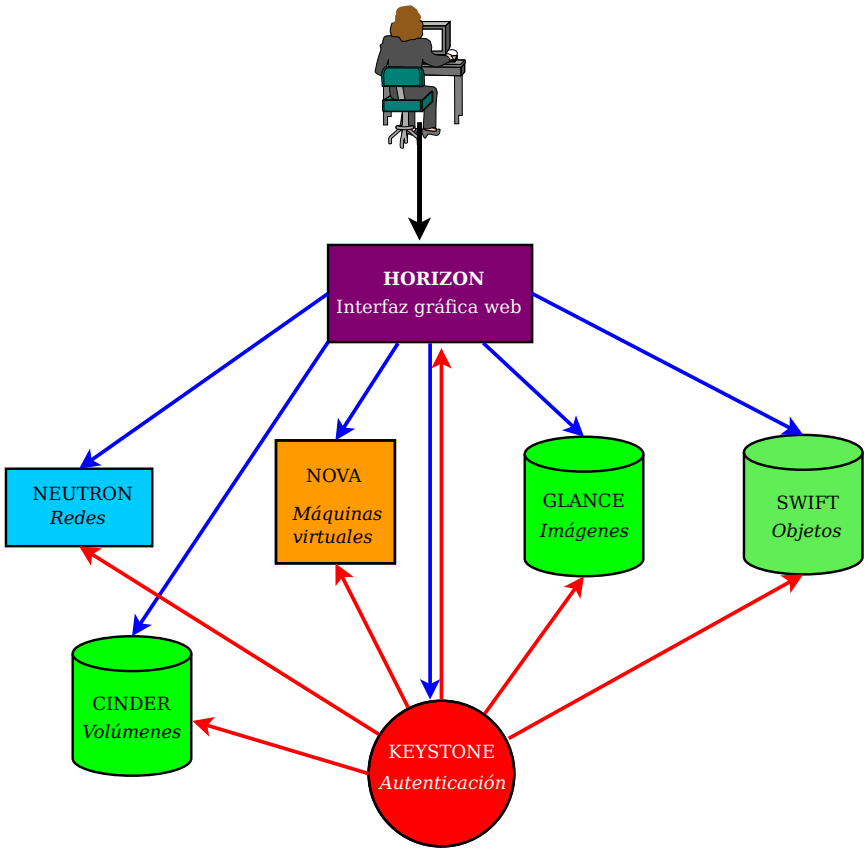

FIGURA 3.3: Arquitectura de OpenStack.

la manera de crear máquinas virtuales sobre infraestructuras distribuidas de manera eficiente y escalable. Ante la creciente adopción de OpenNebula, sus desarrolladores crearon la empresa OpenNebula Systems que proporciona servicios de valor añadido para los usuarios.
OpenNebula integra tecnologías de almacenamiento, red, virtualización, monitorización y seguridad para permitir el desarrollo y despliegue de infraestructuras en la nube. Proporciona varias interfaces *cloud*, como *Amazon EC2 Query* u *Open Cloud Computing Interface*, además de la opción de emplear distintos hipervisores: Xen, KVM o *VMware* [52].

Características de OpenNebula

La solución que ofrece la plataforma OpenNebula incluye las siguientes características [53]:

- **Interfaces para usuarios de cloud**: proporciona tanto API estandarizadas, como interfaces gráficas de fácil empleo para usuarios con menos conocimientos.
- **Gestión de máquinas virtuales**: permite la gestión del ciclo de vida de las VMs, el empleo de catálogos de imágenes de disco, así como la creación de plantillas para la definición de nuevas máquinas virtuales.
- **Gestión de redes virtuales**: incluye capacidades de virtualización de redes, soporte de IPv6 o routers virtuales, entre otros.
- **Configuración y control de aplicaciones**: es posible automatizar la instalación de aplicaciones, así como seguir su estado.

- ***Multi-tenancy* y seguridad**: permite la definición de roles y usuarios, así como listas de acceso de grano fino. Gracias a ésta última propiedad es posible establecer un mayor control sobre las operaciones que determinados usuarios pueden o no realizar. También resulta interesante el registro de *logs* de las actividades de usuarios que guarda.
- **Independiente de plataforma**: existen paquetes para la mayoría de distribuciones Linux.
- **Personalizable e integrable**: cuenta con un diseño modular que facilita su despliegue en cualquier *datacenter* ya existente. También permite la modificación de los drivers de sus componentes.
- **Instalación y actualización**: proporciona las capacidades de computación, almacenamiento y red a partir de una sola instalación. Ofrece, además, estabilidad a largo plazo gracias a las publicaciones de parches y actualizaciones.
- **Soporte**: la comunidad de OpenNebula aporta soporte *best-effort*.

Arquitectura de OpenNebula

Desde una perspectiva global, los elementos de OpenNebula se pueden dividir en tres categorías: virtualización, almacenamiento y redes. En cuanto a funcionalidad, se distinguen dos partes en la arquitectura de OpenNebula: el *front-end* y el *back-end*. En el *front-end* se alojan los servicios de OpenNebula: servidor de gestión de OpenNebula, planificador, MySQL, GUIs, APIs y servicios adicionales. Se comunica con los hipervisores a través de la red de servicio, y se encarga de arrancar máquinas virtuales y realizar otras operaciones de almacenamiento, así como de monitorizar los hipervisores y las máquinas virtuales. Desde ahí se gestionan todos los recursos del *cloud*. El *back-end*, por su parte se encarga tanto del almacenamiento de las instancias de máquinas virtuales que se encuentran en ejecución, como de las imágenes. Se conecta con los hipervisores a través de la red de almacenamiento, que en ocasiones puede ser la misma que la de servicio, tal y como se muestra en la figura 3.4. Por otro, lado además de las redes de servicio y almacenamiento, se puede hablar de red privada, encargada de interconectar las máquinas virtuales mediante VLANs, *Virtual Local Area Networks*, y de red pública pues en algunos casos resulta necesario acceder a recursos de Internet [54].

3.2.5. AWS (*Amazon Web Services*)

Otro proveedor de IaaS es Amazon Web Services, que desde el año 2006 ofrece servicios de computación, almacenamiento, redes y bases de datos para distintos negocios. Al contrario que las plataformas estudiadas anteriormente, AWS no es de código abierto o gratuita, sino que ofrece todos estos servicios bajo demanda, cobrando únicamente por lo se utiliza. Para ofrecer todo ello, cuenta con una infraestructura a nivel global donde los recursos se dividen en Regiones AWS. A su vez, cada Región AWS engloba una serie de Zonas de Disponibilidad con centros de datos en los que se ubican los recursos físicos replicados para garantizar redundancia y tolerancia a fallos. Las regiones se encuentran totalmente separadas, mientras que las zonas dentro de una misma región se encuentran conectadas mediante enlaces.
Son muchos los servicios que se pueden encontrar en AWS, además de los mencionados

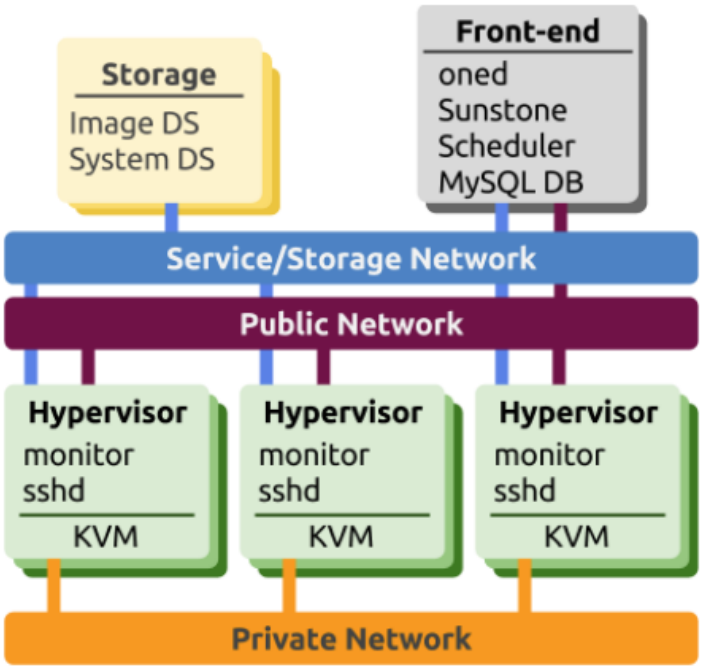

FIGURA 3.4: Arquitectura de OpenNebula [55].

anteriormente, y pertenecientes a diversos ámbitos: análisis, servicios móviles, mensajería, inteligencia artifical, productividad empresarial, *Internet of Things* o desarrollo de videojuegos *aws*.

Arquitectura de AWS

La imagen que se muestra a continuación muestra la arquitectura de AWS, en la que pueden distinguirse los siguientes elementos [56]:

- **EC2 (*Elastic Compute Cloud*)**: permite desplegar y arrancar máquinas virtuales personalizadas, con permisos de administrador y de manera escalable, conforme a la demanda.
- **ELB (*Elastic Load Balancer*)**: al y como su nombre indica se trata de un balanceador que permite distribuir el tráfico entrante entre las máquinas virtuales disponibles. También aporta la tolerancia a fallos.
- **AS (*Auto Scaling*)**: levanta o elimina automáticamente máquinas virtuales en función de la demanda y según la configuración establecida.
- **EBS (*Elastic Block Storage*)**: proporciona almacenamiento de bloques para los volúmenes empleados por las instancias de las máquinas virtuales. Se trata de un almacenamiento persistente.
- **S3 (*Simple Service Storage*)**: almacena tanto datos, como imágenes y *backups* o copias.
- **RDS (*Relational Database Service*)** Este componente permite la creación de instancias de bases de datos relacionales, así como la escala flexible de los recursos asociados a ellas.

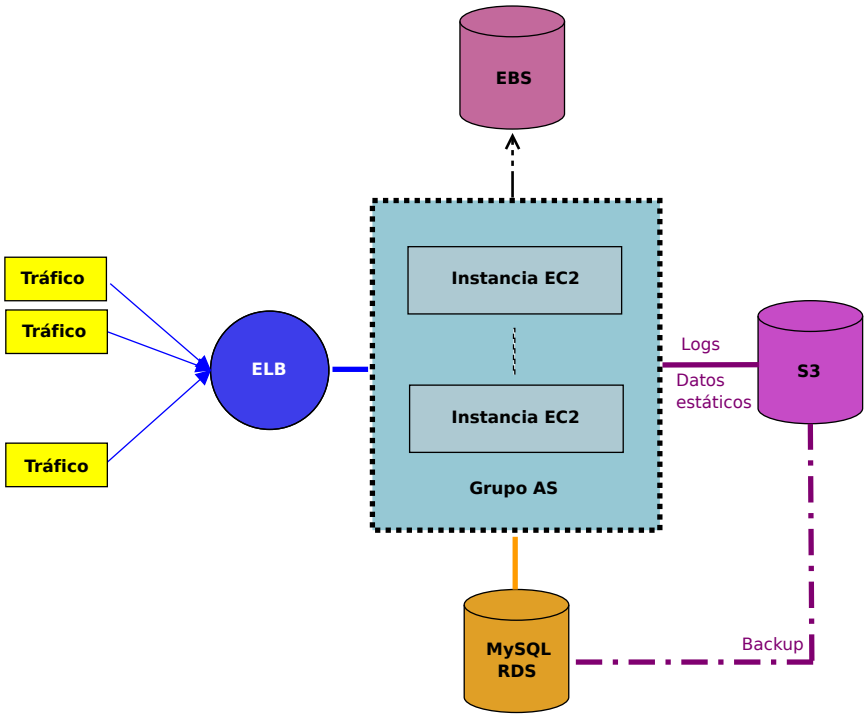

FIGURA 3.5: Arquitectura de AWS

3.3. Discusión

A lo largo de este capítulo se han presentado diversas tecnologías de virtualización candidatas para ser utilizadas en la implementación del *proyecto*. Todas ellas ofrecen diferentes capacidades y, además, constituyen distintos niveles de virtualización. En esta sección se discutirán las ventajas que representan cada una de ellas frente al resto, para finalmente seleccionar una de ellas.
En primer lugar, Docker constituye una herramienta de enorme potencial con la que resulta muy sencillo crear instancias virtuales aisladas. No obstante, el propósito de estos contenedores es el de ejecutar un único proceso, y no todos los que habrían en una máquina completa. Teniendo en cuenta que uno de los propósitos del trabajo es la creación de una máquina a modo de señuelo, esta solución no resulta factible.
Aunque, tal y como se comentaba al inicio de la sección, resulta complicado equiparar tecnologías tan heterogéneas, sí que es posible comparar las que se ubican dentro del marco del *cloud computing*. Véase la tabla 3.1. A modo de referencia se incluyen también las características de KVM.

AWS, por su parte, ofrece numerosos servicios de gran interés, así como una gestión de máquinas virtuales muy simple y fácil de realizar. Sin embargo, la principal desventaja de AWS es que no se trata de código abierto y es necesario pagar por sus servicios. Si bien es cierto que ofrece una determinada capacidad de cómputo de manera gratuita, no resulta viable pues uno de los objetivos del proyecto era la búsqueda de una solución económica y, frente a las opciones *open source* estudiadas, AWS queda descartado.
OpenStack y OpenNebula, las otras plataformas de *cloud*, son soluciones muy similares, destinadas a la creación de entornos en los que la escalabilidad resulta fundamental. Comparándolas entre sí, el elevado número de componentes de OpenStack, así como la terminología que emplea, incrementa su complejidad frente a OpenNebula.

Característica	**AWS**	**OpenStack**	**OpenNebula**	**KVM**
Año de lanzamiento	2006	2010	2005	2006
Desarrollador	Amazon Web Services, Inc	OpenStack Foundation	OpenNebula Systems	Avi Kivity
Tipo de *cloud*	pública	pública, privada e híbrida	pública y privada	-
Licencia *open source*	-	Apache v2.0	Apache v2.0	GNU GPL
Modelo de negocio	Pago por uso	Gratuito	Gratuito	Gratuito
Acceso	*Console* y *cli*	Horizon y *cli*	Sunstone y *cli*	*cli* y *virt-manager*
Hipervisores	Nitro, basado en KVM	Xen, XCP, QEMU, KVM, VMWare, ESXi, UML	Xen, KVM, VMWare ESX, ESXi	-
SS.OO soportados	Mayoría de distribuciones Linux	Mayoría de distribuciones Linux	Mayoría de distribuciones Linux	Mayoría de distribuciones Linux
Modelo de red	Plano con DHCP VLAN IPs elásticas	Plano Plano con DHCP VLAN	VLAN	Plano con DHCP
Propósito principal	Oferta de servicios de *cloud*	Crear nubes púbicas y privadas	Oferta de servicios de *cloud* Gestión de recursos	*Driver* del *kernel* de Linux

Tabla 3.1: Comparativa entre las soluciones de IaaS: AWS, OpenStack y OpenNebula. También se incluye KVM a modo de referencia.

Por otro lado ambas opciones hacen uso de KVM, añadiendo capas de abstracción para la gestión de recursos.
Dado que, como se comento anteriormente, el *hardware* empleado será un ordenador portátil, resulta importante tratar de reducir al máximo tanto la capacidad de cómputo, como de memoria utilizados, lo que se traduce también en una reducción del número de elementos empleados. KVM cuenta con herramientas de gestión por línea de comandos, como *virsh* que permiten la administración de máquinas virtuales, por lo que no es necesario añadir las capas que proporcionan las tecnologías de *cloud computing*.
Por todos estos motivos, se ha decidido que la mejor opción de las evaluadas para creación de máquinas virtuales durante la segunda parte del proyecto es KVM.

Capítulo 4

Solución planteada

A lo largo de este capítulo se expondrá la solución planteada ante los ataques que se producen con frecuencia en la red, cómo detectarlos, disminuir su impacto desviándolos hacia máquinas virtuales e incluso sacar partido de ellos.
En el sistema desarrollado se distinguen claramente dos etapas:

- **Etapa de detección**: durante esta fase, el sistema ha de ser capaz de determinar que la red esta sufriendo un ataque, identificarlo y reportarlo para que tenga lugar la siguiente fase de desvío.
  Para esta primera etapa se empleará principalmente Snort, descrito en el capítulo 2.

- **Etapa de desvío**: en este punto, una vez detectada la intrusión, el sistema ha de ser capaz de redireccionar todo el tráfico procedente del atacante hacia una máquina virtual que habrá creado dinámicamente y de forma transparente para el atacante.
  En esta segunda fase se explotará la tecnología de KVM (véase el capítulo 3).

4.1. Arquitectura del sistema

Para explicar la arquitectura del sistema, es necesario conocer dónde se ubica dicho sistema en relación a la totalidad de Internet, así como los elementos y dispositivos que pueden encontrarse en la red. Internet puede definirse como la red de redes, es decir, un conjunto de redes interconectadas. Teniendo esto en cuenta, la red de la organización o institución a proteger por el sistema, red interna o *intranet*, sería una de estas redes donde cada máquina podría tener conectividad con el resto de máquinas de Internet. Esta conectividad podría ocurrir también en sentido inverso, de manera que cualquier punto de la red podría llegar a acceder a la red interna e, incluso, atacarla.

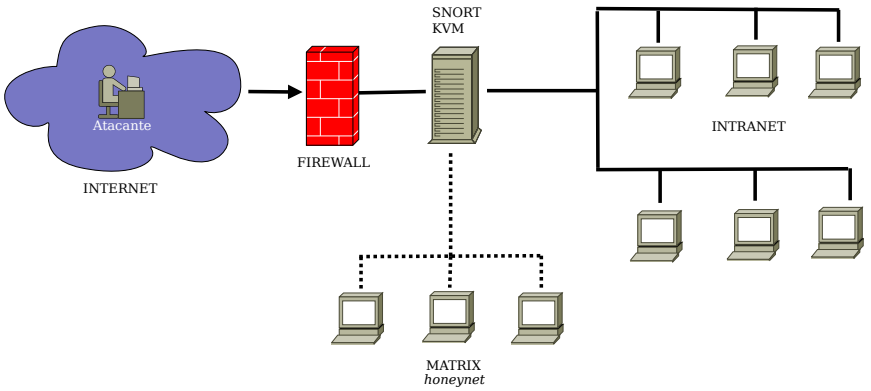

FIGURA 4.1: Arquitectura del sistema propuesto.

No obstante, existe aún un elemento intermedio que puede colocarse a la entrada de la *intranet*, filtrando y bloqueando determinadas conexiones en función de diferentes criterios. Se trata del conocido *firewall* o cortafuegos.
De acuerdo con lo anterior, la figura 4.1 muestra a grandes rasgos donde actuaría el sistema. La nube azul representa todo el conjunto de Internet, desde donde podría generarse un ataque. A continuación se encuentra el *firewall* e, inmediatamente después, ante la *intranet*, se localiza el sistema. Es necesario colocar el sistema y, concretamente Snort, detrás del cortafuegos para que no se generen alarmas a partir de posibles amenazas que ya filtra éste.

Hay que tener en cuenta que la figura se refiere a un entorno real, donde están presentes todos los elementos *hardware*, ordenadores de usuario, servidores, *routers*, etc. En este caso las dos etapas de las que se habló al inicio del capítulo y sus consecuentes componentes se englobarían en un mismo equipo. Teniendo en cuenta las exigencias de computación que requiere Snort para procesar paquetes, así como las de las instancias virtuales que creará KVM, esta máquina ha de ser un servidor de prestaciones relativamente altas. Este elemento se colorará inmediatamente después del *firewall* o cortafuegos, si lo hubiera, y justo antes del resto de la red e inspeccionará todo el tráfico dirigido hacia ésta.

En caso de producirse un ataque, éste podría proceder de cualquier punto de la red pero estaría identificado por una dirección IP origen. Si dicho ataque consiguiera atravesar el cortafuegos y fuese detectado por Snort (primera etapa), KVM crearía una nueva máquina virtual ubicada en una red también virtual, denominada *Matrix*. Para que el sistema sea realmente efectivo y cumpla con su objetivo, es necesario que los nuevos paquetes que lleguen al sistema, cuya IP origen coincida con la que generó el ataque, sean desviados hacia la máquina virtual creada. Para ello, el propio servidor se comportará como un *router*, encaminando y redirigiendo este tráfico.

4.2. Diseño del sistema propuesto

A lo largo de esta sección se describirá la implementación realizada del sistema, así como sus diferencias con respecto a la versión teórica, vista en el punto anterior.

4.2.1. Escenario de la implementación

Teniendo en cuenta las limitaciones que supone el *hardware* disponible para implementar el sistema, resulta necesario recrear un escenario real. Para ello, se utilizará también KVM, tal y como se muestra en la figura 4.2. En el esquema puede observarse como toda la infraestructura de máquinas virtuales se levanta sobre KVM, existiendo también las siguientes redes virtuales:

- **insideNetwork**: se trata de la intranet, una red aislada con encaminamiento interno y a través del *host* únicamente. Su direccionamiento IP es 10.10.10.0/24.
- **management**: es la red de gestión que se utilizará para las comunicaciones entre el *host* y la máquina virtual *Router* en la que está instalado snort. Presenta las mismas características de aislamiento que la red *insideNetwork* y supone una aproximación a la configuración de una red en un sistema real. Esta red es la 10.10.20.0/24.

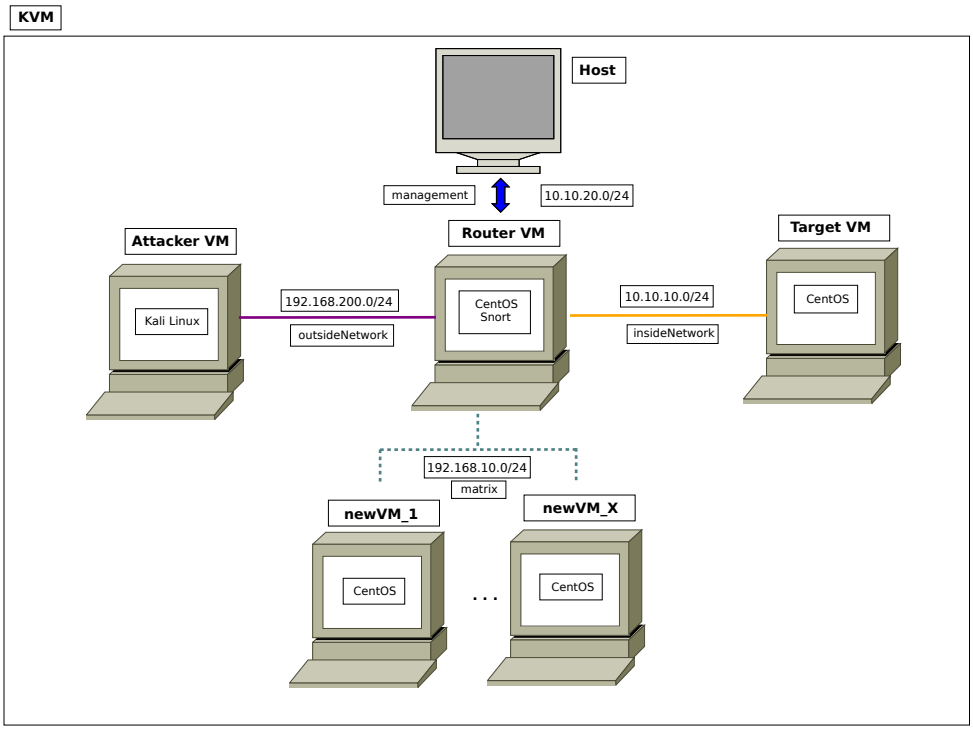

FIGURA 4.2: Esquema del diseño del entorno de implementación

- **outsideNetwork**: esta red simula el exterior, es decir Internet. Como tal, esta red no está aislada, sino que presenta comunicación con el resto de Internet a través del *host* mediante NAT, *Network Addresss Translation*. Debido a las condiciones del entorno simulado, presenta un direccionamiento privado: 192.168.200.0/24.

- **matrix**: se trata de la *honeynet*, la red que albergará las instancias de las máquinas virtuales que se creen dinámicamente. Su direccionamiento es 192.168.10.0/24.

Conectadas a estas redes se encuentran las siguientes máquinas:

- **Attacker**: se encuentra conectada a la red *outsideNetwork* y es la máquina desde la que se llevarán a cabo los ataques contra la red interna. El sistema operativo elegido para esta máquina ha sido *Kali-linux* [57], una distribución diseñada para realizar pruebas de penetración y *hacking* ético. Con ella se pondrá a prueba la eficacia del sistema.

- **Target**: es la máquina que representa el objetivo de los ataques realizados, por lo que se encuentra conectada a *insideNetwork*. Presenta CentOS 7.4 [58] como sistema operativo, pues resulta un sistema muy empleado en servidores de muchas empresas.

- **Router**: podría decirse que se trata del motor de todo el sistema, pues en ella reside gran parte de su lógica. Junto con el *host* conforma el servidor que aparecía en la figura 4.1. Debido a las condiciones que encontramos en este escenario, las funcionalidades que antes se encontraban en un mismo equipo quedan ahora repartidas.
  Está máquina virtual hace, a su vez de router, estando conectada a todas las redes: *insideNetwor, outsideNetwor, management y matrix*. Presenta por lo tanto 4 interfaces de red.

- **NewVM_1-X**: constituyen el conjunto de máquinas virtuales que integrarán la *honeynet* y servirán de señuelo ante un ataque. Se crean tras detectar un ataque, a partir de una plantilla, que se genera a su vez partiendo de la máquina *target*.

4.3. Implementación del sistema propuesto

Tras explicar los aspectos de diseño del entorno recreado, a lo largo de las siguientes secciones se describirá la configuración y funcionamiento en detalle de todos y cada uno de los elementos del sistema, así como su integración entre sí.

4.3.1. Instalación y configuración de KVM

Una vez se ha comprobado que el equipo soporta aceleración *hardware* y que ésta se encuentra habilitada, se realiza la instalación de los siguientes paquetes: *kvm-qemu, libvirt-bin y bridge-utils*. El paquete *libvirt-bin* proporciona una librería que permite explotar las capacidades de virtualización de Linux. Ofrece una API en C compatible

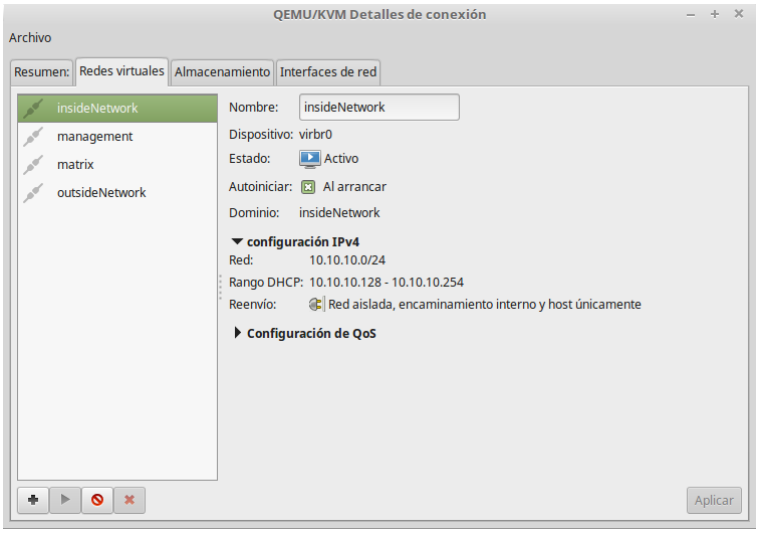

FIGURA 4.3: Virt-manager, interfaz gráfica para gestionar recursos con KVM

con KVM y QEMU, entre otros. Por otro lado, las utilidades *bridge-utils* permiten configurar el puente Ethernet en Linux para, por ejemplo, conectar varios dispositivos Ethernet entre sí de manera transparente. Finalmente, la combinación *kvm-qemu* ya ha sido explicada anteriormente, en el capitulo 3.
Tras la instalación de los paquetes, para que un usuario normal pueda gestionar las máquinas virtuales y demás recursos de KMV sin privilegios de administrador, es necesario que pertenezca a los grupos *kvm* y *libvirt*.

Habiendo configurado correctamente KVM, se utiliza la interfaz gráfica *virt-manager* para la creación de las redes virtuales descritas anteriormente. *Virt-manager*, cuya apariencia puede verse en la figura 4.3, permite la creación sencilla de redes pudiendo especificar diversos parámetros, como la conectividad, el direccionamiento IP o el soporte de IPv6, entre otros.

Esta herramienta crea un fichero *XML* donde quedan definidas todas estas propiedades. A continuación, en el listado 4.1 se muestra el documento XML que define la red *insideNetwork*.

```
<network>
  <name>insideNetwork</name>
  <uuid>fd320f02-9604-41ec-b77c-3f02bc643f57</uuid>
  <bridge name='virbr0' stp='on' delay='0'/>
  <mac address='52:54:00:61:be:f1'/>
  <domain name='insideNetwork'/>
  <ip address='10.10.10.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='10.10.10.128' end='10.10.10.254'/>
    </dhcp>
  </ip>
</network>
```

LISTADO 4.1: Ejemplo de definición de red en XML.

En la definición previa se puede observar como, de forma transparente, KVM crea un *bridge* en el *host*, al que asigna una dirección MAC y que servirá para establecer la comunicación entre las máquinas de esta red, así como con el propio *host*.
El procedimiento para crear las máquinas virtuales de partida, es decir *Attacker*, *Router* y *Target*, es muy similar. También se ha empleado *virt-manager*, siendo necesario únicamente indicar el *path* o ruta de la imagen que serviría de arranque para la instalación del sistema operativo de cada una de ellas: *CentOS-7-x86_64-DVD-1708.iso*, en el caso de Router y Target; *kali-linux-2017.2-amd64.iso*, para Attacker. Otro aspecto que hay que especificar es el volumen del *host* que se utilizará como disco duro en el *guest*. Para ello, KVM soporta varios formatos, de los que se ha elegido *qcow2*. Qcow2, *QEMU Copy On Write 2*, permite aprovechar de manera óptima los recursos del *host* físico, pues únicamente ocupa el espacio que está siendo utilizado. También permite realizar *snapshots* o instantáneas.
Es posible, finalmente, realizar la instalación del sistema operativo a través del cliente VNC, *Virtual Network Computing* de *virt-manager*. Este tipo de conexión solamente se utilizará inicialmente, pues el resto de accesos se realizará mediante SSH, *Secure SHell*.
Para crear o levantar las máquinas virtuales de la *honeynet matrix*, se ha empleado en cambio la interfaz de línea de comandos, *virsh*. Esta utilidad permite multitud de opciones para gestionar las instancias virtuales, desde su creación o arranque, hasta la modificación, listado o borrado. No obstante para la creación de estas máquinas no se ha partido de cero, sino que se han levantado instancias utilizando una plantilla o *template* de *Target* y la opción *clone* del comando *virsh*. Dicha plantilla no es más que un fichero *XML* modificado. A continuación se muestra un ejemplo de la ejecución de dicho comando:

```
virt-clone --original-xml <template> --name <nombreVM> --file <volumen>
```

Con la opción *original-xml* se especifica la ruta en la que se encuentra la plantilla, con *name* se asigna un nombre a la nueva máquina y, finalmente, con *file* se define el nuevo fichero que contendrá el volumen de la instancia.

4.3.2. Instalación y configuración de Snort

La instalación de Snort se realiza a partir de paquetes binarios. Son necesarios el propio Snort, *snort-2.9.9.0-1.x86_64.rpm* y la librería de adquisición de datos *daq-2.0.6-1.el7.x86_64.rpm*. Para localizar ambos paquetes es necesario añadir el repositorio EPEL, *Fedora Extra Packages for Enterprise Linux*, que contiene software que no se encuentra en el repositorio oficial de CentOS. También es necesario descargar las reglas que ofrece la comunidad, *community-rules.tar.gz* y extraerlas en el directorio de configuración correspondiente, es decir en */etc/snort/rules*.
Una vez instalado, Snort se configura mediante la edición de su fichero de configuración, */etc/snort/snort.conf*. Se modifica este fichero para que sea capaz de detectar específicamente tres tipos de ataques:

- **Escaneo de puertos**. Aunque no se trata de un ataque en sí y puede ser realizado en ocasiones por los propios administradores, suele ser una etapa fundamental previa a cualquier ataque, por lo que detectarlo puede llevar incluso a prevenir una futura intrusión.
  Para detectar un escaneo de puertos con Snort es necesario configurar uno de sus preprocesadores, el *sfportscan*, que detecta el escaneo en base al elevado número de respuestas negativas que se devuelven a la máquina atacante como resultado de los sondeos a puertos cerrados. Basta con añadir la siguiente línea al fichero de configuración [1]:

  ```
  preprocessor sfportscan: proto { all } memcap { 10000000 } sense_level {
      medium } scan_type { all } logfile { /var/log/snort/scan/scan.log }
  ```

  Elegir un nivel de sensibilidad adecuado resulta fundamental, pues de ser muy elevado se detectarán escaneos lentos, pero por contra se generarán muchos falsos positivos. Por otro lado una sensibilidad baja puede no llegar a detectar un escaneo.

- **Ataque de fuerza bruta contra el servicio SSH**. Se trata de un ataque que, mediante prueba y error, intenta averiguar un usuario y su correspondiente clave, consiguiendo acceso al intérprete de comando a través del servicio de login remoto SSH. Parte de un diccionario de *logins* frecuentes y prueba contraseñas secuencialmente. De ahí que sea calificado como ataque de fuerza bruta.
  Es posible detectar este tipo de ataques añadiendo una regla a snort que contabilice los intentos de conexión al puerto del servicio SSH, y fije un determinado umbral, a partir del cual pueda considerarse detectado el ataque [2]:

  ```
  alert tcp $EXTERNAL_NET any -> $HOME_NET 22 (msg:"Potencial ataque brute
      force contra el puerto ssh"; flags:S; threshold: type threshold, track
      by_src, count 6, seconds 120; flowbits: set, ssh.brute.attempt;
      classtype:attempted-admin; sid:2001219; rev:8;)
  ```

[1] Se especifican los siguientes parámetros: protocolo a detectar, *proto*; capacidad de memoria, *memcap*; sensibilidad, *sense_level*; tipo de escaneo, *scan_type* y el fichero en el que se registrará cualquier escaneo que se produzca, *logfile*.

[2] Se especifican los puertos sobre los que se registrarán el número de conexiones, el mensaje que se guardará en los logs, así como el umbral para detectarlo. En este caso, se considera que se está llevando a cabo un ataque cuando se producen 6 intentos de conexión contra el puerto SSH en 2 minutos.

- **Inundación del enlace mediante *ping***. Consiste en un ataque DOS, *Denial Of Service*, es decir de denegación de servicio producido por el envío masivo de paquetes *ICMP echo ping request*. La utilidad *ping* cuenta con una opción *flooding*, que le permite enviar peticiones a una tasa muy elevada. Si además se aumenta el tamaño del paquete y, por consiguiente el ancho de banda, el enlace de la máquina de la víctima se satura muy rápidamente.
  Al igual que en el caso anterior, resulta bastante sencillo detectar este ataque con una regla [3]:

```
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"Potencial ataque de
    inundacion del enlace"; threshold: type threshold, track by_dst,count
    30, seconds 120; classtype:attempted-dos; sid:10002;rev:2;)
```

Aparte de añadir las reglas necesarias para detectar estos ataques existen algunos aspectos más que hay que definir en el fichero de configuración, como la red a analizar, qué se considera externo o el módulo de salida a emplear. En este caso, se ha empleado el módulo *syslog*, debido a su posterior integración con *syslog-ng*. Se ha configurado para que filtre aquellos mensajes cuya nivel sea de alerta y los escriba en la *facility local0*, de la que se hablará más adelante, sin ningún tipo de retardo.

4.3.3. Configuración de syslog-ng

Syslog-ng, el sistema de *logging* de aplicaciones, es un sistema flexible y escalable que permite gestionar y centralizar los mensajes de log de diferentes sistemas o servicios.
Syslog-ng se configura a partir de su fichero de configuración */etc/syslog-ng/syslog-ng.conf*, en el que se pueden definir distintos objetos:

- **Origen** Se especifica de dónde proceden los mensajes que recibe, ya sea el propio sistema u otro remoto.
- **Destino** Permite definir dónde se escribirá los logs, típicamente en un fichero, aunque en este caso no tendría por qué ser local.
- **Filtro** Ofrece la posibilidad de establecer opciones de filtrado que se aplicarán sobre los mensajes recibidos. Un ejemplo de estas opciones, es el filtrado por *facility*, campo que especifica qué programa generó el log. Además de las *facilities* reservadas para determinados programas, existen 8 *facilities* reservadas para uso local, *local0-7*.
- **Plantilla** Permite modificar la estructura del log, de manera que parámetros como *host*, hora, etc. se escriban en un determinado orden.

A partir de la definición de los objetos anteriores es posible definir declaraciones de *logs* que procesen los mensajes filtrados de una determinada fuente, para enviarlos a un destino concreto [59]. Un ejemplo de este tipo de declaraciones puede encontrarse en la siguiente línea:

```
log { source(s_sys); filter(f_filtro); destination(d_destino); };
```

[3] La estructura de la regla es similar a la empleada en el caso del ataque de fuerza bruta, con la salvedad de que no se especifica ningún puerto concreto, únicamente el protocolo ICMP y se cambia el umbral de detección, fijándolo en 30 paquetes cada 2 minutos.

Tanto *s_ sys*, como *f_ filtro* o *d_ destino*, son objetos que se han definido previamente e indican el origen, criterio de selección y destino, respectivamente, de los mensajes. En este caso, se añaden los objetos necesarios para que lea los mensajes generados por Snort, tanto desde el preprocesador como desde motor de detección, se modifican para darles un formato sencillo de procesar y se escriben finalmente en un directorio común.

4.3.4. Integración de alertas con KVM

Debido a las características del entorno de implementación y a la separación de Snort y KVM es necesario realizar un paso extra que proporcione la comunicación necesaria entre ambas máquinas.
Esta funcionalidad se ha implementado con un *script* en *Python*, que detecta modificaciones en los ficheros en los que *syslog-ng* escribe los mensajes de alerta de Snort. Para llevar a cabo esta detección se ha utilizado un paquete de *Python* de monitorización de eventos en el sistema de ficheros, denominado *watchdog*. De esta manera, cada vez que se escribe una alerta, modificándose el fichero, ésta es procesada para determinar si se trata de un nuevo ataque o, en su defecto, ya ha sido gestionado. Esta separación se realiza basándose en la dirección IP origen del ataque que, de no estar registrada, indica un nuevo ataque y se procederá a gestionarlo. En este punto, la máquina virtual *Router* se conecta mediante SSH con el *host* y lanza desde ahí un *script* que levantará una nueva máquina virtual en la *honeynet matrix*.

4.3.5. Configuración de login SSH sin clave entre Router VM y el host

Para que la máquina *Router* pueda conectarse al *host* y lanzar un *script* de manera automática, sin que se solicite la clave de usuario, es necesario haber configurado previamente la autenticación SSH con clave pública y privada. Según este método, el cliente SSH, en este caso la máquina *Router*, ha de generar un par de claves, una pública y otra privada. Este mecanismo de autenticación por clave pública y privada constituye un caso de encriptación asimétrica, de manera que la clave privada puede emplearse para cifrar mensajes que solamente la clave pública puede resolver. De esta forma, el servidor, que conoce la clave pública puede validar la identidad del cliente, que guarda la clave privada. El intercambio de mensajes que tiene lugar puede verse en la figura 4.4.

4.3.6. Creación y arranque automático de una máquina virtual

Para crear y arrancar la nueva máquina virtual a la que será desviado el tráfico del atacante, se ha utilizado un *script* en *bash*, uno de los intérpretes de comandos de *GNU Linux*, en la máquina *host*. A su vez, en este *script* se hace uso de la utilidad *virsh*, para gestionar la creación de máquinas virtuales, tal y como se ha explicado previamente. No obstante, ésta no es la única tarea que realiza, sino que, secuencialmente, se realizan las siguientes:

- Comprobar si se ha llegado al máximo número de máquinas virtuales levantadas en *matrix*.
- Clonar una nueva máquina virtual, tomando como base un *template* o plantilla, generado a partir de la máquina virtual *Target*.

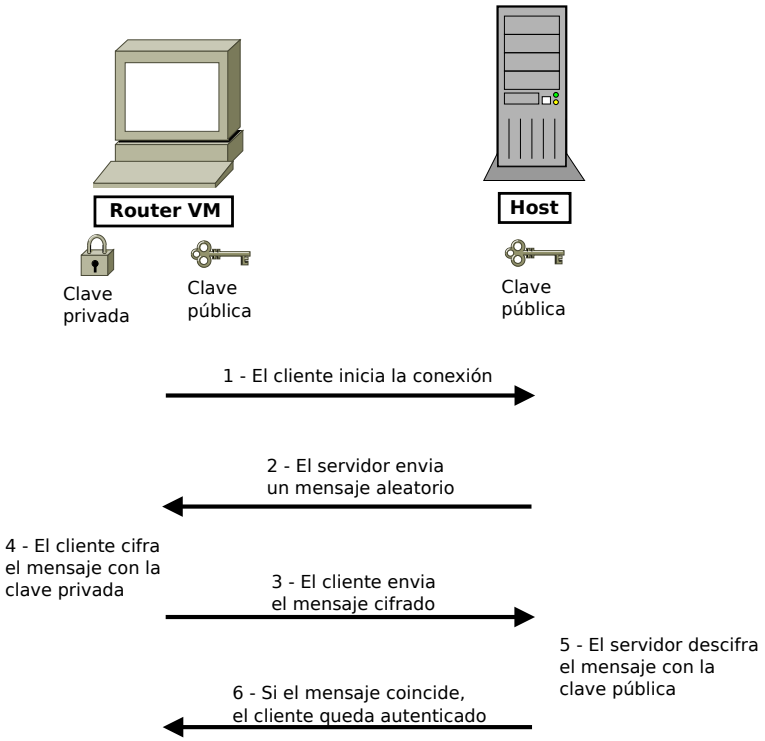

FIGURA 4.4: Auntenticación de un usuario en una conexión SSH mediante clave pública.

- Esperar hasta que el sistema asigne una IP mediante DHCP, *Dynamic Host Configuration Protocol*, a la nueva máquina.
- Configurar el enrutado estático de la máquina creada para que use la máquina virtual *Router* como salida por defecto.
- Utilizar las IPs de la nueva máquina creada y la del atacante, para configurar el *firewall* de la máquina *Router* y desviar el tráfico de la segunda hacia la primera. Esta configuración se explicará en la siguiente sección.
- Borrarlas conexiones guardadas en el *conntrack* de la máquina *Target*. Este paso también será explicado más en detalle a continuación.

El hecho de fijar un número máximo de instancias virtuales resulta de vital importancia, más teniendo en cuenta las limitaciones del *hardware* empleado, pues cada una de ellas supone un gasto de recursos para el *host*. En el caso en que se hubiera superado ya el número máximo de instancias, se prescindiría de la creación de una nueva máquina y sería una de las ya existentes, elegida al azar, la encargada de gestionar el ataque, recibiendo el tráfico procedente de dicha IP.
A la hora de crear una nueva máquina virtual, se utiliza la clonación a partir de una plantilla que presenta las características básicas de la máquina que esta siendo atacada, como el *hostname* o nombre de máquina.
Las configuraciones que se realizan tanto en la nueva máquina creada, como en la máquina *Router*, se llevan a cabo mediante SSH, tal y como ocurría en la comunicación inversa del *Router* hacia el *host*. Con las máquinas virtuales existe la particularidad de que es necesario prescindir de la verificación de su *fingerprint*,[4] pues cambia con

[4]Cuando se instala el servidor SSH se genera una huella o *fingerprint*, que es única para cada servidor. La primera vez que se conecta con un servidor aparece un mensaje para que el cliente siga adelante con la conexión y registre a dicho servidor, junto con su huella, como seguro

cada nueva instancia lanzada.

4.3.7. Configuración del firewall

El *kernel* de Linux cuenta con un *firewall* o cortafuegos que procesa los paquetes que llegan a la máquina, principalmente filtrándolos. Este software es *Netfilter* [60] y, además de realizar las tareas típicas de un cortafuegos, incluye otras funcionalidades como la modificación de los paquetes o la traducción de direcciones IP, lo que se conoce como NAT (*Network Address Translation*). Ésta última característica resulta fundamental de cara a la realización del TFG. Aunque existen varios tipos de NAT, todos tienen en común, tal y como su nombre indica, que traducen direcciones IP. Ésta traducción suele llevarse a cabo en el *router*, de manera que una dirección privada de la red interna se traduce o *mapea* en otra dirección IP fija (NAT estático), en una dirección IP cambiante extraída de un *pool* de direcciones (NAT dinámico) o en un puerto (NAT basado en puerto o NAPT). Este mecanismo generalmente se emplea con el objetivo de reducir el número de direcciones IPv4 públicas utilizadas, pues constituyen un recurso escaso. La figura 4.5 da una idea de cómo funciona el mecanismo de NAT basado en puerto.

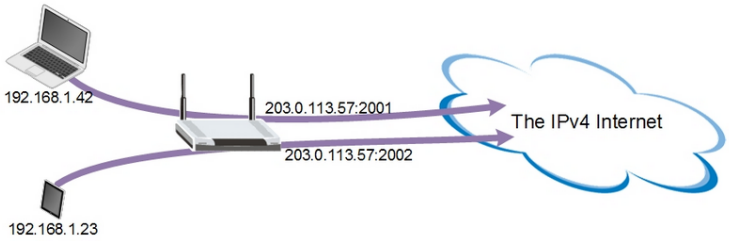

Figura 4.5: Ejemplo de funcionamiento de NAPT [61].

En el caso de Netfilter, la utilidad encargada de gestionar tanto NAT, como el *firewall* es *iptables*. Esta herramienta posee una serie de tablas en las que se encadenan reglas que son examinadas secuencialmente y determinan acciones que realizar sobre los paquetes que llegan y salen de la máquina. Por ejemplo, la tabla *nat*, dispone de dos cadenas: la cadena *prerouting*, que actúa sobre el tráfico que acaba de entrar en la máquina, antes de haber sido enrutado; y la cadena *postrouting*, que afecta a los paquetes que ya han sido reenviados (véase la figura 4.6). Estas dos cadenas que, generalmente, se modifican para configurar NAT, en el presente trabajo se utilizarán para desviar el tráfico atacante hacia la *honeynet*. Se puede definir una regla de tal forma que todo el tráfico que tenga como origen una determinada IP sea encaminado a cierta IP destino:

```
iptables -t nat -I PREROUTING -s <SRC_IP> -j DNAT --to-destination <DST_IP>
```

Ésta es la configuración aplicada en la máquina *Router VM* con respecto a las IPs del atacante y de la nueva instancia virtual. Con cada nueva IP atacante que se detecte, se añadirá una nueva regla al inicio de la tabla *nat* de iptables.

Seguimiento de la conexión o *connection tracking*

Además de las funcionalidades mencionadas anteriormente, Netfilter se caracteriza también por guardar las conexiones y sesiones establecidas con la máquina. Es lo que se conoce como *connection tracking*. Este seguimiento permite que el *kernel* relacione

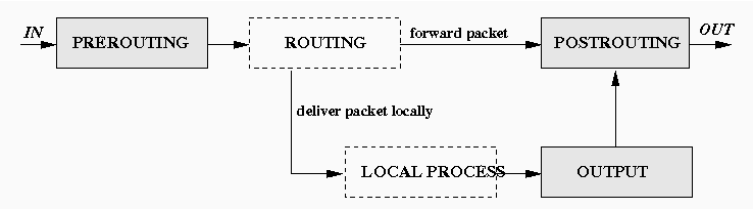

FIGURA 4.6: Cadenas *prerouting* y *postrouting* de iptables [60].

los próximos paquetes que reciba con dichas conexiones. No obstante, estas conexiones o su estado no están relacionados con el de protocolos como TCP, e incluso existen para tráfico no orientado a conexión, como sería UDP. La información de las conexiones establecidas se guarda en el fichero */proc/net/nf_ conntrack*, aunque este fichero puede ser modificado con el conjunto de utilidades *conntrack-tools*. Gracias a esta herramienta, resulta posible eliminar algunas entradas, lo que resulta fundamental para desviar de manera efectiva algunos ataques, tal y como se especificaba en los objetivos. Por ejemplo, ante un ataque DoS, de no eliminar la entrada correspondiente en la tabla, el tráfico procedente del atacante no sería redireccionado pues iptables consideraría que la conexión ya estaba establecida. Hasta que el atacante no interrumpiese el ataque e iniciara uno nuevo, no se aplicaría la regla añadida en iptables.

4.4. Síntesis del funcionamiento del sistema

En este último punto se sintetizará de manera breve y sencilla, con ayuda de unos esquemas, el comportamiento del sistema explicado a lo largo del capítulo.
El diagrama de tiempo que aparece en la figura 4.7 muestra la secuencia de eventos que tiene lugar desde que se inicia un ataque hasta que este ha sido gestionado y ya no supone un peligro. Las flechas rojas representan el ataque que se está efectuando desde Internet. Una vez que Snort, localizado en el *Router*, la detecta y KVM inicia la

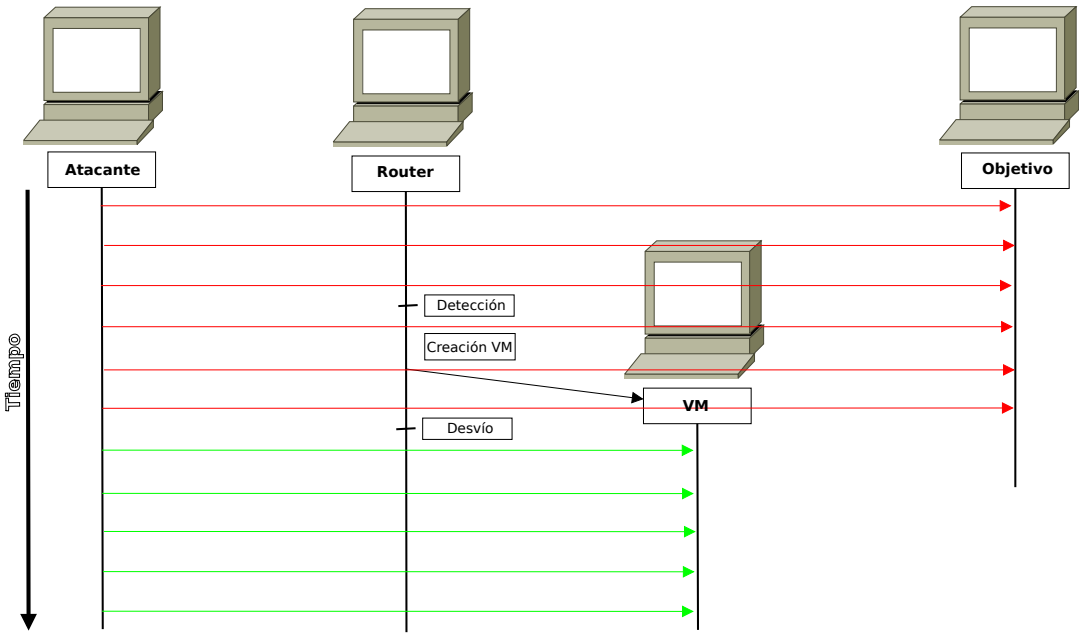

FIGURA 4.7: Diagrama de tiempo que resume la secuencia de eventos en el sistema.

creación de una nueva máquina virtual, existe un pequeño lapso de tiempo hasta que ésta se encuentra disponible para gestionarla. Cuando la nueva instancia está arrancada, el tráfico es redireccionado hacia ella con lo que la máquina Objetivo queda salvaguardada, tal y como indican las flechas verdes.

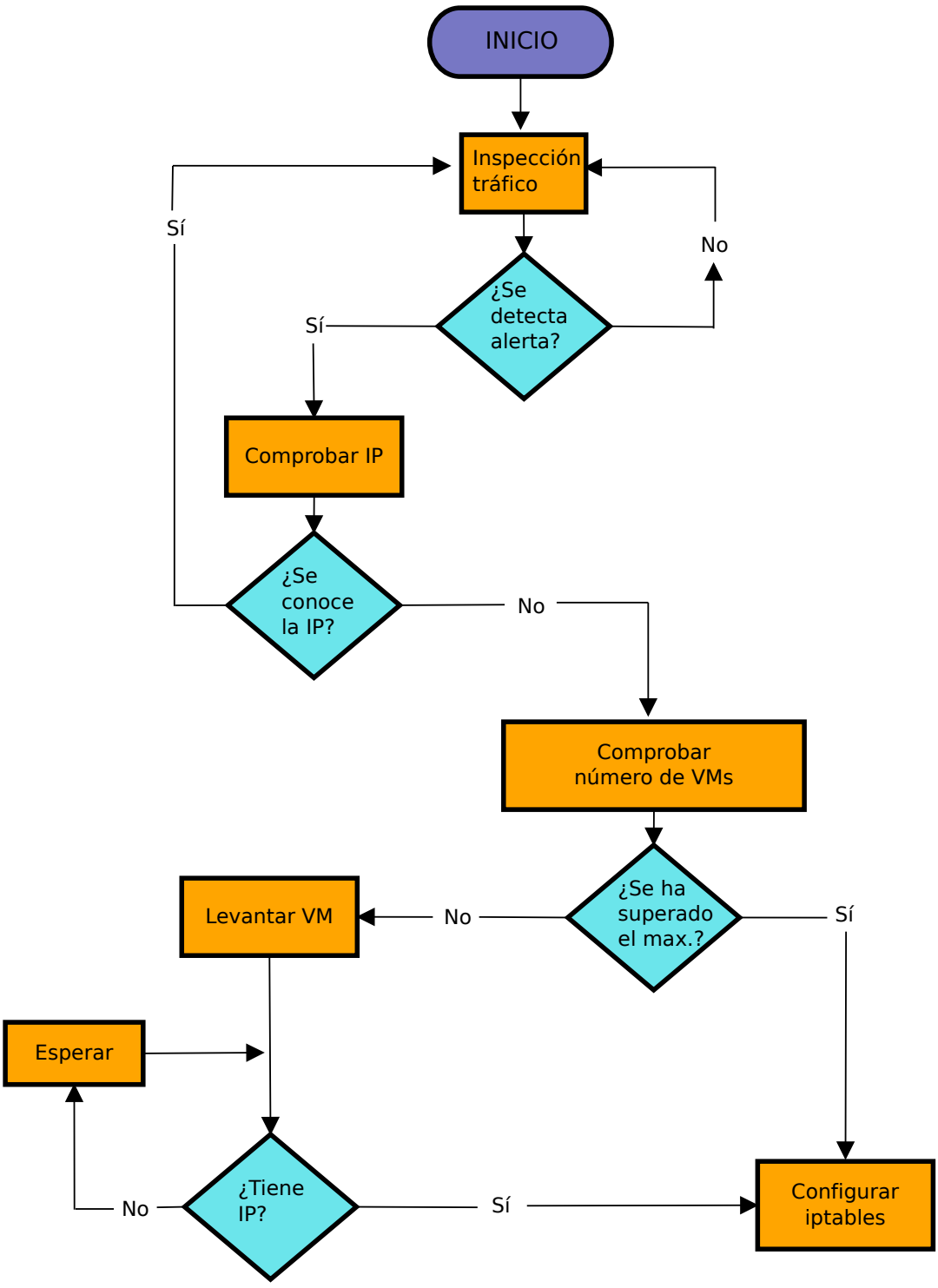

FIGURA 4.8: Diagrama de flujo que resume el comportamiento del sistema.

El segundo esquema, un diagrama de flujo (figura 4.8), sirve de apoyo para resumir la lógica del sistema y los parámetros que son evaluados para determinar cómo gestionará cierto ataque. En este diagrama se abstrae la separación entre máquinas que existe en el entorno implementado para obtener una perspectiva global.

Snort se encuentra arrancado como servicio, de tal manera que analiza el tráfico que llega a la red interna constantemente. En el caso de que se detecte una alarma se comprueba si la IP origen de dicha alarma se ha identificado ya en algún ataque previo y, en cuyo caso ya se estará redirigiendo este tráfico hacia una máquina virtual de matrix. En caso contrario se comprobará el número de máquinas virtuales que se han creado ya y se comparará este valor con el máximo fijado. Si no es posible lanzar otra instancia, se utilizará una de las ya existentes, utilizando su IP para configurar correctamente el *firewall* y desviar la intrusión.

Si, en cambio, resulta factible crear una nueva máquina, cuando esta esté levantada y haya recibido una IP, se procederá igual que si ya estuviera levantada, introduciendo las reglas NAT correspondientes en el *firewall*.

Capítulo 5

Resultados y evaluación del sistema

En este capítulo se explicarán las pruebas realizadas sobre los IDS estudiados, exponiéndose el material y componentes empleados, junto con los resultados obtenidos. Así mismo, se evaluará también el funcionamiento del sistema completo.

5.1. *Hardware* y herramientas empleados

Para la implementación del sistema, además de Snort y KVM se ha hecho uso de más herramientas. Todas ellas se enumeran a continuación, con sus correspondientes versiones.

- **Hardware**
  - Ordenador portátil *packard bell EasyNote TS* con procesador *intel core i5*, 6GB de memoria RAM y 750GB de disco duro.
- **Sistemas operativos**
  - CentOS Linux versión 7.4.1708.
  - Kali Linux versión 4.12.0.
  - Linux Mint 18.2 Sonya.
- **Aplicaciones y paquetes software**
  - Snort IDS versión 2.9.9.0 GRE.
  - Paquete qemu-kvm versión 1:2.5.
  - Libvirt-bin versión 1.3.1.
  - Watchdog version 0.8.3.
- **Lenguajes de programación**
  - Python versión 2.7.5:
- **Utilidades de Linux**
  - Bridge-utils version 1.5.
  - Syslog-ng versión 3.5.6.
  - Netfiler versión 1.6.
  - Virsh version 1.3.1.
  - Virt-manager versión 1.3.2.
  - Tcpdump versión 4.9.0.
  - Nmap versión 7.6.

5.2. Pruebas con Snort y Suricata

En este punto se explican las pruebas realizadas con los IDS, así como los ataques perpetrados para ello.

5.2.1. Escaneo de puertos

Esta técnica no resulta un ataque de por sí, tal y como se ha explicado en el capítulo 4, pues puede ser empleada también por ciertos usuarios, tales como administradores, para comprobar el estado de los puertos y servicios de una determinada máquina. No obstante, es también una etapa fundamental en el estudio previo a la realización de un ataque. Existen multitud de herramientas que permiten llevar a cabo esta tarea, destancando entre ellas, *Nmap, Network mapper*.
Nmap es una utilidad de código abierto distribuido con una licencia GNU, que permite, además de realizar escaneos de puertos, averiguar otras características como el sistema operativo o la versión de los servicios. Además de ser flexible y proporcionar diferentes opciones a la hora de configurar el escaneo, resulta también muy potente y puede emplearse tanto a nivel de un único *host* objetivo como de una red de máquinas.

Tipos de escaneos

Por lo general, al ejecutar el comando *nmap* se lanza un *ping* contra el objetivo, para comprobar que está disponible y, a continuación, se envían una serie de paquetes. En función de cómo sean estos paquetes se distinguen dos tipos de escaneo:

- **Non stealth** Se trata de un tipo de escaneo de fácil detección, pues emplea métodos de conexión TCP, ya sea realizando el típico *three-way handshake*[1] completo o sin el tercer paquete. Este escaneo es el que suelen realizar los administradores de red para realizar sus funciones.

- **Stealth** Se envían paquetes con unos determinados *flags* activados de manera que pueda determinarse el estado de los puertos de manera eficaz. Algunos de las combinaciones de *flags* que pueden utilizarse son F (Fin), *Syn/ack*, *xmas tree*[2] o Null, sin *flag*. En todos estos casos un puerto cerrado responde con un *reset*, RST, mientras que uno abierto no responde. Debido a que no se establece una conexión TCP convencional, este escaneo resulta más difícil de detectar, por lo que es el que emplearía un atacante. Para realizar este escaneo, resulta necesario contar con permisos de administrador para poder generar este tipo de paquetes.

Escaneo con Nmap

El escaneo con Nmap se ha realizado desde la máquina atacante, donde está instalado *Kali Linux*, que ya dispone de esta utilidad.El comando introducido ha sido el siguiente[3]:

```
nmap -A -T4 192.168.1.107
```

El comando *Nmap* se ha ejecutado con la dirección que será objetivo del escaneo y las siguientes opciones:

[1] Negociación en tres etapas que inicia una conexión TCP entre cliente y servidor.
[2] FIN, URG, PSH.
[3] Todas las pruebas se han realizado partiendo de una IP objetivo conocida, aunque averiguarla supondría un paso más en un ataque real.

- `-A` Se utiliza para perpetrar un ataque agresivo en el que, además de detectar el estado de los puertos, se intenta determinar el sistema operativo y la versión de los servicios que se encuentran escuchando en el *host*.
- `-T` Sirve para establecer un esquema de tiempos. En función del valor que tome, se utilizará un mayor o menor ancho de banda, siendo así más sencillo de detectar o no. Un valor de 4 se corresponde con una plantilla temporal agresiva, enviándose los sondeos a una velocidad elevada.

El resultado de realizar este ataque es el siguiente:

```
root@kali:~# nmap -A -T4 10.10.10.145

Starting Nmap 7.60 ( https://nmap.org ) at 2018-03-30 12:39 CEST
mass_dns: warning: Unable to determine any DNS servers. Reverse DNS is
    disabled. Try using --system-dns or specify valid servers with --dns
    -servers
Nmap scan report for 10.10.10.145
Host is up (0.0013s latency).
Not shown: 999 filtered ports
PORT STATE SERVICE VERSION
22/tcp open ssh OpenSSH 7.4 (protocol 2.0)
| ssh-hostkey:
| 2048 d3:5d:58:47:c7:09:ca:dc:6a:4a:71:2e:be:c5:33:2f (RSA)
| 256 75:28:ac:76:82:c8:de:bc:a7:87:7c:6a:53:e9:b4:4f (ECDSA)
|_ 256 b6:18:71:3a:7f:3b:dd:cc:73:95:74:2b:a9:f4:73:33 (EdDSA)
Warning: OSScan results may be unreliable because we could not find at
    least 1 open and 1 closed port
Device type: general purpose
Running: Linux 3.X|4.X
OS CPE: cpe:/o:linux:linux_kernel:3 cpe:/o:linux:linux_kernel:4
OS details: Linux 3.10 - 4.8, Linux 3.2 - 4.8
Network Distance: 2 hops

TRACEROUTE (using port 22/tcp)
HOP RTT ADDRESS
1 1.36 ms 192.168.200.238
2 1.35 ms 10.10.10.145

OS and Service detection performed. Please report any incorrect results
    at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 14.33 seconds
```

La salida de la ejecución muestra que el puerto 22, correspondiente al servicio SSH, se encuentra abierto. El escaneo resulta correcto pues en este caso no hay más servicios arrancados en otros puertos.

Detección con Snort

Se han realizado pruebas de detección variando la sensibilidad, *sense_level*, del preprocesador *sfportscan* entre sus posibles niveles y a partir de la siguiente regla:

```
preprocessor sfportscan: proto { all } memcap { 10000000 } sense_level { medium }
    scan_type { all } logfile { /var/log/snort/scan/scan.log }
```

- **High** Tal y como era de esperar, se detecta la intrusión:

```
Time: 03/30-12:39:31.823299
event_ref: 0
192.168.200.138 -> 10.10.10.145 (portscan) TCP Portscan
Priority Count: 0
Connection Count: 200
IP Count: 1
Scanner IP Range: 192.168.200.138:192.168.200.138
Port/Proto Count: 198
Port/Proto Range: 19:57797
```

- **Medium** En este caso sigue detectándose la intrusión aunque disminuye el recuento de conexiones *Connection Count*:

```
Time: 03/30-15:46:08.814383
event_ref: 0
192.168.200.138 -> 10.10.10.145 (portscan) TCP Portscan
Priority Count: 12
Connection Count: 15
IP Count: 1
Scanner IP Range: 192.168.200.138:192.168.200.138
Port/Proto Count: 15
Port/Proto Range: 23:3306
```

- **Low** Aún con la sensibilidad más baja sigue detectándose el escaneo, aunque esto se debe a que las opciones de *Nmap* definían un modo agresivo:

```
Time: 03/30-15:47:39.781019
event_ref: 0
192.168.200.138 -> 10.10.10.145 (portscan) TCP Portscan
Priority Count: 5
Connection Count: 6
IP Count: 1
Scanner IP Range: 192.168.200.138:192.168.200.138
Port/Proto Count: 6
Port/Proto Range: 23:3389
```

Detección con Suricata

No se ha encontrado ningún elemento equivalente al preprocesador *sfportscan* de Snort en Suricata.

5.2.2. Ataque de fuerza bruta contra el servicio SSH

SSH es una aplicación que permite el *login* remoto de un usuario en el sistema y su acceso al intérprete de comandos. Para conseguir entrar en el sistema son necesarios tanto el nombre, como clave correspondiente de algún usuario, pero dada la elevada capacidad de computo de los ordenadores, puede resultar factible intentar averiguar ambos mediante prueba y error. Este tipo de ataque es lo que se conoce como ataque de fuerza bruta, pues sencillamente parte de un diccionario de *logins* y claves frecuentes que utiliza para tratar de conectarse al puerto ssh, e iniciar una sesión.
La gravedad del éxito de esta intrusión varía en función del usuario, siendo extremadamente elevada para el caso de un usuario con privilegios o root. Es por este motivo

que, como medida de seguridad, resulte conveniente deshabilitar el login remoto como usuario privilegiado o root en el fichero de configuración del servidor SSH, */etc/ssh/sshd.conf*[4].
Se estudiarán tres de las herramientas existentes que permiten realizar este ataque: THC Hydra, Ncrack y Medusa.

THC Hydra

Se trata de una herramienta desarrollada por THC *The Hacker's Choice* y, al igual que ocurría con *Nmap*, es *open source*. Es multiplataforma, se encuentra disponible para diversas plataformas y permite averiguar contraseñas de diferentes servicios y protocolos. Entre los que soporta se encuentran ssh, ldap, ftp y muchos más, lo que la convierte en una de las herramientas más flexibles y versátiles para testear la seguridad de contraseñas y claves. Además, es capaz de probar varias contraseñas de manera concurrente, lo que puede acelerar este ataque, cuyo aspecto crítico es el tiempo empleado.
Para comprobar la detección de este tipo de ataque por los IDSs a estudiar se ha seleccionado como servicio objetivo el del SSH, permitiendo en este caso el login del usuario root.
Desde la máquina atacante, se ha descargado un diccionario de claves, un fichero que contiene un enorme número de posibles claves sobre las que Hydra iterará. Es muy sencillo encontrar ficheros de este tipo en la red, aunque en caso de no disponer de uno, Hydra también tiene una opción que permite probar todas las combinaciones de letras y números de determinadas longitudes.
Dado que el objetivo de este ataque es probar la capacidad del sistema de detección y no la eficacia del ataque en sí, en el fichero utilizado, *password.short* se ha añadido, tras la línea 91, el password del *host* objetivo para no invertir demasiada capacidad de cómputo.
El comando ejecutado desde el atacante es:

```
hydra -l root -P password.short 10.10.10.145 ssh
```

Las opciones especificadas son:

- `-l` Usuario con el que se realiza la conexión
- `-P` *Path* o ruta del fichero diccionario.
- **Dirección IP** En este caso la de la máquina *target*.
- **Servicio** Servicio contra el que se va a realizar el *login*.

Se ha obtenido el siguiente resultado:

```
Hydra v8.2 (c) 2016 by van Hauser/THC - Please do not use in military or
    secret service organizations, or for illegal purposes.

Hydra (http://www.thc.org/thc-hydra) starting at 2017-06-12 10:48:22
[WARNING] Many SSH configurations limit the number of parallel tasks, it is
    recommended to reduce the tasks: use -t 4
[DATA] max 16 tasks per 1 server, overall 64 tasks, 91 login tries (l:1/p
    :91), ~0 tries per task
[DATA] attacking service ssh on port 22
[22][ssh] host: 10.10.10.145 login: root password: Admin.1617
```

[4]La opción para deshabilitarlo es *PermitRootLogin no*.

```
1 of 1 target successfully completed, 1 valid password found
Hydra (http://www.thc.org/thc-hydra) finished at 2017-06-12 10:48:31
```

Como puede observarse, Hydra ha comprobado 92 contraseñas y encontrado la correcta en apenas 8 segundos.

Ncrack

Es, al igual que Hydra, una utilidad para *crackear* contraseñas y llevar a cabo auditorías de la seguridad de contraseñas en el sistema. Fue desarrollada como parte del proyecto *nmap*, permite la adición de módulos para el soporte nuevos de protocolos y presenta distintas opciones de configuración para permitir adaptar su ejecución a las condiciones del entorno. Esta diseñada para realizar auditorías en redes de gran tamaño, aunque también puede utilizarse contra servidores aislados.
También en este caso es posible utilizar un fichero de posibles *logins* con los que autenticarse en el sistema. Para llevar a cabo el estudio de esta herramienta, se establecerán los mismos parámetros que en el caso anterior, es decir, se tratará de encontrar la contraseña de root.
En el comando a ejecutar se especifican el usuario, el fichero con las claves, el número de puerto del servicio y la dirección IP de la máquina:

```
ncrack -p 22 --user root -P password.short 10.10.10.145
```

La ejecución del programa devuelve:

```
Starting Ncrack 0.5 ( http://ncrack.org ) at 2017-06-13 10:51 EDT

Discovered credentials for ssh on 10.10.10.145 22/tcp:
10.10.10.145 22/tcp ssh: 'root' 'Admin.1617'

Ncrack done: 1 service scanned in 30.04 seconds.
```

El resultado es el mismo que al ejecutar Hydra, con la salvedad de que el tiempo empleado es mayor.

Medusa

Se trata del tercer programa empleado para el estudio. Presenta características similares a los anteriores: ejecución concurrente, soporte de multitud de protocolos o diseño modular. Entre las diferencias existentes se encuentra el empleo de threads o hilos para las tareas en paralelo en el caso de Medusa, y de procesos en Hydra.
Para ejecutar Medusa se emplea la siguiente línea:

```
medusa -u root -P password.short -h 10.10.10.145 -M ssh
```

El resultado de ejecutar ese comando ha sido:

```
ACCOUNT CHECK: [ssh] Host: 10.10.10.145 (1 of 1, 0 complete) User: root (1
    of 1, 0 complete) Password: df12345 (1 of 91 complete)
ACCOUNT CHECK: [ssh] Host: 10.10.10.145 (1 of 1, 0 complete) User: root (1
    of 1, 0 complete) Password: df123456 (2 of 91 complete)
ACCOUNT CHECK: [ssh] Host: 10.10.10.145 (1 of 1, 0 complete) User: root (1
    of 1, 0 complete) Password: DF123456df (3 of 91 complete)
```

```
ACCOUNT CHECK: [ssh] Host: 10.10.10.145 (1 of 1, 0 complete) User: root (1
    of 1, 0 complete) Password: df12345v* (4 of 91 complete)

....


ACCOUNT CHECK: [ssh] Host: 10.10.10.145 (1 of 1, 0 complete) User: root (1
    of 1, 0 complete) Password: df2wilson34 (52 of 91 complete)
ACCOUNT CHECK: [ssh] Host: 10.10.10.145 (1 of 1, 0 complete) User: root (1
    of 1, 0 complete) Password: Admin.1617 (53 of 91 complete)
ACCOUNT FOUND: [ssh] Host: 10.10.10.145 User: root Password: Admin.1617 [
    SUCCESS]
```

Aunque la salida del comando no ha registrado el tiempo empleado en la ejecución, éste superó el minuto, siendo notablemente más lento que los anteriores. Hay que tener en cuenta que en éste último caso, cada intento de *login* se mostraba por consola, lo que supone un retardo considerable.

Detección con Snort

Para detectar los ataques anteriores es necesario recurrir al sistema de reglas de Snort. Se registrará una alerta ante la llegada de un determinado número de paquetes TCP con la misma IP de origen, procedentes de cualquier puerto, pero dirigidos al 22. Dichos paquetes han de tener el bit de sincronismo, *SYNC*, que indica el establecimiento de una nueva conexión, activado. El campo *threshold* determina el umbral a partir del cual se considera que se está produciendo el ataque. En este caso, a partir del séptimo paquete en menos de dos minutos, se generará una alerta.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 22 (msg:"Potencial ataque brute force contra
     el puerto ssh"; flags:S; threshold: type threshold, track by_src, count 6,
    seconds 120; flowbits: set, ssh.brute.attempt; classtype:attempted-admin; sid
    :2001219; rev:8;)
```

Efectivamente, se puede comprobar en el fichero de logs como los ataques de fuerza bruta quedan registrados:

```
06/12-16:48:24.370214 [**] [1:2001219:8] Potencial ataque brute force contra
     el puerto ssh [**] [Classification: Attempted Administrator Privilege
    Gain] [Priority: 2] {TCP} 192.168.200.138:57809 -> 10.10.10.145:22
06/12-16:48:24.378220 [**] [1:2001219:8] Potencial ataque brute force contra
     el puerto ssh [**] [Classification: Attempted Administrator Privilege
    Gain] [Priority: 2] {TCP} 192.168.200.138:55428 -> 10.10.10.145:22
06/12-16:48:24.381768 [**] [1:2001219:8] Potencial ataque brute force contra
     el puerto ssh [**] [Classification: Attempted Administrator Privilege
    Gain] [Priority: 2] {TCP} 192.168.200.138:37179 -> 10.10.10.145:22
```

Detección con Suricata

Al igual que Snort, Suricata presenta un sistema de detección basado en reglas, por lo que basta con añadir la regla correspondiente al fichero *local.rules*, tal y como se hizo con Snort:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 22 (msg:"SSH brute force attack"; flags:S;
    flow:to_server; threshold: type threshold, track by_src, count 6, seconds 120;
    flowbits: set, ssh.brute.attempt; classtype:attempted-admin; sid:2; rev:2;)
```

La única diferencia con respecto al caso de Snort es el campo *flow*, en el que se especifica el sentido del tráfico. En este caso, hacia el servidor.
También en este caso se detecta de manera efectiva el ataque, como puede verse en el registro de logs[5]:

```
07/09/2017-16:14:22.315508 [**] [1:2:2] SSH brute force attack [**] [
    Classification: Attempted Administrator Privilege Gain] [Priority: 1] {
    TCP} 192.168.200.138:60478 -> 10.10.10.145:22
07/09/2017-16:14:22.315508 [**] [1:2001219:20] ET SCAN Potential SSH Scan
    [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP}
    192.168.200.138:60478 -> 10.10.10.145:22
07/09/2017-16:14:22.316087 [**] [1:2:2] SSH brute force attack [**] [
    Classification: Attempted Administrator Privilege Gain] [Priority: 1] {
    TCP} 192.168.200.138:53358 -> 10.10.10.145:22
07/09/2017-16:14:27.406369 [**] [1:2:2] SSH brute force attack [**] [
    Classification: Attempted Administrator Privilege Gain] [Priority: 1] {
    TCP} 192.168.200.138:37406 -> 10.10.10.145:22
07/09/2017-16:14:35.201756 [**] [1:2:2] SSH brute force attack [**] [
    Classification: Attempted Administrator Privilege Gain] [Priority: 1] {
    TCP} 192.168.200.138:46622 -> 10.10.10.145:22
```

5.2.3. Ataque DOS de inundación del enlace

Este ataque de denegación de servicio consiste en la inundación del enlace mediante el envío masivo de paquetes *ICMP echo ping request*. Para realizar este tipo de ataque no es necesario recurrir a ningún tipo de aplicación específica, sino que se puede emplear la utilidad *ping*, empleada generalmente para descubrir si un *host* está activo. Entre sus opciones, se encuentra la de *flooding*, *-f*, que permite enviar a una tasa muy elevada peticiones ICMP. Si además se aumenta el tamaño del paquete, *opción -s*, aumentará también el ancho de banda empleado, y es que cuanto mayor sea la tasa a la que envía la máquina atacante, más rápido quedará saturado el enlace de la máquina víctima.
Basta ejecutar el siguiente comando para que la máquina afectada quede colapsada muy rápidamente:

```
ping -f -s 65500 10.10.10.145
```

Detección con Snort

El modo de detección resulta equivalente al empleado en el caso del ataque de fuerza bruta, variando únicamente la caracterización del tráfico en la nueva regla:

```
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"Potencial ataque de inundacion
    del enlace"; threshold: type threshold, track by_dst,count 30, seconds 120;
    classtype:attempted-dos; sid:10002;rev:2;)
```

En este caso se contabilizarán los paquetes ICMP, y además en base a su IP de origen, aumentando el umbral hasta los 30 cada 2 minutos, pues al producirse una inundación del enlace la tasa de llegada será muy elevada.
Las alertas registradas en los logs son las siguientes:

[5]Puede observarse, incluso, como alguna de las reglas de la comunidad genera una alerta de escaneo del puerto SSH, ET SCAN Potential SSH Scan.

```
07/20-18:18:52.105177 [**] [1:10002:2] Potencial ataque de inundacion del
    enlace [**] [Classification: Attempted Denial of Service] [Priority: 2]
    {ICMP} 192.168.200.138 -> 10.10.10.145
07/20-18:18:52.091097 [**] [1:10002:2] Potencial ataque de inundacion del
    enlace [**] [Classification: Attempted Denial of Service] [Priority: 2]
    {ICMP} 192.168.200.138 -> 10.10.10.145
07/20-18:18:52.076538 [**] [1:10002:2] Potencial ataque de inundacion del
    enlace [**] [Classification: Attempted Denial of Service] [Priority: 2]
    {ICMP} 192.168.200.138 -> 10.10.10.145
07/20-18:18:52.055221 [**] [1:10002:2] Potencial ataque de inundacion del
    enlace [**] [Classification: Attempted Denial of Service] [Priority: 2]
    {ICMP} 192.168.200.138 -> 10.10.10.145
07/20-18:18:52.033063 [**] [1:10002:2] Potencial ataque de inundacion del
    enlace [**] [Classification: Attempted Denial of Service] [Priority: 2]
    {ICMP} 192.168.200.138 -> 10.10.10.145
```

Detección con Suricata

La regla empleada en la configuración de Suricata resulta análoga a la de Snort:

```
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"Ataque de inundacion del enlace";
     threshold: type threshold, track by_dst,count 30, seconds 120; flowbits: set,
    dos.attempt; classtype:attempted-dos; sid:10002;rev:2;)
```

Al igual que el resultado visto en los logs:

```
07/20-18:18:52.105177 [**] [1:10002:2] Ataque de inundacion del enlace [**]
    [Classification: Attempted Denial of Service] [Priority: 2] {ICMP}
    192.168.200.138 -> 10.10.10.145
07/20-18:18:52.091097 [**] [1:10002:2] Ataque de inundacion del enlace [**]
    [Classification: Attempted Denial of Service] [Priority: 2] {ICMP}
    192.168.200.138 -> 10.10.10.145
07/20-18:18:52.076538 [**] [1:10002:2] Ataque de inundacion del enlace [**]
    [Classification: Attempted Denial of Service] [Priority: 2] {ICMP}
    192.168.200.138 -> 10.10.10.145
07/20-18:18:52.055221 [**] [1:10002:2] Ataque de inundacion del enlace [**]
    [Classification: Attempted Denial of Service] [Priority: 2] {ICMP}
    192.168.200.138 -> 10.10.10.145
07/20-18:18:52.033063 [**] [1:10002:2] Ataque de inundacion del enlace [**]
    [Classification: Attempted Denial of Service] [Priority: 2] {ICMP}
    192.168.200.138 -> 10.10.10.145
```

5.3. Evaluación del sistema completo

La evaluación del sistema se ha realizado utilizando como base los objetivos de partida (véase el capítulo 1) y lo que se pretendía conseguir. Tal y como se explica en el capítulo anterior, 4, existen dos etapas concretas a realizar. En primer lugar, resulta necesario que el sistema sea capaz de detectar una intrusión o ataque a la red haciendo uso de un IDS. En segundo lugar se ha de desviar de manera efectiva dicho ataque hacia una *honeynet*.
De cara a esa primera fase de detección, se han evaluado y comparado dos de los IDS más utilizados a día de hoy, Snort y Suricata, siendo el primero el más adecuado para

este caso concreto[6]. Las pruebas de la sección anterior muestran que, efectivamente, este software puede detectar y alertar de los ataques propuestos al inicio de la memoria.

En lo que respecta a la etapa de desvío de la intrusión, se ha utilizado KVM, conjuntamente con Netfilter, para en primer lugar crear la red virtual que constituye la *honeynet*, con KVM y, en segundo lugar, redirigir el tráfico procedente del atacante mediante la utilidad iptables de Netfilter.

Para verificar el funcionamiento del sistema completo y la integración de sus componentes, se ha recurrido a diferentes *logs*, generados a partir de los *scripts* y a capturas del tráfico en distintos puntos de las redes con la utilidad tcpdump:

- ***Logs***: existen logs, tanto en las máquinas virtuales, como en el propio *host*, que permiten monitorizar los eventos que tienen lugar en el sistema. A continuación se muestran los que han resultado de mayor utilidad:
  - */tmp/launchMachine.log*: este *log* de la máquina *host* registra el evento de creación de una nueva máquina virtual lanzado desde el *script launch Machine.sh*. Recoge la información de la nueva máquina creada, así como la configuración de su enrutamiento o de iptables en la máquina *router* mediante SSH.
  - */var/log/snort/snort.log*: es el *log* generado por *syslog-ng* ante una alerta de Snort. En él se muestra la información referente al ataque detectado.
  - *Output del script eventDetection.py*: no se trata de un *log* en sí, sino de la salida por pantalla de la ejecución del programa. Muestra el estado de la gestión de un ataque, sí es nuevo o, en su defecto, ya ha sido desviado.
- **Capturas de tráfico**: se ha emmpleado *tcpdump* para analizar el tráfico que pasa por las interfaces de la máquina *Router eth1* y *eth2*, es decir, las conectadas a las redes *insideNetwork* y *matrix*. Puede observarse cómo, ante un desvío efectivo del sistema, el tráfico deja de atravesar la interfaz de *insideNetwork*, para utilizar la de la interfaz *matrix*.

A continuación se muestran los *logs* y las capturas de tráfico obtenidas ante la realización de un ataque DoS, de inundación del enlace desde la máquina *Attacker*:

- **snort.log**: el formato de estas alarmas se vio también al inicio del capítulo, donde se mostraban las pruebas realizadas con Snort y Suricata.

```
[root@router ~]# tail -100 /var/log/snort/snort.log
2018-04-21T12:54:14+02:00 router [1:10002:2] Potencial ataque de inundacion
    del enlace [Classification: Attempted Denial of Service] [Priority: 2] {
    ICMP} 192.168.200.138 -> 10.10.10.145
2018-04-21T12:54:14+02:00 router [1:10002:2] Potencial ataque de inundacion
    del enlace [Classification: Attempted Denial of Service] [Priority: 2] {
    ICMP} 192.168.200.138 -> 10.10.10.145
2018-04-21T12:54:14+02:00 router [1:10002:2] Potencial ataque de inundacion
    del enlace [Classification: Attempted Denial of Service] [Priority: 2] {
    ICMP} 192.168.200.138 -> 10.10.10.145
2018-04-21T12:54:14+02:00 router [1:10002:2] Potencial ataque de inundacion
    del enlace [Classification: Attempted Denial of Service] [Priority: 2] {
    ICMP} 192.168.200.138 -> 10.10.10.145
2018-04-21T12:54:14+02:00 router [1:10002:2] Potencial ataque de inundacion
    del enlace [Classification: Attempted Denial of Service] [Priority: 2] {
    ICMP} 192.168.200.138 -> 10.10.10.145
```

[6]Véase el capítulo 2.

```
2018-04-21T12:54:14+02:00 router [1:10002:2] Potencial ataque de inundacion
    del enlace [Classification: Attempted Denial of Service] [Priority: 2] {
    ICMP} 192.168.200.138 -> 10.10.10.145
2018-04-21T12:54:14+02:00 router [1:10002:2] Potencial ataque de inundacion
    del enlace [Classification: Attempted Denial of Service] [Priority: 2] {
    ICMP} 192.168.200.138 -> 10.10.10.145
2018-04-21T12:54:14+02:00 router [1:10002:2] Potencial ataque de inundacion
    del enlace [Classification: Attempted Denial of Service] [Priority: 2] {
    ICMP} 192.168.200.138 -> 10.10.10.145
2018-04-21T12:54:14+02:00 router [1:10002:2] Potencial ataque de inundacion
    del enlace [Classification: Attempted Denial of Service] [Priority: 2] {
    ICMP} 192.168.200.138 -> 10.10.10.145
```

- **eventDetection.py**: dado que en este caso se está realizando un ataque de *flooding* y llegan muchos paquetes, una vez iniciada la gestión del nuevo ataque seguirán generándose alertas, pero éstas ya no darán lugar a la creación de nuevas máquinas. La ejecución de este *script* muestra la siguiente información:

```
[root@router ~]# ./eventDetection.py
Waiting for event
SE HA DETECTADO UN NUEVO ATAQUE!!!!
Ejecutado comando en host remoto
El ataque ya ha sido gestionado
El ataque ya ha sido gestionado
El ataque ya ha sido gestionado
```

- **launchMachine.log**: se muestran en este orden la clonación de la nueva máquina virtual, la información de la misma, la IP que recibe, la configuración de la ruta por defecto en esta nueva instancia, la configuración de *iptables* en la máquina *Router* y, finalmente, el borrado de las conexiones guardadas en su *kernel*.

```
Exec: main
Exec: func_check_vm_num
0
newVM_1
Exec: func_clone_machine
Asignando 'newVM_1' | 40 GB 00:44

El clon 'newVM_1' ha sido creado exitosamente.

Getting information of new machine
Id: -
Nombre: newVM_1
UUID: 7da5bd2d-331e-4369-be58-29999c853ee0
Tipo de sistema operatuvo: hvm
Estado: apagado
CPU(s): 1
Memoria maxima: 2097152 KiB
Memoria utilizada: 2097152 KiB
Persistente: si
Autoinicio: desactivar
Guardar administrado: no
Modelo de seguridad: apparmor
DOI de seguridad: 0

Se ha iniciado el dominio newVM_1
```

```
192.168.10.169

Exec: ssh root@192.168.10.169 ip addr add 192.168.10/24 dev eth0
Exec: ssh root@192.168.10.169 ip r add default via 192.168.10.150

Fijar regla en el firewall
iptables -t nat -I PREROUTING -s 192.168.200.138 -j DNAT --to-destination
    192.168.10.169

liberar conexiones a la maquina target

Exec: ssh root@router /usr/sbin/conntrack -D -s 192.168.200.138 -d
    10.10.10.145
icmp 1 29 src=192.168.200.138 dst=10.10.10.145 type=8 code=0 id=772 src
    =10.10.10.145 dst=192.168.200.138 type=0 code=0 id=772 mark=0 secctx=
    system_u:object_r:unlabeled_t:s0 use=1

Execution completed
```

- **Captura del tráfico en *eth1***: se trata de la interfaz conectada a *insideNetwork*, por lo que se ven los paquetes que llegan a la máquina *Target* al inicio del ataque.

```
[root@router ~]# date & tcpdump -i eth1 -e icmp
[1] 2002
sab abr 21 12:52:37 CEST 2018
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
12:52:40.615017 52:54:00:7f:20:da (oui Unknown) > 52:54:00:ba:33:4d (oui
    Unknown), ethertype IPv4 (0x0800), length 98: attacker.insideNetwork >
    target: ICMP echo request, id 972, seq 1, length 64
12:52:40.615461 52:54:00:ba:33:4d (oui Unknown) > 52:54:00:7f:20:da (oui
    Unknown), ethertype IPv4 (0x0800), length 98: target > attacker.
    insideNetwork: ICMP echo reply, id 972, seq 1, length 64
12:52:40.616016 52:54:00:7f:20:da (oui Unknown) > 52:54:00:ba:33:4d (oui
    Unknown), ethertype IPv4 (0x0800), length 98: attacker.insideNetwork >
    target: ICMP echo request, id 972, seq 2, length 64
12:52:40.616205 52:54:00:ba:33:4d (oui Unknown) > 52:54:00:7f:20:da (oui
    Unknown), ethertype IPv4 (0x0800), length 98: target > attacker.
    insideNetwork: ICMP echo reply, id 972, seq 2, length 64
12:52:40.616614 52:54:00:7f:20:da (oui Unknown) > 52:54:00:ba:33:4d (oui
    Unknown), ethertype IPv4 (0x0800), length 98: attacker.insideNetwork >
    target: ICMP echo request, id 972, seq 3, length 64
12:52:40.616905 52:54:00:ba:33:4d (oui Unknown) > 52:54:00:7f:20:da (oui
    Unknown), ethertype IPv4 (0x0800), length 98: target > attacker.
    insideNetwork: ICMP echo reply, id 972, seq 3, length 64
12:52:40.617334 52:54:00:7f:20:da (oui Unknown) > 52:54:00:ba:33:4d (oui
    Unknown), ethertype IPv4 (0x0800), length 98: attacker.insideNetwork >
    target: ICMP echo request, id 972, seq 4, length 64
12:52:40.617533 52:54:00:ba:33:4d (oui Unknown) > 52:54:00:7f:20:da (oui
    Unknown), ethertype IPv4 (0x0800), length 98: target > attacker.
    insideNetwork: ICMP echo reply, id 972, seq 4, length 64
12:52:40.617913 52:54:00:7f:20:da (oui Unknown) > 52:54:00:ba:33:4d (oui
    Unknown), ethertype IPv4 (0x0800), length 98: attacker.insideNetwork >
    target: ICMP echo request, id 972, seq 5, length 64
12:52:40.618099 52:54:00:ba:33:4d (oui Unknown) > 52:54:00:7f:20:da (oui
    Unknown), ethertype IPv4 (0x0800), length 98: target > attacker.
    insideNetwork: ICMP echo reply, id 972, seq 5, length 64
```

```
12:52:40.618474 52:54:00:7f:20:da (oui Unknown) > 52:54:00:ba:33:4d (oui
    Unknown), ethertype IPv4 (0x0800), length 98: attacker.insideNetwork >
    target: ICMP echo request, id 972, seq 6, length 64
```

- **Captura del tráfico en *eth2***: los paquetes empiezan a aparecer por esta interfaz una vez se ha creado la máquina virtual de la *honeynet* y se han realizado las configuraciones correspondientes. De hecho puede observarse una diferencia de casi dos minutos desde que se produce el ataque, hasta que se empieza a registrar tráfico en ésta segunda captura.

```
[root@router ~]# date & tcpdump -i eth2 -e icmp
[1] 2004
sab abr 21 12:52:38 CEST 2018
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth2, link-type EN10MB (Ethernet), capture size 262144 bytes
12:54:14.168396 52:54:00:ea:ad:fc (oui Unknown) > 52:54:00:5c:04:89 (oui
    Unknown), ethertype IPv4 (0x0800), length 98: attacker.insideNetwork >
    192.168.10.195: ICMP echo request, id 972, seq 64848, length 64
12:54:14.168638 52:54:00:5c:04:89 (oui Unknown) > 52:54:00:ea:ad:fc (oui
    Unknown), ethertype IPv4 (0x0800), length 98: 192.168.10.195 > attacker.
    insideNetwork: ICMP echo reply, id 972, seq 64848, length 64
12:54:14.169101 52:54:00:ea:ad:fc (oui Unknown) > 52:54:00:5c:04:89 (oui
    Unknown), ethertype IPv4 (0x0800), length 98: attacker.insideNetwork >
    192.168.10.195: ICMP echo request, id 972, seq 64849, length 64
12:54:14.169516 52:54:00:5c:04:89 (oui Unknown) > 52:54:00:ea:ad:fc (oui
    Unknown), ethertype IPv4 (0x0800), length 98: 192.168.10.195 > attacker.
    insideNetwork: ICMP echo reply, id 972, seq 64849, length 64
12:54:14.170020 52:54:00:ea:ad:fc (oui Unknown) > 52:54:00:5c:04:89 (oui
    Unknown), ethertype IPv4 (0x0800), length 98: attacker.insideNetwork >
    192.168.10.195: ICMP echo request, id 972, seq 64850, length 64
12:54:14.170254 52:54:00:5c:04:89 (oui Unknown) > 52:54:00:ea:ad:fc (oui
    Unknown), ethertype IPv4 (0x0800), length 98: 192.168.10.195 > attacker.
    insideNetwork: ICMP echo reply, id 972, seq 64850, length 64
12:54:14.170749 52:54:00:ea:ad:fc (oui Unknown) > 52:54:00:5c:04:89 (oui
    Unknown), ethertype IPv4 (0x0800), length 98: attacker.insideNetwork >
    192.168.10.195: ICMP echo request, id 972, seq 64851, length 64
```

En definitiva, a la vista de los resultados de las pruebas realizadas, se consideran cumplidos los objetivos establecidos al inicio del presente TFG.

Capítulo 6

Conclusiones y líneas futuras de investigación

A lo largo de este capítulo se comentarán las conclusiones extraídas de la realización del trabajo, tanto en lo referente a las herramientas y tecnologías empleadas, como en lo referente al ámbito personal. Finalmente, se presentan algunas de las posibles investigaciones que pudieran surgir en el futuro a partir de este TFG.

6.1. Conclusiones

A continuación, en este punto se refieren las conclusiones más relevantes extraídas del trabajo.

6.1.1. Observaciones

En este punto se comentarán algunos de los más relevantes aspectos advertidos a raíz del trabajo con los distintos elementos del sistema:

- Snort constituye una herramienta muy potente para detectar ataques, aunque esto también supone que en ocasiones sea algo complejo configurarlo.
- El sistema de *logging* syslog-ng resulta de gran utilidad para procesar y gestionar mensajes, siendo además más robusto que su alternativa, *rsyslog*. Éste último suponía a veces un problema al bloquearse y ser necesario reiniciarlo.
- La configuración de syslog-ng, haciendo uso de objetos, es extremadamente sencilla.
- Ha resultado fundamental el empleo de la utilidad *virsh* y sus múltiples opciones para la creación de instancias virtuales, así como para la lectura de algunos de sus parámetros.
- Para la realización de los tests de penetración desde la máquina *Kali Linux* ha sido de gran ayuda contar con multitud de blogs y foros con información y tutoriales al respecto.
- Una de las utilidades más empleadas ha sido Tcpdump, pues resultaba muy efectivo a la hora de comprobar la correcta configuración de *Iptables* e inspeccionar el tráfico que llegaba a las máquinas.

6.1.2. Logros y puntos de mejora del sistema

En esta sección se analizarán tanto los aspectos positivos, como los que resulta necesario mejorar en el sistema.

Logros

- Detección eficaz de tres tipos distintos de ataques llevados a cabo desde una única máquina.
- Registro temporal de las direcciones IP de origen de los ataques. Supone un ahorro de recursos y tiempo, pues los ataques son gestionados una única vez.
- Creación de manera automática de nuevas máquinas virtuales, sin intervención de ningún administrador.
- Control sobre el número de instancias virtuales creadas para evitar superar las capacidades del *host* físico.
- Configuración automática del encaminamiento estático en las nuevas máquinas virtuales.
- Desvío eficaz de la intrusión hacia una red creada dinámicamente, con la configuración automática de *Iptables*.

Puntos de mejora y vulnerabilidades

- La detección de los ataques mencionados anteriormente no resulta eficaz si estos se realizan haciendo uso de una *botnet*[1].
- Resultaría conveniente ampliar el rango de ataques que Snort es capaz de detectar. Haciendo uso de las reglas de la comunidad es posible identificar más ataques, aunque sería necesario verificarlo de manera fehaciente.
- Registro permanente de las direcciones IP atacantes en una base de datos. Actualmente, estos datos se guardan en un *array*[2] y permanecen en memoria únicamente mientras el programa se está ejecutando.
- Empaquetado del *script* de *Python* para que pueda ser ejecutado como un servicio más. Aunque una vez arrancado, inicia la comprobación de alarmas, así como el arranque de máquinas virtuales automáticamente, resulta necesario ejecutarlo manualmente. Si fuera un servicio sería posible, por ejemplo, ejecutarlo desde el arranque del sistema.
- Mejora del *script* de creación de las nuevas máquinas virtuales. Tal y como esta ahora, si coinciden dos ataques y han de crearse dos instancias a la vez, se produciría un conflicto.

6.1.3. Conclusiones personales

La realización de este trabajo ha supuesto un reto en diversos aspectos, desde lo que supone abordar un proyecto de forma individual hasta el empleo de nuevas tecnologías.
Por un lado, la investigación en un ámbito de gran importancia como es la seguridad en Internet me ha resultado muy interesante, además de útil dado el gran impacto que tiene a día de hoy. Es posible aplicar estos nuevos conocimientos tanto en el entorno laboral, como a nivel personal. Por otro lado, el uso de aplicaciones concretas

[1] Una botnet es una red de ordenadores infectados que son controlados remotamente para perpetrar ataques.
[2] Un array es un vector que guarda una serie de elementos.

con las que no había trabajado previamente, como son Snort y KVM, ha resultado una etapa de aprendizaje crucial, especialmente en el caso de KVM pues la virtualización supone la base de muchas de las tecnologías punteras que existen actualmente. Además, el empleo de sistemas y *software* muy diferentes me ha obligado a buscar distintas soluciones para integrarlos, aumentando así mis conocimientos sobre Linux y familiarizándome con su gestión.
En definitiva, considero que este trabajo ha resultado fundamental para consolidar y poner en práctica los conocimientos adquiridos a lo largo de todo el grado.

6.2. Posibles trabajos futuros

Este trabajo podría servir como base para el desarrollo de nuevas investigaciones:

6.2.1. Traslado del sistema a un entorno real

Para empezar, sería muy interesante montar el sistema planteado en el capítulo 4 en un escenario real, lo que permitiría analizar su viabilidad en redes no simuladas. Esta tarea supondría llevar a cabo una serie de cambios en los *scripts* empleados puesto que desaparecen las comunicaciones entre el *host* y la máquina *Router* al encontrarse Snort y KVM en la misma máquina. También implica disponer del software necesario.

6.2.2. Rastreo del ataque

Una de las razones que impulsó la creación de los *honeypots*, o *honeynets* en general, fue la de aumentar los conocimientos acerca de los ataques que se producen en la red. En este caso se desvía de manera efectiva la intrusión, pero el siguiente paso sería aprovecharla, investigar las conexiones con el atacante para localizarlo y averiguar cómo actúa. Sería necesario, por ejemplo, un sistema de monitorización que siguiera la actividad que el atacante lleva a cabo en la máquina virtual.

6.2.3. Pruebas de rendimiento

Sería necesario analizar de manera cuantitativa el comportamiento del sistema. Para ello podrían realizarse pruebas de carga y de estrés[3]. También habría que determinar de manera exacta el tiempo de respuesta del sistema[4] y estudiar si habría alguna forma de reducirlo.

[3]Las pruebas de carga estudian el comportamiento de una aplicación ante una cantidad esperada de peticiones, mientras que en las pruebas de estrés éstas se aumentan mucho tratando de *romper* la aplicación.

[4]El tiempo que transcurre desde que se detecta la intrusión hasta que ésta ha sido gestionada.

Apéndice A

Ficheros de configuración y *scripts* del sistema

En este segundo apéndice se mostrarán tanto los ficheros de configuración de los distintos servicios del sistema, como los *scripts* en *Python* y *bash* empleados.

A.1. Máquinas KVM

A continuación se muestran las definiciones de las máquinas virtuales empleadas inicialmente en el entorno: *attacker*, *router* y *target*. Aunque existen multitud de dispositivos e interfaces, que emulan una máquina física, los atributos más importantes son los siguientes:

- **disk** Apunta al volumen de la máquina física que será el disco duro de la máquina virtual. En todos los casos se ha utilizado una imagen *qcow2*.
- **interface** Define la interfaz de red, la red virtual a la que se conectará, así como la dirección MAC.

Las configuraciones de las tres máquinas son muy similares, con la salvedad de las 4 interfaces de red de que dispone la máquina *router*, tal y como puede verse en los siguientes puntos.
Para mostrar las definiciones de las máquinas se utiliza el comando *virsh dumpxml* <*nombre_maquina*>.

A.1.1. Attacker

```
<domain type='kvm'>
  <name>Attacker</name>
  <uuid>b0490c77-fa5e-4837-825c-634025be3c4d</uuid>
  <memory unit='KiB'>2097152</memory>
  <currentMemory unit='KiB'>2097152</currentMemory>
  <vcpu placement='static'>1</vcpu>
  <os>
    <type arch='x86_64' machine='pc-i440fx-xenial'>hvm</type>
    <boot dev='hd'/>
  </os>
  <features>
    <acpi/>
    <apic/>
  </features>
  <cpu mode='custom' match='exact'>
    <model fallback='allow'>SandyBridge</model>
  </cpu>
  <clock offset='utc'>
```

```
  <timer name='rtc' tickpolicy='catchup'/>
  <timer name='pit' tickpolicy='delay'/>
  <timer name='hpet' present='no'/>
</clock>
<on_poweroff>destroy</on_poweroff>
<on_reboot>restart</on_reboot>
<on_crash>restart</on_crash>
<pm>
  <suspend-to-mem enabled='no'/>
  <suspend-to-disk enabled='no'/>
</pm>
<devices>
  <emulator>/usr/bin/kvm-spice</emulator>
  <disk type='file' device='disk'>
    <driver name='qemu' type='qcow2'/>
    <source file='/home/helena/Tfg/Development/outside/images/kali.qcow2'/>
    <target dev='hda' bus='ide'/>
    <address type='drive' controller='0' bus='0' target='0' unit='0'/>
  </disk>
  <disk type='file' device='cdrom'>
    <driver name='qemu' type='raw'/>
    <target dev='hdb' bus='ide'/>
    <readonly/>
    <address type='drive' controller='0' bus='0' target='0' unit='1'/>
  </disk>
  <controller type='usb' index='0' model='ich9-ehci1'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x7'/>
  </controller>
  <controller type='usb' index='0' model='ich9-uhci1'>
    <master startport='0'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x0'
        multifunction='on'/>
  </controller>
  <controller type='usb' index='0' model='ich9-uhci2'>
    <master startport='2'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x1'/>
  </controller>
  <controller type='usb' index='0' model='ich9-uhci3'>
    <master startport='4'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x2'/>
  </controller>
  <controller type='pci' index='0' model='pci-root'/>
  <controller type='ide' index='0'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x1'/>
  </controller>
  <controller type='virtio-serial' index='0'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0'/>
  </controller>
  <interface type='network'>
    <mac address='52:54:00:7f:20:da'/>
    <source network='outsideNetwork'/>
    <model type='rtl8139'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
  </interface>
  <serial type='pty'>
    <target port='0'/>
  </serial>
  <console type='pty'>
    <target type='serial' port='0'/>
  </console>
  <channel type='spicevmc'>
    <target type='virtio' name='com.redhat.spice.0'/>
    <address type='virtio-serial' controller='0' bus='0' port='1'/>
```

```
    </channel>
    <input type='mouse' bus='ps2'/>
    <input type='keyboard' bus='ps2'/>
    <graphics type='spice' autoport='yes'>
      <image compression='off'/>
    </graphics>
    <sound model='ich6'>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x04' function='0x0'/>
    </sound>
    <video>
      <model type='qxl' ram='65536' vram='65536' vgamem='16384' heads='1'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x02' function='0x0'/>
    </video>
    <redirdev bus='usb' type='spicevmc'>
    </redirdev>
    <redirdev bus='usb' type='spicevmc'>
    </redirdev>
    <memballoon model='virtio'>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x07' function='0x0'/>
    </memballoon>
  </devices>
</domain>
```

A.1.2. Router

```
<domain type='kvm' id='2'>
  <name>router</name>
  <uuid>41f78cf1-dfdf-4ced-a992-9b5e9a73d5b6</uuid>
  <memory unit='KiB'>2097152</memory>
  <currentMemory unit='KiB'>2097152</currentMemory>
  <vcpu placement='static'>1</vcpu>
  <resource>
    <partition>/machine</partition>
  </resource>
  <os>
    <type arch='x86_64' machine='pc-i440fx-xenial'>hvm</type>
    <boot dev='hd'/>
  </os>
  <features>
    <acpi/>
    <apic/>
  </features>
  <cpu mode='custom' match='exact'>
    <model fallback='allow'>SandyBridge</model>
  </cpu>
  <clock offset='utc'>
    <timer name='rtc' tickpolicy='catchup'/>
    <timer name='pit' tickpolicy='delay'/>
    <timer name='hpet' present='no'/>
  </clock>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>restart</on_crash>
  <pm>
    <suspend-to-mem enabled='no'/>
    <suspend-to-disk enabled='no'/>
  </pm>
  <devices>
    <emulator>/usr/bin/kvm-spice</emulator>
    <disk type='file' device='disk'>
      <driver name='qemu' type='qcow2'/>
```

```
    <source file='/home/helena/Tfg/Final/snort/router.qcow2'/>
    <backingStore/>
    <target dev='vda' bus='virtio'/>
    <alias name='virtio-disk0'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x07' function='0x0'/>
  </disk>
  <disk type='file' device='cdrom'>
    <driver name='qemu' type='raw'/>
    <backingStore/>
    <target dev='hda' bus='ide'/>
    <readonly/>
    <alias name='ide0-0-0'/>
    <address type='drive' controller='0' bus='0' target='0' unit='0'/>
  </disk>
  <controller type='pci' index='0' model='pci-root'>
    <alias name='pci.0'/>
  </controller>
  <controller type='ide' index='0'>
    <alias name='ide'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x1'/>
  </controller>
  <controller type='virtio-serial' index='0'>
    <alias name='virtio-serial0'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0'/>
  </controller>
  <controller type='usb' index='0'>
    <alias name='usb'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x2'/>
  </controller>
  <interface type='network'>
    <mac address='52:54:00:ad:95:7b'/>
    <source network='insideNetwork' bridge='virbr0'/>
    <target dev='vnet1'/>
    <model type='virtio'/>
    <alias name='net0'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
  </interface>
  <interface type='network'>
    <mac address='52:54:00:ba:33:4d'/>
    <source network='outsideNetwork' bridge='virbr2'/>
    <target dev='vnet2'/>
    <model type='virtio'/>
    <alias name='net1'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x0'/>
  </interface>
  <interface type='network'>
    <mac address='52:54:00:ea:ad:fc'/>
    <source network='matrix' bridge='virbr3'/>
    <target dev='vnet3'/>
    <model type='virtio'/>
    <alias name='net2'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x09' function='0x0'/>
  </interface>
  <interface type='network'>
    <mac address='52:54:00:71:08:7f'/>
    <source network='management' bridge='virbr1'/>
    <target dev='vnet4'/>
    <model type='virtio'/>
    <alias name='net3'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x0a' function='0x0'/>
  </interface>
  <serial type='pty'>
    <source path='/dev/pts/4'/>
```

```
      <target port='0'/>
      <alias name='serial0'/>
    </serial>
    <console type='pty' tty='/dev/pts/4'>
      <source path='/dev/pts/4'/>
      <target type='serial' port='0'/>
      <alias name='serial0'/>
    </console>
    <channel type='unix'>
      <source mode='bind' path='/var/lib/libvirt/qemu/channel/target/domain-router/org
          .qemu.guest_agent.0'/>
      <target type='virtio' name='org.qemu.guest_agent.0' state='disconnected'/>
      <alias name='channel0'/>
      <address type='virtio-serial' controller='0' bus='0' port='1'/>
    </channel>
    <channel type='spicevmc'>
      <target type='virtio' name='com.redhat.spice.0' state='disconnected'/>
      <alias name='channel1'/>
      <address type='virtio-serial' controller='0' bus='0' port='2'/>
    </channel>
    <input type='tablet' bus='usb'>
      <alias name='input0'/>
    </input>
    <input type='mouse' bus='ps2'/>
    <input type='keyboard' bus='ps2'/>
    <graphics type='spice' port='5901' autoport='yes' listen='127.0.0.1'>
      <listen type='address' address='127.0.0.1'/>
      <image compression='off'/>
    </graphics>
    <sound model='ich6'>
      <alias name='sound0'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x04' function='0x0'/>
    </sound>
    <video>
      <model type='qxl' ram='65536' vram='65536' vgamem='16384' heads='1'/>
      <alias name='video0'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x02' function='0x0'/>
    </video>
    <redirdev bus='usb' type='spicevmc'>
      <alias name='redir0'/>
    </redirdev>
    <redirdev bus='usb' type='spicevmc'>
      <alias name='redir1'/>
    </redirdev>
    <memballoon model='virtio'>
      <alias name='balloon0'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x08' function='0x0'/>
    </memballoon>
  </devices>
  <seclabel type='dynamic' model='apparmor' relabel='yes'>
    <label>libvirt-41f78cf1-dfdf-4ced-a992-9b5e9a73d5b6</label>
    <imagelabel>libvirt-41f78cf1-dfdf-4ced-a992-9b5e9a73d5b6</imagelabel>
  </seclabel>
</domain>
```

A.1.3. Target

```
<domain type='kvm'>
  <name>target</name>
  <uuid>fd081caf-7fb2-4ddd-9d6b-ed65334f4023</uuid>
  <memory unit='KiB'>2097152</memory>
```

```
<currentMemory unit='KiB'>2097152</currentMemory>
<vcpu placement='static'>1</vcpu>
<os>
  <type arch='x86_64' machine='pc-i440fx-xenial'>hvm</type>
  <boot dev='hd'/>
</os>
<features>
  <acpi/>
  <apic/>
</features>
<cpu mode='custom' match='exact'>
  <model fallback='allow'>SandyBridge</model>
</cpu>
<clock offset='utc'>
  <timer name='rtc' tickpolicy='catchup'/>
  <timer name='pit' tickpolicy='delay'/>
  <timer name='hpet' present='no'/>
</clock>
<on_poweroff>destroy</on_poweroff>
<on_reboot>restart</on_reboot>
<on_crash>restart</on_crash>
<pm>
  <suspend-to-mem enabled='no'/>
  <suspend-to-disk enabled='no'/>
</pm>
<devices>
  <emulator>/usr/bin/kvm-spice</emulator>
  <disk type='file' device='disk'>
    <driver name='qemu' type='qcow2'/>
    <source file='/home/helena/Tfg/Final/target/target.qcow2'/>
    <target dev='vda' bus='virtio'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x07' function='0x0'/>
  </disk>
  <disk type='file' device='cdrom'>
    <driver name='qemu' type='raw'/>
    <target dev='hda' bus='ide'/>
    <readonly/>
    <address type='drive' controller='0' bus='0' target='0' unit='0'/>
  </disk>
  <controller type='pci' index='0' model='pci-root'/>
  <controller type='ide' index='0'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x1'/>
  </controller>
  <controller type='virtio-serial' index='0'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0'/>
  </controller>
  <controller type='usb' index='0'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x2'/>
  </controller>
  <interface type='network'>
    <mac address='52:54:00:11:f7:5a'/>
    <source network='insideNetwork'/>
    <model type='virtio'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
  </interface>
  <serial type='pty'>
    <target port='0'/>
  </serial>
  <console type='pty'>
    <target type='serial' port='0'/>
  </console>
  <channel type='unix'>
    <source mode='bind'/>
```

```
      <target type='virtio' name='org.qemu.guest_agent.0'/>
      <address type='virtio-serial' controller='0' bus='0' port='1'/>
    </channel>
    <channel type='spicevmc'>
      <target type='virtio' name='com.redhat.spice.0'/>
      <address type='virtio-serial' controller='0' bus='0' port='2'/>
    </channel>
    <input type='tablet' bus='usb'/>
    <input type='mouse' bus='ps2'/>
    <input type='keyboard' bus='ps2'/>
    <graphics type='spice' autoport='yes'>
      <image compression='off'/>
    </graphics>
    <sound model='ich6'>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x04' function='0x0'/>
    </sound>
    <video>
      <model type='qxl' ram='65536' vram='65536' vgamem='16384' heads='1'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x02' function='0x0'/>
    </video>
    <redirdev bus='usb' type='spicevmc'>
    </redirdev>
    <redirdev bus='usb' type='spicevmc'>
    </redirdev>
    <memballoon model='virtio'>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x08' function='0x0'/>
    </memballoon>
  </devices>
</domain>
```

A.2. Definición de redes virtuales

A continuación se muestran las configuraciones de las cuatro redes empleadas para la implementación.

A.2.1. insideNetwork

Se trata de la red aislada que simula la intranet de cualquier organización o institución. En ella se encuentran las máquinas *router* y *target*.

```
<network>
  <name>insideNetwork</name>
  <uuid>fd320f02-9604-41ec-b77c-3f02bc643f57</uuid>
  <bridge name='virbr0' stp='on' delay='0'/>
  <mac address='52:54:00:61:be:f1'/>
  <domain name='insideNetwork'/>
  <ip address='10.10.10.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='10.10.10.128' end='10.10.10.254'/>
    </dhcp>
  </ip>
</network>
```

A.2.2. outsideNetwork

Es la red que simula la red externa. Presenta conexión con el resto de Internet, a través del *host* mediante NAT. Conectadas a esta red, se encuentran la máquina *attacker* y *router*.

```
<network>
  <name>outsideNetwork</name>
  <uuid>daa97c60-44ea-4e63-bae4-0134796c8471</uuid>
  <forward mode='nat'>
    <nat>
      <port start='1024' end='65535'/>
    </nat>
  </forward>
  <bridge name='virbr2' stp='on' delay='0'/>
  <mac address='52:54:00:08:c7:5a'/>
  <domain name='outsideNetwork'/>
  <ip address='192.168.200.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='192.168.200.128' end='192.168.200.254'/>
    </dhcp>
  </ip>
</network>
```

A.2.3. management

Se trata de la red de gestión, empleada para las comunicaciones entre el *host* y la máquina *router*.

```
<network>
  <name>management</name>
  <uuid>31cce8ff-4888-4d22-95d4-eee343c27db8</uuid>
  <bridge name='virbr1' stp='on' delay='0'/>
  <mac address='52:54:00:ee:5c:ea'/>
  <domain name='management'/>
  <ip address='10.10.20.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='10.10.20.128' end='10.10.20.254'/>
    </dhcp>
  </ip>
</network>
```

A.2.4. matrix

Es la *honeynet*, es decir, la red que albergará las máquinas hacia las que se desviarán las futuras intrusiones.

```
<network>
  <name>matrix</name>
  <uuid>23ceeec6-6f7f-4ef9-b898-254096b68189</uuid>
  <bridge name='virbr3' stp='on' delay='0'/>
  <mac address='52:54:00:d0:97:7f'/>
  <domain name='matrix'/>
  <ip address='192.168.10.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='192.168.10.128' end='192.168.10.254'/>
    </dhcp>
  </ip>
</network>
```

A.3. Configuración de Snort

A.3.1. Fichero /etc/snort/snort.conf

```
#--------------------------------------------------
# VRT Rule Packages Snort.conf
#
# For more information visit us at:
# http://www.snort.org Snort Website
# http://vrt-blog.snort.org/ Sourcefire VRT Blog
#
# Mailing list Contact: snort-sigs@lists.sourceforge.net
# False Positive reports: fp@sourcefire.com
# Snort bugs: bugs@snort.org
#
# Compatible with Snort Versions:
# VERSIONS : 2.9.9.0
#
# Snort build options:
# OPTIONS : --enable-gre --enable-mpls --enable-targetbased --enable-ppm
     --enable-perfprofiling --enable-zlib --enable-active-response --
    enable-normalizer --enable-reload --enable-react --enable-flexresp3
#
# Additional information:
# This configuration file enables active response, to run snort in
# test mode -T you are required to supply an interface -i <interface>
# or test mode will fail to fully validate the configuration and
# exit with a FATAL error
#--------------------------------------------------

###################################################
# This file contains a sample snort configuration.
# You should take the following steps to create your own custom
    configuration:
#
# 1) Set the network variables.
# 2) Configure the decoder
# 3) Configure the base detection engine
# 4) Configure dynamic loaded libraries
# 5) Configure preprocessors
# 6) Configure output plugins
# 7) Customize your rule set
###################################################

###################################################
# Step #1: Set the network variables. For more information, see README.
    variables
###################################################

# Setup the network addresses you are protecting
ipvar HOME_NET 10.10.10.145

# Set up the external network addresses. Leave as "any" in most
    situations
ipvar EXTERNAL_NET any

# List of DNS servers on your network
ipvar DNS_SERVERS $HOME_NET

# List of SMTP servers on your network
ipvar SMTP_SERVERS $HOME_NET
```

```
# List of web servers on your network
ipvar HTTP_SERVERS $HOME_NET

# List of sql servers on your network
ipvar SQL_SERVERS $HOME_NET

# List of telnet servers on your network
ipvar TELNET_SERVERS $HOME_NET

# List of ssh servers on your network
ipvar SSH_SERVERS $HOME_NET

# List of ftp servers on your network
ipvar FTP_SERVERS $HOME_NET

# List of sip servers on your network
ipvar SIP_SERVERS $HOME_NET

# List of ports you run web servers on
portvar HTTP_PORTS
    [80,81,311,383,591,593,901,1220,1414,1741,1830,2301,2381,2809,3037,3128,3702,4343,4848,


# List of ports you want to look for SHELLCODE on.
portvar SHELLCODE_PORTS !80

# List of ports you might see oracle attacks on
portvar ORACLE_PORTS 1024:

# List of ports you want to look for SSH connections on:
portvar SSH_PORTS 22

# List of ports you run ftp servers on
portvar FTP_PORTS [21,2100,3535]

# List of ports you run SIP servers on
portvar SIP_PORTS [5060,5061,5600]

# List of file data ports for file inspection
portvar FILE_DATA_PORTS [$HTTP_PORTS,110,143]

# List of GTP ports for GTP preprocessor
portvar GTP_PORTS [2123,2152,3386]

# other variables, these should not be modified
ipvar AIM_SERVERS
    [64.12.24.0/23,64.12.28.0/23,64.12.161.0/24,64.12.163.0/24,64.12.200.0/24,205.188.3.0/2


# Path to your rules files (this can be a relative path)
# Note for Windows users: You are advised to make this an absolute path,
# such as: c:\snort\rules
var RULE_PATH /etc/snort/rules
var SO_RULE_PATH ../so_rules
var PREPROC_RULE_PATH ../preproc_rules

# If you are using reputation preprocessor set these
# Currently there is a bug with relative paths, they are relative to
    where snort is
# not relative to snort.conf like the above variables
# This is completely inconsistent with how other vars work, BUG 89986
# Set the absolute path appropriately
var WHITE_LIST_PATH ../rules
```

```
var BLACK_LIST_PATH ../rules

##################################################
# Step #2: Configure the decoder. For more information, see README.
    decode
##################################################

# Stop generic decode events:
config disable_decode_alerts

# Stop Alerts on experimental TCP options
config disable_tcpopt_experimental_alerts

# Stop Alerts on obsolete TCP options
config disable_tcpopt_obsolete_alerts

# Stop Alerts on T/TCP alerts
config disable_tcpopt_ttcp_alerts

# Stop Alerts on all other TCPOption type events:
config disable_tcpopt_alerts

# Stop Alerts on invalid ip options
config disable_ipopt_alerts

# Alert if value in length field (IP, TCP, UDP) is greater th elength of
     the packet
# config enable_decode_oversized_alerts

# Same as above, but drop packet if in Inline mode (requires
    enable_decode_oversized_alerts)
# config enable_decode_oversized_drops

# Configure IP / TCP checksum mode
config checksum_mode: all

# Configure default log directory for snort to log to. For more
    information see snort -h command line options (-l)
config logdir: /var/log/logSnort


##################################################
# Step #3: Configure the base detection engine. For more information,
    see README.decode
##################################################

# Configure PCRE match limitations
config pcre_match_limit: 3500
config pcre_match_limit_recursion: 1500

# Configure the detection engine See the Snort Manual, Configuring Snort
     - Includes - Config
config detection: search-method ac-split search-optimize max-pattern-len
     20

# Configure the event queue. For more information, see README.
    event_queue
config event_queue: max_queue 8 log 5 order_events content_length

##################################################
# Configure protocol aware flushing
# For more information see README.stream5
##################################################
```

```
config paf_max: 16000

##################################################
# Step #4: Configure dynamic loaded libraries.
# For more information, see Snort Manual, Configuring Snort - Dynamic
    Modules
##################################################

# path to dynamic preprocessor libraries
dynamicpreprocessor directory /usr/lib64/snort-2.9.9.0
    _dynamicpreprocessor/

# path to base preprocessor engine
dynamicengine /usr/lib64/snort-2.9.9.0_dynamicengine/libsf_engine.so

# path to dynamic rules libraries
#dynamicdetection directory /usr/local/lib/snort_dynamicrules

##################################################
# Step #5: Configure preprocessors
# For more information, see the Snort Manual, Configuring Snort -
    Preprocessors
##################################################

# GTP Control Channle Preprocessor. For more information, see README.GTP
# preprocessor gtp: ports { 2123 3386 2152 }

# Inline packet normalization. For more information, see README.
    normalize
# Does nothing in IDS mode
preprocessor normalize_ip4
preprocessor normalize_tcp: ips ecn stream
preprocessor normalize_icmp4
preprocessor normalize_ip6
preprocessor normalize_icmp6

# Target-based IP defragmentation. For more inforation, see README.frag3
preprocessor frag3_global: max_frags 65536
preprocessor frag3_engine: policy windows detect_anomalies overlap_limit
     10 min_fragment_length 100 timeout 180

# Target-Based stateful inspection/stream reassembly. For more
    inforation, see README.stream5
preprocessor stream5_global: track_tcp yes, \
track_udp yes, \
track_icmp no, \
max_tcp 262144, \
max_udp 131072, \
max_active_responses 2, \
min_response_seconds 5
preprocessor stream5_tcp: log_asymmetric_traffic no, policy windows, \
detect_anomalies, require_3whs 180, \
overlap_limit 10, small_segments 3 bytes 150, timeout 180, \
ports client 21 22 23 25 42 53 79 109 110 111 113 119 135 136 137 139
    143 \
161 445 513 514 587 593 691 1433 1521 1741 2100 3306 6070 6665 6666 6667
     6668 6669 \
7000 8181 32770 32771 32772 32773 32774 32775 32776 32777 32778 32779, \
ports both 80 81 311 383 443 465 563 591 593 636 901 989 992 993 994 995
     1220 1414 1830 2301 2381 2809 3037 3128 3702 4343 4848 5250 6988
    7907 7000 7001 7144 7145 7510 7802 7777 7779 \
7801 7900 7901 7902 7903 7904 7905 7906 7908 7909 7910 7911 7912 7913
    7914 7915 7916 \
```

```
7917 7918 7919 7920 8000 8008 8014 8028 8080 8085 8088 8090 8118 8123
    8180 8243 8280 8300 8800 8888 8899 9000 9060 9080 9090 9091 9443
    9999 11371 34443 34444 41080 50002 55555
preprocessor stream5_udp: timeout 180

# HTTP normalization and anomaly detection. For more information, see
    README.http_inspect
preprocessor http_inspect: global iis_unicode_map unicode.map 1252
    compress_depth 65535 decompress_depth 65535
preprocessor http_inspect_server: server default \
http_methods { GET POST PUT SEARCH MKCOL COPY MOVE LOCK UNLOCK NOTIFY
    POLL BCOPY BDELETE BMOVE LINK UNLINK OPTIONS HEAD DELETE TRACE TRACK
     CONNECT SOURCE SUBSCRIBE UNSUBSCRIBE PROPFIND PROPPATCH BPROPFIND
    BPROPPATCH RPC_CONNECT PROXY_SUCCESS BITS_POST CCM_POST SMS_POST
    RPC_IN_DATA RPC_OUT_DATA RPC_ECHO_DATA } \
chunk_length 500000 \
server_flow_depth 0 \
client_flow_depth 0 \
post_depth 65495 \
oversize_dir_length 500 \
max_header_length 750 \
max_headers 100 \
max_spaces 200 \
small_chunk_length { 10 5 } \
ports { 80 81 311 383 591 593 901 1220 1414 1741 1830 2301 2381 2809
    3037 3128 3702 4343 4848 5250 6988 7000 7001 7144 7145 7510 7777
    7779 8000 8008 8014 8028 8080 8085 8088 8090 8118 8123 8180 8181
    8243 8280 8300 8800 8888 8899 9000 9060 9080 9090 9091 9443 9999
    11371 34443 34444 41080 50002 55555 } \
non_rfc_char { 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 } \
enable_cookie \
extended_response_inspection \
inspect_gzip \
normalize_utf \
unlimited_decompress \
normalize_javascript \
apache_whitespace no \
ascii no \
bare_byte no \
directory no \
double_decode no \
iis_backslash no \
iis_delimiter no \
iis_unicode no \
multi_slash no \
utf_8 no \
u_encode yes \
webroot no

# ONC-RPC normalization and anomaly detection. For more information, see
     the Snort Manual, Configuring Snort - Preprocessors - RPC Decode
preprocessor rpc_decode: 111 32770 32771 32772 32773 32774 32775 32776
    32777 32778 32779 no_alert_multiple_requests
    no_alert_large_fragments no_alert_incomplete

# Back Orifice detection.
preprocessor bo

# FTP / Telnet normalization and anomaly detection. For more information
    , see README.ftptelnet
preprocessor ftp_telnet: global inspection_type stateful
    encrypted_traffic no check_encrypted
preprocessor ftp_telnet_protocol: telnet \
```

```
ayt_attack_thresh 20 \
normalize ports { 23 } \
detect_anomalies
preprocessor ftp_telnet_protocol: ftp server default \
def_max_param_len 100 \
ports { 21 2100 3535 } \
telnet_cmds yes \
ignore_telnet_erase_cmds yes \
ftp_cmds { ABOR ACCT ADAT ALLO APPE AUTH CCC CDUP } \
ftp_cmds { CEL CLNT CMD CONF CWD DELE ENC EPRT } \
ftp_cmds { EPSV ESTA ESTP FEAT HELP LANG LIST LPRT } \
ftp_cmds { LPSV MACB MAIL MDTM MIC MKD MLSD MLST } \
ftp_cmds { MODE NLST NOOP OPTS PASS PASV PBSZ PORT } \
ftp_cmds { PROT PWD QUIT REIN REST RETR RMD RNFR } \
ftp_cmds { RNTO SDUP SITE SIZE SMNT STAT STOR STOU } \
ftp_cmds { STRU SYST TEST TYPE USER XCUP XCRC XCWD } \
ftp_cmds { XMAS XMD5 XMKD XPWD XRCP XRMD XRSQ XSEM } \
ftp_cmds { XSEN XSHA1 XSHA256 } \
alt_max_param_len 0 { ABOR CCC CDUP ESTA FEAT LPSV NOOP PASV PWD QUIT
    REIN STOU SYST XCUP XPWD } \
alt_max_param_len 200 { ALLO APPE CMD HELP NLST RETR RNFR STOR STOU XMKD
     } \
alt_max_param_len 256 { CWD RNTO } \
alt_max_param_len 400 { PORT } \
alt_max_param_len 512 { SIZE } \
chk_str_fmt { ACCT ADAT ALLO APPE AUTH CEL CLNT CMD } \
chk_str_fmt { CONF CWD DELE ENC EPRT EPSV ESTP HELP } \
chk_str_fmt { LANG LIST LPRT MACB MAIL MDTM MIC MKD } \
chk_str_fmt { MLSD MLST MODE NLST OPTS PASS PBSZ PORT } \
chk_str_fmt { PROT REST RETR RMD RNFR RNTO SDUP SITE } \
chk_str_fmt { SIZE SMNT STAT STOR STRU TEST TYPE USER } \
chk_str_fmt { XCRC XCWD XMAS XMD5 XMKD XRCP XRMD XRSQ } \
chk_str_fmt { XSEM XSEN XSHA1 XSHA256 } \
cmd_validity ALLO < int [ char R int ] > \
cmd_validity EPSV < [ { char 12 | char A char L char L } ] > \
cmd_validity MACB < string > \
cmd_validity MDTM < [ date nnnnnnnnnnnnnn[.n[n[n]]] ] string > \
cmd_validity MODE < char ASBCZ > \
cmd_validity PORT < host_port > \
cmd_validity PROT < char CSEP > \
cmd_validity STRU < char FRPO [ string ] > \
cmd_validity TYPE < { char AE [ char NTC ] | char I | char L [ number ]
    } >
preprocessor ftp_telnet_protocol: ftp client default \
max_resp_len 256 \
bounce yes \
ignore_telnet_erase_cmds yes \
telnet_cmds yes


# SMTP normalization and anomaly detection. For more information, see
    README.SMTP
preprocessor smtp: ports { 25 465 587 691 } \
inspection_type stateful \
b64_decode_depth 0 \
qp_decode_depth 0 \
bitenc_decode_depth 0 \
uu_decode_depth 0 \
log_mailfrom \
log_rcptto \
log_filename \
log_email_hdrs \
normalize cmds \
```

```
normalize_cmds { ATRN AUTH BDAT CHUNKING DATA DEBUG EHLO EMAL ESAM ESND
    ESOM ETRN EVFY } \
normalize_cmds { EXPN HELO HELP IDENT MAIL NOOP ONEX QUEU QUIT RCPT RSET
     SAML SEND SOML } \
normalize_cmds { STARTTLS TICK TIME TURN TURNME VERB VRFY X-ADAT X-DRCP
    X-ERCP X-EXCH50 } \
normalize_cmds { X-EXPS X-LINK2STATE XADR XAUTH XCIR XEXCH50 XGEN
    XLICENSE XQUE XSTA XTRN XUSR } \
max_command_line_len 512 \
max_header_line_len 1000 \
max_response_line_len 512 \
alt_max_command_line_len 260 { MAIL } \
alt_max_command_line_len 300 { RCPT } \
alt_max_command_line_len 500 { HELP HELO ETRN EHLO } \
alt_max_command_line_len 255 { EXPN VRFY ATRN SIZE BDAT DEBUG EMAL ESAM
    ESND ESOM EVFY IDENT NOOP RSET } \
alt_max_command_line_len 246 { SEND SAML SOML AUTH TURN ETRN DATA RSET
    QUIT ONEX QUEU STARTTLS TICK TIME TURNME VERB X-EXPS X-LINK2STATE
    XADR XAUTH XCIR XEXCH50 XGEN XLICENSE XQUE XSTA XTRN XUSR } \
valid_cmds { ATRN AUTH BDAT CHUNKING DATA DEBUG EHLO EMAL ESAM ESND ESOM
     ETRN EVFY } \
valid_cmds { EXPN HELO HELP IDENT MAIL NOOP ONEX QUEU QUIT RCPT RSET
    SAML SEND SOML } \
valid_cmds { STARTTLS TICK TIME TURN TURNME VERB VRFY X-ADAT X-DRCP X-
    ERCP X-EXCH50 } \
valid_cmds { X-EXPS X-LINK2STATE XADR XAUTH XCIR XEXCH50 XGEN XLICENSE
    XQUE XSTA XTRN XUSR } \
xlink2state { enabled }

# Portscan detection. For more information, see README.sfportscan
preprocessor sfportscan: proto { all } memcap { 10000000 } sense_level {
     low } scan_type { all } logfile { /var/log/snort/scan/scan.log }

# SSH anomaly detection. For more information, see README.ssh
preprocessor ssh: server_ports { 22 } \
autodetect \
max_client_bytes 19600 \
max_encrypted_packets 20 \
max_server_version_len 100 \
enable_respoverflow enable_ssh1crc32 \
enable_srvoverflow enable_protomismatch

# SMB / DCE-RPC normalization and anomaly detection. For more
    information, see README.dcerpc2
preprocessor dcerpc2: memcap 102400, events [co ]
preprocessor dcerpc2_server: default, policy WinXP, \
detect [smb [139,445], tcp 135, udp 135, rpc-over-http-server 593], \
autodetect [tcp 1025:, udp 1025:, rpc-over-http-server 1025:], \
smb_max_chain 3, smb_invalid_shares ["C$", "D$", "ADMIN$"]

# DNS anomaly detection. For more information, see README.dns
preprocessor dns: ports { 53 } enable_rdata_overflow

# SSL anomaly detection and traffic bypass. For more information, see
    README.ssl
preprocessor ssl: ports { 443 465 563 636 989 992 993 994 995 7801 7802
    7900 7901 7902 7903 7904 7905 7906 7907 7908 7909 7910 7911 7912
    7913 7914 7915 7916 7917 7918 7919 7920 }, trustservers,
    noinspect_encrypted

# SDF sensitive data preprocessor. For more information see README.
    sensitive_data
preprocessor sensitive_data: alert_threshold 25
```

```
# SIP Session Initiation Protocol preprocessor. For more information see
     README.sip
preprocessor sip: max_sessions 40000, \
ports { 5060 5061 5600 }, \
methods { invite \
cancel \
ack \
bye \
register \
options \
refer \
subscribe \
update \
join \
info \
message \
notify \
benotify \
do \
qauth \
sprack \
publish \
service \
unsubscribe \
prack }, \
max_uri_len 512, \
max_call_id_len 80, \
max_requestName_len 20, \
max_from_len 256, \
max_to_len 256, \
max_via_len 1024, \
max_contact_len 512, \
max_content_len 2048

# IMAP preprocessor. For more information see README.imap
preprocessor imap: \
ports { 143 } \
b64_decode_depth 0 \
qp_decode_depth 0 \
bitenc_decode_depth 0 \
uu_decode_depth 0

# POP preprocessor. For more information see README.pop
preprocessor pop: \
ports { 110 } \
b64_decode_depth 0 \
qp_decode_depth 0 \
bitenc_decode_depth 0 \
uu_decode_depth 0

# Modbus preprocessor. For more information see README.modbus
preprocessor modbus: ports { 502 }

# DNP3 preprocessor. For more information see README.dnp3
preprocessor dnp3: ports { 20000 } \
memcap 262144 \
check_crc

##################################################
# Step #6: Configure output plugins
# For more information, see Snort Manual, Configuring Snort - Output
    Modules
```

```
##################################################

# syslog
output alert_syslog: LOG_LOCAL0 LOG_ALERT LOG_NDELAY

# metadata reference data. do not modify these lines
include classification.config
include reference.config


##################################################
# Step #7: Customize your rule set
# For more information, see Snort Manual, Writing Snort Rules
#
# NOTE: All categories are enabled in this conf file
##################################################

# site specific rules
include $RULE_PATH/local.rules
```

Como puede verse, se trata de un fichero muy extenso, con multitud de parámetros que configurar. No obstante, cabe destacar las siguientes opciones:

- **ipvar HOME_NET**: esta variable especifica el rango de IPs a proteger.
- **ipvar EXTERNAL_NET**: determina que se considera la red externa. Con la palabra *any* cualquier IP no incluida en *HOME_NET* es externa.
- **config logdir**: esta directriz fija la ruta del fichero de *logs* que utilizará Snort para registrar las alertas.
- **preprocessor sfportscan**: en este apartado se fija la regla que utilizará el preprocesador sfportscan para detectar escaneos de puertos.
- **output alert_syslog**: en esta línea se configura el módulo de salida de syslog, para integrarlo con Snort.
- **include**: determina los ficheros de reglas a incluir. En este caso, el fichero *local.rules* donde se han escrito las reglas para detectar los ataques estudiados[1].

A.3.2. Configuración de Snort como servicio

A continuación se mostrarán los ficheros desarrollados para que Snort se ejecute como un servicio desde que se produzca el arranque de la máquina. Dado que sistema operativo es un CentOS 7, los servicios son gestionados por *systemd*[2]. Para configurar un nuevo servicio es necesario añadir un fichero, denominado *unit* o unidad, para definirlo en el directorio */etc/systemd/system*. En este caso el fichero se ha denominado *snort.service*

```
[Unit]
Description=script to start snort
After=network.target


[Service]
```

[1] Véase el capítulo 5.

[2] Se trata de un software capaz de gestionar distintos aspectos de la máquina como servicios, dispositivos o estados del sistema.

```
Type=simple
ExecStart=/usr/bin/tfg/startSnort.sh
TimeoutStartSec=0


[Install]
WantedBy=default.target
```

En este fichero se especifica cuando ha de arrancar el servicio, despúes de haberse realizado la configuración de red, así como el script a ejecutar para arrancarlo, */usr/-bin/tfg/startSnort.sh*. El contenido de este *script* es el siguiente:

```
#!/bin/bash
echo "Iniciando snort" > /var/log/snort/snortBoot.log

nohup /usr/sbin/snort -c /etc/snort/snort.conf

echo "Snort se ha iniciado" >> /var/log/snort/snortBoot.log
```

Se trata de un *script* muy sencillo en el que únicamente se ejecuta Snort con el comando *nohup* que evita que se cierre al terminar de ejecutarse el script.

A.4. Configuración del enrutado con iptables

Para que la máquina virtual *Router* funcione como tal es necesario que todo el tráfico que recibe por la interfaz eth1, conectada al exterior sea reenviada hacia la red interna por la interfaz eth0. Al igual que ocurría con Snort se ha configurado con un *script* que se ejecuta al arranque del sistema para cargar la configuración correcta en *iptables*.

A.4.1. Unidad iptsetup.service

El fichero añadido en */etc/systemd/system* es el siguiente:

```
[Unit]
Description=Script to set up iptables
After=network.target


[Service]
Type=simple
ExecStart=/usr/bin/tfg/iptsetup.sh
TimeoutStartSec=0


[Install]
WantedBy=default.target
```

El contenido de este fichero es muy similar al de Snort, con la salvedad de que ejecuta un nuevo *script*, *iptsetup.sh*:

```
#!/bin/bash
```

```
echo "This is a script to set up the configuration of iptables and
    enable forwarding" > /var/tmp/iptsetup.log
echo "The time the script run was --> ‘date‘" >> /var/tmp/iptsetup.log

IPT_CONFIG="/usr/lib/iptables_config.txt"

iptables-restore $IPT_CONFIG


echo 1 > /proc/sys/net/ipv4/ip_forward


echo "iptables set up correctly"
```

Con el comando *iptables-restore* se carga la configuración contenida en el fichero *iptables_ config.txt*, mientras que escribiendo un 1 en el fichero *ip_ forward* se permite el reenvío de paquetes.

A.5. Configuración de syslog-ng

La configuración de syslog-ng se encuentra en el fichero */etc/syslog-ng/syslog-ng.conf*:

```
@version:3.5
@include "scl.conf"

# syslog-ng configuration file.
#
# This should behave pretty much like the original syslog on RedHat. But
# it could be configured a lot smarter.
#
# See syslog-ng(8) and syslog-ng.conf(5) for more information.
#
# Note: it also sources additional configuration files (*.conf)
# located in /etc/syslog-ng/conf.d/

options {
flush_lines (0);
time_reopen (10);
log_fifo_size (1000);
chain_hostnames (off);
use_dns (no);
use_fqdn (no);
create_dirs (no);
keep_hostname (yes);
};

source s_sys {
system();
internal();
# udp(ip(0.0.0.0) port(514));
};

source s_file{ file("/var/log/snort/scan/scan.log"); };

destination d_cons { file("/dev/console"); };
destination d_mesg { file("/var/log/messages"); };
destination d_auth { file("/var/log/secure"); };
destination d_mail { file("/var/log/maillog" flush_lines(10)); };
destination d_spol { file("/var/log/spooler"); };
```

```
destination d_boot { file("/var/log/boot.log"); };
destination d_cron { file("/var/log/cron"); };
destination d_kern { file("/var/log/kern"); };
destination d_mlal { usertty("*"); };

template t_snort {
template("${ISODATE} ${HOST} ${MESSAGE}\n"); };

destination d_snort { file("/var/log/snort/snort.log" template(t_snort))
    ; };
destination d_pre_snort { file("/var/log/snort/snort.log"); };



filter f_kernel { facility(kern); };
filter f_default { level(info..emerg) and
not (facility(mail)
or facility(authpriv)
or facility(cron)); };
filter f_auth { facility(authpriv); };
filter f_mail { facility(mail); };
filter f_emergency { level(emerg); };
filter f_news { facility(uucp) or
(facility(news)
and level(crit..emerg)); };
filter f_boot { facility(local7); };
filter f_cron { facility(cron); };
filter f_snort { facility(local0); };



#log { source(s_sys); filter(f_kernel); destination(d_cons); };
log { source(s_sys); filter(f_kernel); destination(d_kern); };
log { source(s_sys); filter(f_default); destination(d_mesg); };
log { source(s_sys); filter(f_auth); destination(d_auth); };
log { source(s_sys); filter(f_mail); destination(d_mail); };
log { source(s_sys); filter(f_emergency); destination(d_mlal); };
log { source(s_sys); filter(f_news); destination(d_spol); };
log { source(s_sys); filter(f_boot); destination(d_boot); };
log { source(s_sys); filter(f_cron); destination(d_cron); };
log { source(s_sys); filter(f_snort); destination(d_snort); };
log { source(s_file); destination(d_pre_snort); };

# Source additional configuration files (.conf extension only)
@include "/etc/syslog-ng/conf.d/*.conf"


# vim:ft=syslog-ng:ai:si:ts=4:sw=4:et:
```

Al fichero por defecto se le han añadido los siguientes objetos:

- `source s_file{ file("/var/log/snort/scan/scan.log"); };`

  Indica la fuente de la que se leen los mensajes.

- ```
  template t_snort {
  template("${ISODATE} ${HOST} ${MESSAGE}\n"); };
  ```

  Se trata de una plantilla que se aplica sobre los mensajes, de manera que el formato sea fecha, *host* y contenido.
  ```

- `destination d_snort { file("/var/log/snort/snort.log" template(t_snort)); };`

  Determina un fichero de destino y aplica la plantilla anterior.

- `destination d_pre_snort { file("/var/log/snort/snort.log"); };`

  Fija un fichero de destino para los mensajes del preprocesador.

- `filter f_snort { facility(local0); };`

  Filtra los mensajes procedentes de la facilidad del sistema *local0*.

Una vez creados estos objetos, se registran los *logs* mediante las siguientes directrices:

- `log { source(s_sys); filter(f_snort); destination(d_snort); };`

  Redirige las alertas generadas por el motor de detección de Snort.

- `log { source(s_file); destination(d_pre_snort); };`

  Redirige las alertas generadas por el preprocesador de Snort.

A.6. *Script* en Python eventDetection.py

Este *script* en *Python* controla las modificaciones que se producen en los ficheros que registran los logs de alarma, leyendo su contenido y gestionando las nuevas alarmas. Hace uso de una librería no estándar, denominada *watchdog*, que permite monitorizar el estado de los ficheros de un directorio.
En primer lugar, se crea una instancia de los objetos *Observer* y *Event_ Handler*, que se encargan de monitorizar posibles eventos y gestionarlos, respectivamente. Cuando el fichero en cuestión es modificado, se lee lo último escrito gracias a la existencia de una variable que almacena la última posición del cursor. Si de esta lectura se extrae una alarma, se comprueba que la dirección IP que la originó no ha sido registrada aún en el array *attackers*. De ser así, la alarma se gestiona ejecutando un comando de manera remota por ssh en el *host*.

```
#!/usr/bin/env python

# coding=utf-8

#Script to read messages from syslog


import time
import sys
import logging
import logging.handlers
import subprocess
import os.path

from watchdog.observers import Observer
from watchdog.events import FileSystemEventHandler
import watchdog.events

class ScanLogging:

'Class forwarding scanning alerts to rsyslog'


# Metodo principal init. Se crea un handler que esperare la modificacion de ficheros
    en el directorio especificado. Desde aqui se llama al resto de metodos.
```

```
def __init__(self,dir,file):
  self.log_file = '/var/log/snort/event.log'
  self.write_log("######## Inicializando ############")
  self.file = file
  self.cleanEnv()
  self.event_handler = watchdog.events.FileSystemEventHandler()
  self.event_handler.on_modified = self.on_modified
  self.observer = Observer()
  self.observer.schedule(self.event_handler,path=dir, recursive=True)
  self.observer.start()
  print 'Waiting for event'
  self.filePos = None
  self.attackers = []



#Metodo que se ejecuta cuando se modifica unos de los ficheros del directorio
    especificado. Se comprueba el fichero, para leerlo y procesar el ataque.

def on_modified(self, event):
  W = '\033[0m'
  R = '\033[31m'
  if event.src_path != self.file:
    self.write_log("El fichero modificado no es el deseado")
    return
  attackerIp, targetIp,empty = self.readFile()
  if empty != None:
    self.write_log("Error al leer el fichero")
  return
  if self.checkAttackers(attackerIp):
    print "El ataque ya ha sido gestionado"
    return
  else:
    print (R+"SE HA DETECTADO UN NUEVO ATAQUE!!!!"+W)
    cmd = "ssh helena@host /bin/bash /home/helena/Tfg/Scripts/launchMachine.sh" + ' '
         + attackerIp + ' ' + targetIp
    subprocess.Popen(cmd,shell=True,executable='/bin/bash')
    print 'Ejecutado comando en host remoto'


#Metodo encargado de leer la ultima modificacion del fichero a partir de un cursor

def readFile(self):
  messages = []
  with open(self.file,'r') as fp:
    if self.filePos is None:
      self.filePos = fp.tell()
    fp.seek(self.filePos)
    content = fp.read()
    if content == '':
      return None,None,1
    ip1,ip2 = self.process_content(content)
    self.filePos = fp.tell()
  fp.close()
  return ip1,ip2,None



#Metodo que comprueba si se ha producido una alarma y en ese caso devuelve las IPs
    origen del ataque y destino.

def process_content(self,content):
```

```
  lines = content.split('\n')
  ip1 = ''
  ip2 = ''
  for line in lines:
    self.write_log(line)
    if '->' not in line:
      continue
    else:
      if 'Portscan' in line:
        cmd="systemctl restart snort"
        subprocess.Popen(cmd,shell=True)
      ip1,ip2 = self.getIpAddress(line)
  return ip1, ip2



# Metodo que limpia el directorio de ficheros durante la inicializacion

def cleanEnv(self):
  comm='/usr/bin/tfg/startServices.sh'
  subprocess.call(comm,shell=True)
  if os.path.isfile(self.file):
    comm='rm -f ' + self.file
    subprocess.call(comm,shell=True)
    while os.path.isfile(self.file):
      pass
    self.write_log("Se ha borrado el fichero")
  comm = 'touch ' + self.file
  subprocess.call(comm,shell=True)
  self.write_log("Se ha creado el fichero")



#Metodo que parsea las lineas que contienen las IPs en las alarmas escritas.

def getIpAddress(self,message):
  if 'ssh' in message:
  self.write_log("Alerta por ataque al puerto ssh")
    parts = message.split("->")
    tmp1 = parts[0].rstrip()
    tmp2 = parts[1].rstrip()
    tmp3 = tmp1.split(" ")
    tmp4 = tmp2.split(" ")
    aux1 = tmp3[len(tmp3) - 1]
    aux2 = tmp4[1]
    aux3 = aux1.split(":")
    aux4 = aux2.split(":")
    ip1 = aux3[0]
    ip2 = aux4[0]
  else:
    parts = message.split("->")
    tmp1 = parts[0].rstrip()
    tmp2 = parts[1].rstrip()
    tmp3 = tmp1.split(" ")
    tmp4 = tmp2.split(" ")
    ip1 = tmp3[len(tmp3) - 1]
    ip2 = tmp4[1]
  self.write_log("Ip atacante"+ip1)
  self.write_log("Ip destiono"+ip2)
  return ip1, ip2
```

```
#Metodo que verifica si se ha producido alguna coincidencia con alguna de las IPs
    atacantes ya registradas.

def checkAttackers(self,ip):
  for attacker in self.attackers:
    self.write_log(attacker)
    if attacker == ip:
      return True
  self.attackers.append(ip)
  return False


#Metodo que libera el handler y para la el proceso que escucha las modificaciones.

def stop(self):
  self.observer.stop()
  self.observer.join()

#Metodo que escribe en el fichero de logs del script

def write_log(self,log):
  with open(self.log_file,'a') as f:
    f.write(log)
  f.close()

#Instanciacion de la clase ScanLogging

scanAlert = ScanLogging('/var/log/snort','/var/log/snort/snort.log')


#El programa se ejecuta hasta que se produzca una interrupcion del teclado.

try:
  while True:
    time.sleep(1)
except KeyboardInterrupt:
  print 'Closing programme'

scanAlert.stop()
```

A.7. *Script* en bash launchMachine.sh

El script que se muestra a continuación se ejecuta desde la máquina virtual *router* para lanzar una nueva instancia en la *honeynet* a la que desviar la intrusión. Se trata de un script, compuesto por varias funciones, en las que se realizan secuencialmente todas las tareas necesarias para arrancar la nueva máquina.

- **func_check_vm_num** Se comprueba si puede lanzarse una nueva maquina virtual.
- **func_clone_machine $machine** En caso de ser posible, se arranca una nueva VM, mediante clonación a partir de un template.
- **func_getIp** Se obtiene la IP de la nueva maquina levantada a partir de su dirección MAC.
- **func_setup_route** Se configura el enrutamiento estático en la nueva instancia, de manera que su salida por defecto sea la máquina *router*.

- **func_exec_remote_cmd** Se desvía el tráfico atacante hacia la máquina recién creada mediante una regla de iptables en la máquina *router*.

```
#!/bin/bash

#Definicion de variables
#Prefijo de la nueva maquina
machine="newVM_"
#Path a la plantilla que sirve de base para crear una nueva maquina
pathTemplate="/home/helena/Tfg/Final/matrix/templates/centosTemplate.xml"
#Path a la imagen de la nueva maquinva virtual
pathImage="/home/helena/Tfg/Final/matrix/images/"
#Fichero que guarda la configuracion de la VM
file="/home/helena/Tfg/Scripts/dumpedXml.xml"
#Nombre de red de la nueva maquina virtual
net="matrix"
#Ip atacante, recibida como primer parametro
attackerIp=$1
targetIp=$2
#Constante que fija el numero maximo de instancias levantadas
MAX_VM=2


#Funcion que comprueba si ya hay un proceso launchMachine ejecutandose

func_check_process(){
  pss=$(ps -waux | grep -v grep | grep "launchMachine.sh" | wc -l)
  if [ $pss -gt $MAX_PS ]
  then
    echo "ya hay un proceso levantado"
    exit 0
  else
    return 0
  fi
}


#Funcion que comprueba las VMs levantadas y asigna un nuevo nombre de maquina o elige
     una de las que ya estan levantadas

func_check_vm_num(){
  echo "Exec: func_check_vm_num"
  #comprueba el numero de maquinas virtuales levantadas
  count=$(ls $pathImage | wc -l)
  echo $count
  #Se supera MAX_VM se elige una maquina al azar para desviar la intrusion
  if [ $count -ge $MAX_VM ]
  then
    echo "Ya se ha alcanzo el maximo numero de maquinas virtuales"
    echo "Se eligira una al azar"
    num=$(( ( RANDOM % 2 ) + 1 ))
    machine=$machine$num
    return 1
  else
    ((count++))
    machine=$machine$count
    return 2
  fi
}
```

```
#Funcion que lanza una nueva VM, clonandola de una plantilla

func_clone_machine(){

  echo "Exec: func_clone_machine"
  newVM="$1"

  #Clone machine from template
  virt-clone --original-xml $pathTemplate --name $newVM --file ${pathImage}${newVM}

  #Se muestra la informacion de la nueva maquina virtual
  echo "Getting information of new machine"
  virsh dominfo $newVM
  #Se arranca la nueva maquina
  virsh start $newVM
}

# Funcion para averiguar la nueva IP de la maquina creada

func_getIp(){
  #echo "Exec: func_getIp"
  newVM="$1"

  #Se extrae la IP a partir de la direccion MAC, que existe desde la creacion de la
      maquina

  virsh dumpxml $newVM > $file
  mac=$(xmllint --xpath 'string(//domain/devices/interface/mac/@address)' $file )


  #En este punto se espera a que la nueva maquina reciba una IP

  tmp=$(virsh net-dhcp-leases $net $mac)
  until [[ "$tmp" =~ $mac ]]; do
    tmp=$(virsh net-dhcp-leases $net $mac)
    sleep 1
  done

  ip=$(virsh net-dhcp-leases $net $mac | awk '$5 ~ "192" { print $5 }' | cut -d'/' -f
       1)
}

# Funcion para configurar la ruta por defecto en la nueva maquina

func_setup_route(){
  SUBNET="$1"
  ROUTER="$2"
  MACHINE="$3"

  #Los comandos se ejecutan remotamente en la nueva VM
  ssh -o StrictHostKeyChecking=no root@$MACHINE ip addr add $SUBNET dev eth0
  ssh -o StrictHostKeyChecking=no root@$MACHINE ip r add default via $ROUTER

}

#Funcion para configurar iptables de manera remota en el router

func_exec_remote_cmd(){
  attackIp="$1"
  dstIp="$2"

  ssh root@router /bin/bash /usr/bin/addIpRule.sh $attackIp $dstIp
```

```
  echo "liberar conexiones a la maquina target"
  ssh root@router /usr/sbin/conntrack -D -s $attackIp -d $targetIp
}

#Funcion principal

main(){
  func_check_process
  echo "Exec: main"
  func_check_vm_num
  if [ $? = 2 ]
  then
    func_clone_machine $machine
  fi
  func_getIp $machine
  newIp=$(func_getIp $machine)
  echo $newIp
  func_setup_route "192.168.10/24" "192.168.10.150" "$newIp"
  func_exec_remote_cmd $attackerIp $newIp
  echo "Execution completed"
}

# Volcado de los logs en el fichero /tmp/launchMachine.log

main "$&" >> /tmp/launchMachine.log
```

A.8. Plantilla para la creación de una nueva máquina virtual

El *template* que se muestra a continuación se ha extraído de la máquina *targe*, tal y como puede comprobarse a partir del atributo *name*, que será sobreescrito en el momento en que se produzca la clonación. De hecho, esta plantilla es en realidad una definición de la máquina *target* sobre la que se han modificado los siguientes aspectos:

- **uuid** Desaparece este atributo, que será asignado de forma automática cuando se cree una nueva máquina.
- **disk** Se utilizará como disco duro una imagen base de centos en la que ya se encuentran configuradas las claves ssh necesarias para la comunicación entre *host* y *router*.
- **interface** La máquina se conecta a la red *matrix* y, además, se elimina la dirección MAC de manera que se le asigne una única durante el proceso de creación.

```
<domain type='kvm'>
  <name>target</name>
  <memory unit='KiB'>2097152</memory>
  <currentMemory unit='KiB'>2097152</currentMemory>
  <vcpu placement='static'>1</vcpu>
  <os>
    <type arch='x86_64' machine='pc-i440fx-xenial'>hvm</type>
    <boot dev='hd'/>
  </os>
  <features>
    <acpi/>
    <apic/>
  </features>
```

```
<cpu mode='custom' match='exact'>
  <model fallback='allow'>SandyBridge</model>
</cpu>
<clock offset='utc'>
  <timer name='rtc' tickpolicy='catchup'/>
  <timer name='pit' tickpolicy='delay'/>
  <timer name='hpet' present='no'/>
</clock>
<on_poweroff>destroy</on_poweroff>
<on_reboot>restart</on_reboot>
<on_crash>restart</on_crash>
<pm>
  <suspend-to-mem enabled='no'/>
  <suspend-to-disk enabled='no'/>
</pm>
<devices>
  <emulator>/usr/bin/kvm-spice</emulator>
  <disk type='file' device='disk'>
    <driver name='qemu' type='qcow2'/>
    <source file='/home/helena/Tfg/Final/target/centos.qcow2'/>
    <target dev='vda' bus='virtio'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x07' function='0x0'/>
  </disk>
  <disk type='file' device='cdrom'>
    <driver name='qemu' type='raw'/>
    <target dev='hda' bus='ide'/>
    <readonly/>
    <address type='drive' controller='0' bus='0' target='0' unit='0'/>
  </disk>
  <controller type='pci' index='0' model='pci-root'/>
  <controller type='ide' index='0'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x1'/>
  </controller>
  <controller type='virtio-serial' index='0'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0'/>
  </controller>
  <interface type='network'>
    <source network='matrix'/>
    <model type='virtio'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
  </interface>
  <serial type='pty'>
    <target port='0'/>
  </serial>
  <console type='pty'>
    <target type='serial' port='0'/>
  </console>
  <channel type='unix'>
    <source mode='bind'/>
    <target type='virtio' name='org.qemu.guest_agent.0'/>
    <address type='virtio-serial' controller='0' bus='0' port='1'/>
  </channel>
  <channel type='spicevmc'>
    <target type='virtio' name='com.redhat.spice.0'/>
    <address type='virtio-serial' controller='0' bus='0' port='2'/>
  </channel>
  <input type='tablet' bus='usb'/>
  <input type='mouse' bus='ps2'/>
  <input type='keyboard' bus='ps2'/>
  <graphics type='spice' autoport='yes'>
    <image compression='off'/>
  </graphics>
  <sound model='ich6'>
```

```
      <address type='pci' domain='0x0000' bus='0x00' slot='0x04' function='0x0'/>
    </sound>
    <video>
      <model type='qxl' ram='65536' vram='65536' vgamem='16384' heads='1'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x02' function='0x0'/>
    </video>
    <redirdev bus='usb' type='spicevmc'>
    </redirdev>
    <redirdev bus='usb' type='spicevmc'>
    </redirdev>
    <memballoon model='virtio'>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x08' function='0x0'/>
    </memballoon>
  </devices>
</domain>
```

Bibliografía

[1] William Stallings. *Network Security Essentials*. Harlow, United Kingdom: Pearson Education Limited, 2016.

[2] Cisco Systems. *What Is Network Security?* URL: https://www.cisco.com/c/en/us/products/security/what-is-network-security.html.

[3] Comité Consultivo Internacional Telegráfico y Telefónico. *Recomendacion X.800*. Unión Internacional de Telecomunicaciones, 1991.

[4] R.Shirey. *RFC 4949*. IETF, 2007.

[5] Verizon Communications Inc. *2017 Data Breach Investigations Report*. Inf. téc. VerizonEnterprise.

[6] Cisco Systems. *Cisco 2017 Annual Cybersecurity Report*. Inf. téc. Cisco Systems, Inc., 2017.

[7] Nmap Project. *SecTools.Org: Top 125 Network Security Tools*. 2018. URL: http://sectools.org/?sort=rank.

[8] Wireshark Foundation. *About Wireshark*. URL: https://www.wireshark.org/.

[9] Inc Rapid7. *Metasploit. The world's most used penetration testing framework*. URL: https://www.metasploit.com/.

[10] Inc Tenable. *Nessus Professional*. URL: https://www.tenable.com/products/nessus/nessus-professional.

[11] Aircrack-ng. *Aircrack-ng*. URL: https://www.aircrack-ng.org/.

[12] Cisco Systems. *Snort*. URL: https://www.snort.org/.

[13] Massimiliano Montoro. *Cain & Abel - User Manual*. URL: http://www.oxid.it/ca_um/.

[14] Offensive Security. *BackTrack*. URL: https://www.backtrack-linux.org/.

[15] Nmap.org. *Ncat*. URL: https://nmap.org/ncat/.

[16] Tcpdump.org. *Manpage of TCPDUMP*. URL: https://www.tcpdump.org/tcpdump_man.html.

[17] Openwall Project. *John the Ripper password cracker*. URL: http://www.openwall.com/john/.

[18] Seguridad en Sistemas y Técnicas de Hacking. TheHackerWay. *HoneyPots Parte 1 – Kippo*. 2015. URL: https://thehackerway.com/2015/03/24/honeypots-parte-1-kippo/.

[19] Jamie Riden. *Know Your Enemy: Malicious Web Servers*. 2008. URL: http://www.honeynet.org/papers/mws.

[20] Herbert Bos. *Shelia: a client-side honeypot for attack detection*. URL: http://www.cs.vu.nl/%7Eherbertb/misc/shelia/ (visitado 10-03-2018).

[21] Kemmerer & Vigna. «Intrusion detection: a brief history and overview». En: *Computer* 35.4 (2002).

[22] OpenNebulaConf 2016, ed. *HIDS (Host-Based Intrusion Detection System)*.

[23] Mohammed Farik Rajesh Vuppala. «Intrusion Detection & Prevention Systems - Sourcefire Snort». En: *International Journal of Scientific & Technology Research* 5 (2016). URL: http://www.ijstr.org/final-print/july2016/Intrusion-Detection-Prevention-Systems-Sourcefire-Snort.pdf.

[24] OISF. *Suricata User Guide*. 2016. URL: http://suricata.readthedocs.io/en/suricata-4.0.4/.

[25] Foster & Posluns. «Snort: The Inner Workings». En: *Snort intrusion detection 2.0*. 2003.

[26] Shimonski & Shinder. «Chapter 29 Introducing Snort». En: *The best damn firewall book period*. 2003.

[27] Cisco Systems. *SNORT Users Manual*. The Snort Project.

[28] OISF. *Suricata Tutorial, FloCon 2016*. 2016. URL: https://resources.sei.cmu.edu/asset_files/Presentation/2016_017_001_449890.pdf.

[29] Ashok & Manikrao. «Comparative Study and Analysis of Network Intrusion Detection Tools». En: *2015 International Conference on Applied and Theoretical Computing and Communication Technology*.

[30] Arafat Ali Noshy Ibrahim. «Optimization of live virtual machine migration in cloud computing: A survey and future directions». En: *Journal of Network and Computer Applications* (2018).

[31] Inc VMware. *ESXi*. URL: https://www.vmware.com/es/products/esxi-and-esx.html.

[32] Linux-kvm.org. *Kernel Virtual Machine*. URL: https://www.linux-kvm.org/page/Main_Page.

[33] Microsoft Corporation. *Hyper-V*. URL: https://es.wikipedia.org/wiki/Microsoft.

[34] Oracle Corporation. *VirtualBox*. URL: https://www.virtualbox.org/.

[35] Inc VMware. *Workstation Pro*. URL: https://www.vmware.com/es/products/workstation-pro.html.

[36] Michael Gabriel. *What is a Hypervisor?* Ed. por Pluralsight LLC. 2013. URL: https://www.pluralsight.com/blog/it-ops/what-is-hypervisor.

[37] Inc Red Hat. *¿Qué es la virtualización?* Ed. por Inc Red Hat. 2018. URL: https://www.redhat.com/es/topics/virtualization/what-is-virtualization.

[38] Oracle Corporation. *OracleVM User's Guide for Release 3.0.3*. Ed. por Oracle. Cap. Introduction to Virtualization.

[39] Inc Red Hat. *Why Use Virtualization?* Ed. por Inc Red Hat. 2018. URL: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/virtualization_getting_started_guide/chap-virtualization_getting_started-advantages.

[40] Thecloudguytim's Blog. *Five Levels Of Virtualization*. 2010. URL: https://thecloudguytim.wordpress.com/2010/09/01/5-levels-of-virtualization/.

[41] Inc VMware. *Horizon 7*. URL: https://www.vmware.com/es/products/horizon.html.

[42] Amit Shah. *Ten years of KVM*. Ed. por LWN.net. 2016. URL: https://lwn.net/Articles/705160/.

[43] Qemu.org. *What is QEMU?* URL: https://www.qemu.org/.

[44] Linux-kvm.org. *KVM-Features*. URL: https://www.linux-kvm.org/page/KVM_Features.

[45] Inc Red Hat, ed. *¿Qué es DOCKER*. URL: https://www.redhat.com/es/topics/containers/what-is-docker.

[46] Linuxcontainers.org. *Infrastructure for container projects*. URL: https://linuxcontainers.org/.

[47] Docker Inc, ed. *Docker overview*. 2018. URL: https://docs.docker.com/engine/docker-overview/.

[48] Jason Soto. *Meet Docker*. 2016. URL: https://jsitech1.gitbooks.io/meet-docker/arquitectura_de_docker.html.

[49] Jain Bhardwaj Jain. «Cloud computing: a study of infraestructure as a Service (IAAS)». En: *International Journal of Engineering and Information Technology* (2010).

[50] Inc Red Hat. *El concepto de OpenStack*. URL: https://www.redhat.com/es/topics/openstack.

[51] Opensource.com. *What is OpenStack?* URL: https://opensource.com/resources/what-is-openstack.

[52] Wikipedia. *OpenNebula*. 2018. URL: https://en.wikipedia.org/wiki/OpenNebula.

[53] OpenNebula.systems. *OpenNebula Key Features*. URL: http://docs.opennebula.org/5.4/intro_release_notes/concepts_terminology/key_features.html#opennebula-key-features.

[54] OpenNebula.systems. *Open Cloud Reference Architecture*. 2015.

[55] OpenNebulaConf 2016, ed. *Hypervisors and Containers*. 2016.

[56] Luca Foschini Fabio Bracci Antonio Corradi. «Database Security Management for Healthcare SaaS in the Amazon AWS Cloud». En: *Computers and Communications (ISCC), 2012 IEEE Symposium on*. Ed. por IEEE. 2012.

[57] Kali Linux. *Kali*. 2018. URL: https://www.kali.org/.

[58] The CentOS Project. *The CentOS Project*. 2018. URL: https://www.centos.org/.

[59] Balabit Corp. *Documentation Syslog-ng Open Source Edition*. URL: https://syslog-ng.com/documents/html/syslog-ng-ose-latest-guides/en/syslog-ng-ose-guide-admin/html/so-contents.html.

[60] Netfilter.org. *The netfilter.org project*. URL: https://www.netfilter.org/.

[61] Chris Grundemann. *NAT444 (CGN/LSN) and What it Breaks*. URL: https://chrisgrundemann.com/index.php/2011/nat444-cgn-lsn-breaks/.