Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
(Mención en Tecnologías de la Información)

Desarrollo de una honeynet en la UVa para la investigación de ataques informáticos

Autor:

D. Luis Tomé Urgellés

Agradecimientos

Este proyecto no se habría llevado a cabo de no ser por todas esas personas que han estado a mi lado durante el transcurso de la carrera.

A los compañeros y amigos, los que ya tenía y los que he ido conociendo, por los buenos momentos y su compañía.

A los profesores y en especial a mi tutor por darme los conocimientos y guiarme por el mundo de la informática.

Y por último, a las tres personas más importantes de mi vida: mis padres y mi hermana. Por su apoyo emocional y económico como sus sabios consejos durante mi carrera académica.

Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
(Mención en Tecnologías de la Información)

Desarrollo de una honeynet en la UVa para la investigación de ataques informáticos

Autor:

D. Luis Tomé Urgellés

Tutores:

D. Blas Torregrosa García

Resumen

El principal objetivo de este trabajo de fin de grado es implementar y diseñar una red HoneyNet virtual. Se pretende con ello comprender el funcionamiento de esta red en una infraestructura existente como es la Universidad de Valladolid.

La idea del proyecto nace de un incremento de ciberataques durante estos últimos años a empresas privadas e instituciones públicas que conllevan pérdidas millonarias o información confidencial. El objetivo de los atacantes siempre son los activos de las empresas almacenados en servidores y equipos, siendo estos los que reciben la mayoría de vectores de ataque. En consecuencia, para evitar estos ataques, se diseñó expresamente una red expuesta al público con el objetivo de engañar a los ciberatacantes para que crean que están explotando la verdadera red de la empresa. Esta red contendrá servicios alojados con un nivel específico de interacción y sin activos de empresa.

Para este proyecto se desplegará una honeynet virtual que implemente servicios similares de la red de la Universidad de Valladolid y que sea capaz de capturar, centralizar y visualizar el tráfico generado de los diferentes sistemas por los intrusos, de alertar en caso de una detección de ataque y de analizar ese tráfico para sacar una conclusión de qué técnicas o *Modus operandi* usó el atacante.

Índice

Página

Lista de abreviaturas

CEO → Chief Executive Officer
CERT → Computer Emergency Response Team
CSIRT → Computer Security Indicent Response Team
CVE → Common Vulnerabilities and Exposures
DDoS → Distributed Denial of Service
DNS → Domain Name System
ELK → Elasticsearch Logstash Kibana
FTP → File Transfer Protocol
GPG → GNU Privacy Guard
HTTPS → HyperText Transfer Protocol Secure
ICMP → Internet Control Message Protocol
IDS → Intrusion Detection System
IP → Internet Protocol
IPS → Intrusion Prevention System
JSON → JavaScript Object Notation
JVM → Java Virtual Machine
KVM → Kernel-based Virtual Machine
LAN → Local Area Network
MAC → Media Access Control
NAT → Network Address Translation
NIPS → Network Intrusion Prevention System
NMAP → Network Mapper
OSIF → Open Information Security Foundation
OSSIM → Open Source Security Information Management System
PHP → Hypertext Preprocessor
RAM → Random Access Memory
SIEM → Security Information and Event Management
SOC → Security Operation Center
SQL → Structured Query Language
SSH → Secure Shell
TCP → Transmission Control Protocol
TIC → Information and Communication Technology
TLS → Transport Layer Security
TTL → Time To Live
UDP → User Datagram Protocol
UTM → Unified Threat Management
VM → Virtual Machine
VPN → Virtual Private Network
WAN → Wide Area Network
XST → Cross-Site Tracing

1. Introducción y objetivos

En el presente documento, se acopia y expresa lo relativo al TFG 'HoneyNet', incluyendo aspectos técnicos y metodologías hasta la elaboración de los distintos manuales para la utilización del usuario.

1.1. Antecedentes

Debido a la evolución y la complejidad de los ciberataques estos últimos años, las empresas han invertido una gran cantidad de dinero en mejorar su estructura de red frente a estos ataques. De manera directa, comprando equipos más novedosos o nuevas tecnologías, o de manera indirecta, formando en aspectos de seguridad a los empleados. Este último año 2020, se ha batido récords de ciberataques debido en parte a la pandemia, y por ende, a adoptar otros aspectos como el teletrabajo y el cloud, que tienen una gran compatibilidad con la seguridad. En instituciones públicas y empresas privadas ha habido un incremento relevante de ataques respecto a otros años.

Según un estudio de las principales preocupaciones de los CEOs de las empresas, los ciberataques se encuentran en la segunda posición en los CEOs Españoles y cuarta posición a nivel mundial. Se estima que se produce un ciberataque cada 39 segundos, causando perdidas de hasta 550.000 millones de dólares en un año. Por esto, a muchas empresas se les recomienda invertir en ciberseguridad para paliar estas pérdidas económicas. Sin ir mas lejos, España invirtió 1.400 millones en ciberseguridad el pasado año 2020, ofreciendo un incremento del 6 % con respecto a 2019, y tendrá un crecimiento medio anual del 5,8 % en el período 2019-2022.[**key1**]

Figura 1: Preocupaciones CEOs nivel España y mundial [**key29**]

La mayoría de estos ataques son generados por fallos de configuración, ingeniería social, software desactualizado, etcétera, los cuales pueden ser prevenidas por el administrador de la red o sistema. Otros ataques menos comunes como los zero-day (también conocidos como 0day), son un tipo de vulnerabilidad las cuales acaban de ser descubiertas y que aún no tienen un parche que las solucione. Los zero-day entran dentro de los ataques desconocidos hasta el momento y son los más difíciles de prevenir por el administrador.

Ante estos ataques, se creó la idea de diseñar una red de baja/media/alta interacción con servicios vulnerables o no y sin activos de empresa para ser comprometida por los atacantes, y posteriormente capturar el tráfico y analizar las técnicas que han utilizado. De esta manera, podemos descubrir que nuevas técnicas o vulnerabilidades existan y poder informar o parchearlas antes de que sean usadas contra una red de una empresa o institución. El nombre que se le dan a este tipo de redes es HoneyNet.

1.2. Objetivos y alcance del proyecto

Como se ha mencionado anteriormente, nuestro objetivo principal es el desarrollo de una Honeynet virtual con similitudes que refleja la red de la UVa y comprender su funcionamiento y de las tecnologías que se usaran. Algunos de los distintos objetivos de este proyecto son los siguientes:

- **Llevar a la práctica lo aprendido a lo largo de la carrera:** Tras varios años cursando la carrera de Ingeniería Informática y especializándome en la mención de Tecnologías de la Información, este proyecto es una buena forma de plasmar los conocimientos de redes y sistemas operativos aprendidos estos años, además de ampliar los conocimientos en Ciberseguridad.
- **Aprender los conocimientos que se pueden requerir en un CERT:** Un objetivo derivado del anterior es aprender los métodos que se usan en los Equipos de Respuesta ante Emergencias Informáticas (CERT) donde se analizan eventos anómalos en redes y sistemas, proporcionar servicio de respuesta ante incidentes a víctimas de ciberataques, publicar informes relativos a amenazas y vulnerabilidades, y ofrecer información que ayude a mejorar la seguridad de estos sistemas.
- **Alcanzar una gran similitud a la red de la Universidad de Valladolid:** La HoneyNet tiene que tener un gran parentesco con la red la UVa. Para ello se implementarán algunos servicios públicos similares que se exponen como el jair.lab.inf.uva.es y aulas.inf.uva.es para confundir a los intrusos y que crean que están atacando a una red "verdadera".
- **Virtualización:** La Honeynet debe tener una topología de Generación III virtualizada.
- **Segura:** La Honeynet debe tener implementadas reglas de seguridad mediante IPtables y métodos de prevención de tal manera que no tenga fugas de seguridad ni se utilice para comprometer otros sistemas de Internet.
- **Escalable:** Es necesario que la HoneyNet sea escalable y se pueda implementar servicios ya existentes o nuevos servicios que puedan instalarse a futuro en la Universidad de Valladolid.
- **Funcional:** La HoneyNet tiene que alertar perfectamente cuando detecte una intrusión y poder reconocer el patrón del ataque si es conocido como puede ser un DDoS, bruteforce o un CVE conocido.

2. Planificación temporal

2.1. Gestión de riesgos

En esta sección, comentamos las posibles situaciones que pueden ocurrir y afectar tanto al desarrollo del proyecto como a la planificación. Algunos de estos riesgos son los siguientes:

- **Fecha de entrega del proyecto**: Se realiza una estimación con ayuda de un diagrama de Gantt de la duración de las tareas a realizar en el TFG. Cualquier imprevisto puede alterar al desarrollo del proyecto teniendo que retrasar otras tareas o la entrega del proyecto. Un plan de acción frente a este peligro es evaluar los problemas y las posibles soluciones.
- **Pérdida de información del proyecto**: La perdida de información puede suponer un gran retraso en el proyecto, por lo que se decidió hacer copias de las máquinas virtuales cada 15 días en un disco diferente, además de generar snapshot para tener un control de las versiones y volver entre ellas. En el caso de la memoria, se usará Overleaf que se encuentra en un servidor cloud, por lo que se guardará una copia en local cada 15 días también.
- **Desconocimiento de la tecnología utilizada**: Se parte de un conocimiento muy vago de estas tecnologías dado que no existe ninguna asignatura que las mencione en la carrera. Ciertas implementaciones pueden conllevar estudiar más a fondo una tecnología o cambiar a otra que encaje más en el proyecto.
- **Carencia de tiempo para realizar el proyecto**: Durante el desarrollo del proyecto, la persona encargada estará trabajando a jornada parcial por las mañanas, por lo que solo se dispondrá de tiempo para dedicárselo por las tardes. Como plan de acción, se intenta optimizar el tiempo disponible lo máximo posible.

2.2. Planificación temporal

Para el desarrollo del proyecto, se ha divido en distintas fases con el fin de planificar el tiempo que se dedica a cada una y de que no haya un retraso en la entrega final del proyecto. El siguiente diagrama de Gantt permite observar la distribución temporal de cada tarea con sus fechas de inicio y fin:

TFG	**147 días**	**mar 01/12/20**	**mar 22/06/21**	
Propuesta	15 días	mar 01/12/20	lun 21/12/20	
Elaboración de requisitos	20 días	mar 22/12/20	vie 15/01/21	2
Descripción actual de la red universitaria	11 días	sáb 09/01/21	sáb 23/01/21	
Estudio teorico	**28 días**	**sáb 23/01/21**	**mié 03/03/21**	**4**
Conocimiento sobre las Honeynet	17 días	sáb 23/01/21	dom 14/02/21	
Posibles tecnologías a aplicar en el desarrollo de la Honeynet	13 días	lun 15/02/21	mié 03/03/21	6
Desarrollo del sistema	**62 días**	**mié 03/03/21**	**jue 27/05/21**	**5**
Despliegue del laboratorio	12 días	mié 03/03/21	jue 18/03/21	
Despliegue de las tecnologías	**50 días**	**vie 19/03/21**	**jue 27/05/21**	
HoneyDocker	10 días	vie 19/03/21	jue 01/04/21	9
HoneyWall	40 días	vie 02/04/21	jue 27/05/21	11
Bateria de pruebas	9 días	vie 28/05/21	mié 09/06/21	12
Documentación	103 días	sáb 23/01/21	mar 15/06/21	
Últimos detalles, revisión y corrección de errores	5 días	mié 16/06/21	mar 22/06/21	14;13

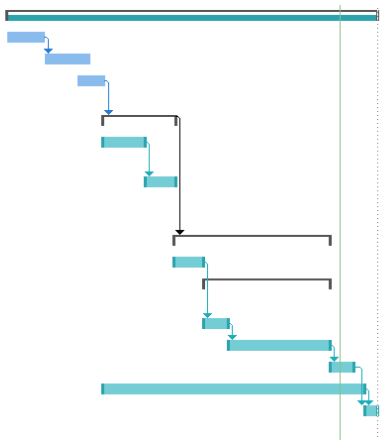

Figura 2: Diagrama de Gantt

- **Propuesta**: Durante la propuesta se comunica con el tutor el interés del alumno sobre el proyecto, la idea propuesta por el tutor y los trámites correspondientes con el proyecto.

- **Elaboración de requisitos**: Se realizó un estudio rápido sobre los objetivos del proyecto, hasta donde acotarlo y una idea poco detallada del proyecto.
- **Descripción actual de la red universitaria**: Se hace un estudio de la red universitaria de Valladolid para saber dónde implementar la posible futura Honeynet y que servicios replicar.
- **Estudio teórico**: Durante esta tarea se investigó la teoría relacionada con las Honeynet, los distintos tipos de Honeynet, qué sistemas usan, la tecnología que suele implementar, etcétera.
- **Despliegue del laboratorio**: Se realizó la creación de las máquinas virtuales, la conectividad entre ellas y su configuración básica. Durante esta tarea, se probaron distintos sistemas operativos y conexiones de red hasta encontrar la más óptima. Esto supuso un retraso de una semana.
- **Despliegue de las tecnologías**: Durante esta tarea, se instaló y configuró todo el sistema Honeynet. Esta fue la tarea más larga debido a los problemas del desconocimiento de las tecnologías y que muchas tecnologías no eran compatibles entre sí, teniendo que cambiar a otras opciones.
- **Batería de pruebas**: Con la Honeynet terminada, se puso a prueba mediante posibles ataques conocidos generados por el sistema Kali Linux y se detallaron los resultados de estos.
- **Documentación**: Para no perder los pasos que se iban siguiendo durante el proyecto, la memoria se ha ido escribiendo durante el desarrollo del proyecto. Esto supone la desventaja de tener que cambiar la documentación cada poco tiempo por cambios de decisión, aunque se le dio más importancia al poder redactar algo recién realizado y volver atrás para ver lo que se ha ido realizando que esa desventaja.

3. Descripción actual de la red universitaria de Valladolid

3.1. Infraestructura universitaria

En España, existe RedIRIS que es la red española que interconecta los recursos informáticos de las universidades y centros de investigación de todas las comunidades autónomas, además de proveer de servicios de conexión a Internet a estas instituciones. Despliega múltiples enlaces de fibra óptica de hasta 100.000 Mbps que son utilizados tanto para proyectos científicos como para facilitar la actividad docente. [**key2**]

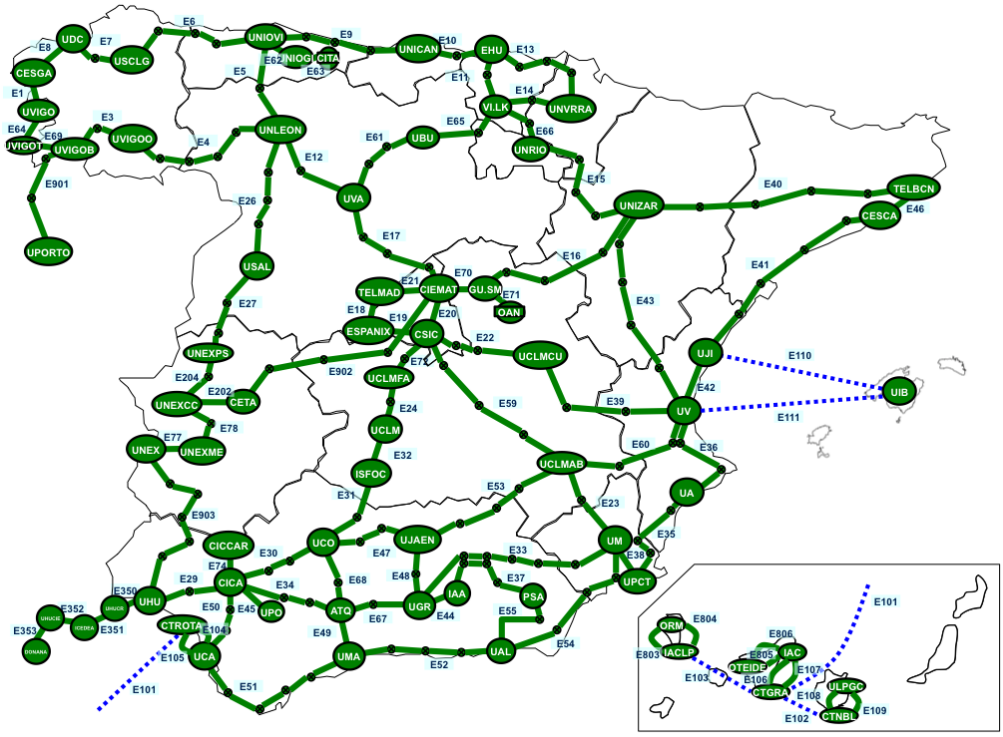

Figura 3: Mapa de interconexiones de RedIRIS España

Dentro de la RedIRIS, cada comunidad autónoma gestiona su parte de la RedIRIS. En Castilla y León, la infraestructura de red se llama Red de Ciencia y Tecnología de Castilla y León (CAYLE) y esta compuesta por 11 nodos: Ávila, Béjar, Burgos, León, Palencia, Ponferrada, Segovia, Soria, Valladolid y Zamora. De estos 11 nodos, hay 2 routers de alta prestación que se encuentran en Valladolid y León, y son los encargados de llevar el intercambio de tráfico IP entre los centros conectados a la Red Regional y el resto de redes académicas nacionales e internacionales. [**key3**]

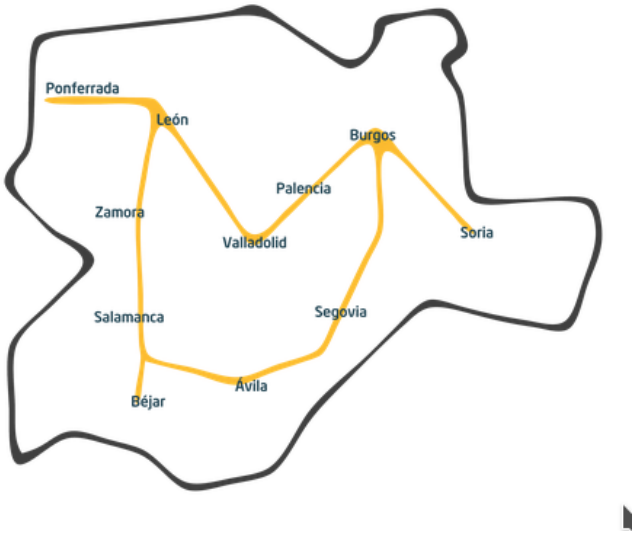

Figura 4: Mapa de interconexiones de Red CAYLE

Estos dos routers externos de la red se comunican con cada uno de los routers de la capa óptica (León, Salamanca, Burgos y Valladolid). En este proyecto, nos centraremos en analizar los servicios que ofrece al exterior el router de Valladolid de la capa óptica. Éste también da soporte a los campus de Palencia, Segovia y Soria.

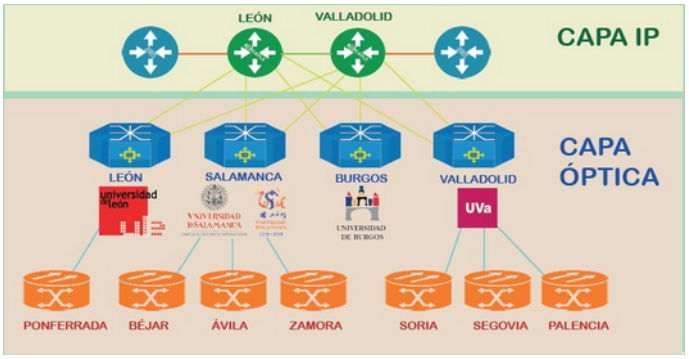

Figura 5: Infraestructura de Red CAYLE

3.2. Reconocimiento de servicios

Este paso es interesante cuando se quiere investigar que servicios replicar en la Honeynet. En este caso, ya decidimos imitar los servicios de jair.lab.inf.uva.es y aulas.inf.uva.es, pero aun así es un paso importante cuando desconoces los servicios de la red.
Para hacer un reconocimiento de los servicios, se han utilizado herramientas como amass y NMAP. Amass [23] es una herramienta de código libre para el mapeo de superficies de ataque y descubrimiento de activos externos mediante la recopilación de información. Con esta herramienta, podemos obtener todos los subdominios recursivamente que cuelguen del dominio principal que será **uva.es** e ir escaneando los que veamos convenientes en busca de los servicios públicos que estén disponibles.

Con el siguiente comando obtenemos todos los subdominios e IP de uva.es.

```
$ amass enum -ipv4 -d uva.es
```

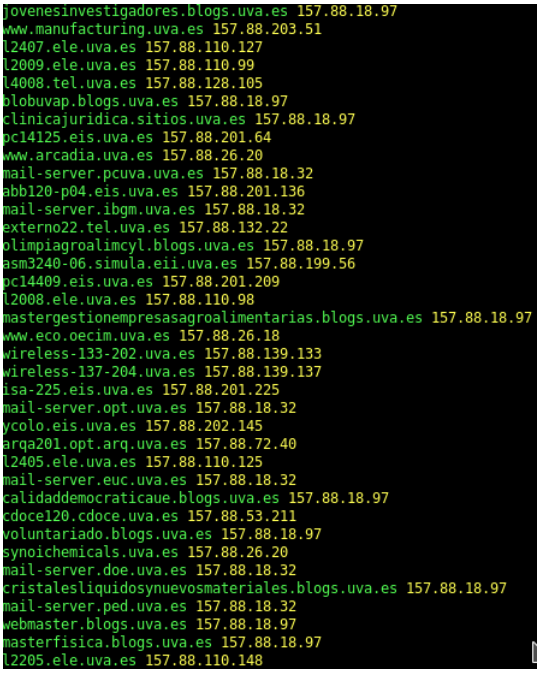

Figura 6: Algunos subdominios de uva.es

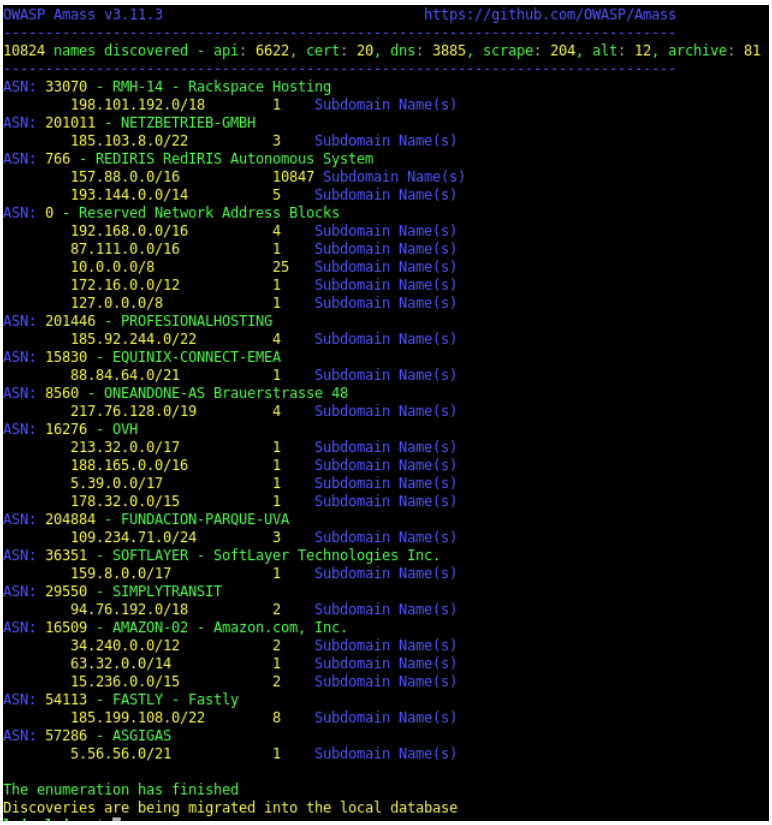

Figura 7: Resumen de Amass sobre uva.es

Como se puede ver, 10824 subdominios pertenecen a uva.es. Existen muchos que no están relacionados con la RedIRIS y la Universidad de Valladolid, otros están deshabilitados, no tienen ningún servicio o son

páginas creadas para un día en concreto. A partir de este punto, empieza la investigación por parte del administrador de la Honeynet de buscar los servicios que podrían ser más interesantes para los intrusos, nosotros en este TFG escogeremos dos servicios de la rama de Ingeniería Informática como es **aulas.inf.uva.es** y **jair.lab.inf.uva.es**.

Dominio	IP	Funcionamiento del Dominio
aulas.inf.uva.es	157.88.109.245	Moodle virtual para alumnos y profesores de la Escuela de Ingeniería Informática
jair.lab.inf.uva.es	157.88.125.192	Servidor para conectarnos a nuestro perfil de alumno de Ingeniería Informática

Cuadro 1: Tabla de dominios a implementar en la HoneyNet

El uso de Amass es interesante cuando se desconoce que subdominios tiene una IP o si se esta utilizando virtual hosting o alojamiento compartido. La infraestructura de la UVa utiliza la dirección de red pública 157.88.0.0/16.

3.2.1. Reconocimiento de puertos con NMAP

Una vez localizados estos dos dominios, pasaremos a escanear sus puertos con NMAP. **NMAP** [**key6**] es una herramienta de código abierto que sirve para efectuar escaneos de puertos o equipos en internet o una red interna. De esta manera, podemos saber si un host tiene puertos abiertos, si están filtrados por un firewall, detectar qué servicios se están ejecutando en esos puertos, e incluso obtener que sistema operativo y versiones utiliza el objetivo.

Las opciones que tenemos para realizar un escaneo son numerosas. Sin embargo, en este caso usaremos las opciones:

- **-p-** para que compruebe todos los puertos (0-65536)
- **-O** para identificar el sistema operativo y versión
- **-f** para fragmentar los paquetes de escaneo por si hay algún firewall o IDS entre medias
- **-Pn** para evitar que nos bloqueen los ping el firewall o un dispositivo de seguridad
- **-oN** para guardar la salida del comando en un fichero.

```
$ nmap -p- -O -f -Pn 157.88.109.245 -oN aulas.txt
```

```
Starting Nmap 7.70 ( https://nmap.org ) at 2021-02-15 19:44 CET
Nmap scan report for eyon.inf.uva.es (157.88.109.245)
Host is up (0.011s latency).
Not shown: 65530 filtered ports
PORT     STATE  SERVICE
53/tcp   closed domain
80/tcp   open   http
113/tcp  closed ident
443/tcp  open   https
8008/tcp open   http
Device type: general purpose|storage-misc|firewall
Running (JUST GUESSING): Linux 2.6.X|3.X|4.X (87%), Synology DiskStation Manager 5.X (87%)
OS CPE: cpe:/o:linux:linux_kernel:2.6.32 cpe:/o:linux:linux_kernel:3 cpe:/o:linux:linux_ke
Aggressive OS guesses: Linux 2.6.32 (87%), Linux 2.6.39 (87%), Linux 3.10 - 3.12 (87%), Li
No exact OS matches for host (test conditions non-ideal).

OS detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 674.99 seconds
```

Figura 8: NMAP sobre aulas.inf.uva.es

Con esto obtenemos todos los puertos pero sin mucha información de qué hay detrás de estos puertos. Entonces ejecutamos un segundo comando especificando los puertos abiertos que nos han aparecido con **-p[port],[port]** y con **-sV** para obtener más información sobre estos. La información que podemos obtener de cada puerto depende de si esta filtrado por el firewall, si aparece 'closed' y otros factores.

```
$ nmap -p80,443,8008 -sV -f -Pn 157.88.109.245 -oN aulas_port_sv.txt
```

```
Starting Nmap 7.70 ( https://nmap.org ) at 2021-02-15 20:08 CET
Nmap scan report for eyon.inf.uva.es (157.88.109.245)
Host is up (0.0096s latency).

PORT     STATE  SERVICE  VERSION
53/tcp   closed domain
80/tcp   open   http     Apache httpd (PHP 7.3.25)
113/tcp  closed ident
443/tcp  open   ssl/http Apache httpd (PHP 7.3.25)
8008/tcp open   http
```

Figura 9: NMAP sobre aulas.inf.uva.es con la opción -sV

En este caso, solo hemos obtenido mas información del puerto 80 y 443 que es un servicio de Apache httpd con versión PHP 7.3.25. Realizamos la misma operación sobre el dominio jair.lab.inf.uva.es para tener una idea de que servicios y versiones implementaremos en nuestra honeynet:

```
Starting Nmap 7.70 ( https://nmap.org ) at 2021-02-15 22:02 CET
Nmap scan report for jair.lab.inf.uva.es (157.88.125.192)
Host is up (0.012s latency).

PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 8.4 (protocol 2.0)
8008/tcp open  http
| fingerprint-strings:
|   FourOhFourRequest:
|     HTTP/1.1 302 Found
|     Location: https://:8010/nice%20ports%2C/Tri%6Eity.txt%2ebak
|     Connection: close
|     X-Frame-Options: SAMEORIGIN
|     X-XSS-Protection: 1; mode=block
|     X-Content-Type-Options: nosniff
|     Content-Security-Policy: frame-ancestors
|   GenericLines, HTTPOptions, RTSPRequest, SIPOptions:
|     HTTP/1.1 302 Found
|     Location: https://:8010
|     Connection: close
|     X-Frame-Options: SAMEORIGIN
|     X-XSS-Protection: 1; mode=block
|     X-Content-Type-Options: nosniff
|     Content-Security-Policy: frame-ancestors
|   GetRequest:
|     HTTP/1.1 302 Found
|     Location: https://:8010/
|     Connection: close
|     X-Frame-Options: SAMEORIGIN
|     X-XSS-Protection: 1; mode=block
|     X-Content-Type-Options: nosniff
|_    Content-Security-Policy: frame-ancestors
|_http-title: Did not follow redirect to https://jair.lab.inf.uva.es:8010/
```

Figura 10: NMAP sobre jair.lab.inf.uva.es

- **aulas.inf.uva.es:**

  - **Port 80 y 443:** Aloja un servicio apache httpd y versión PHP 7.3.25 para la web de aulas de Ingeniería Informática.
  - **Port 8008:** Sobre este puerto no se ha obtenido ningún tipo de información.

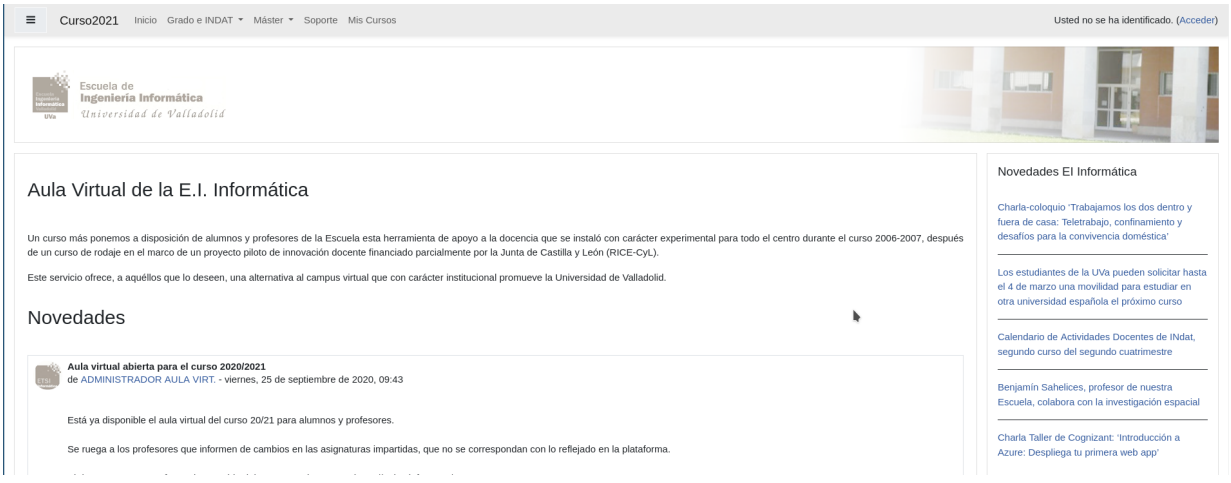

Figura 11: Portal web/moodle alojado en aulas.inf.uva.es

- **jair.lab.inf.uva.es:**
  - **Port 22:** Protocolo SSH versión OpenSSH 8.4
  - **Port 8008:** Sobre este puerto no se ha obtenido ningún tipo de información.

4. Teoría Honeynet

Dentro de las organizaciones, la mayoría tienen implementadas soluciones o dispositivos de seguridad conocidos como pueden ser firewalls, IDS, IPS, sniffers, antivirus, soluciones contra DDoS, VPNs, proxy pero en muy pocas empresas se despliega una solución como son los Honeypots o Honeynet. Esta solución es desconocida incluso por gente del sector TIC que no sabe cuál es su finalidad, qué tipos de honeynet existen, dónde se puede desplegar esta arquitectura, con qué programas trabaja conjuntamente y otra serie de preguntas. En este apartado, resolveremos este tipo de preguntas teóricas.

4.1. ¿Qué es un honeypot?¿y una honeynet?

Un honeypot o sistema trampa es un dispositivo de seguridad basado en un sistema con ficheros, directorios y servicios como un sistema real implementado en la red cuyo objetivo es ser probado, atacado y comprometido por un posible ataque informático, para así detectarlo y permitir el análisis del ataque en un entorno controlado. [**key32**]

Un honeypot, como se puede observar, no es una herramienta preventiva ya que su función es la de recoger información de ataques informáticos y detectar la firma de estos, por lo que es más una herramienta de detección. Además de ser una herramienta de detección, entra dentro de la categoría de herramientas de respuesta ya que, una vez recogida la información , se analiza el proceso de cómo se ejecutó el ataque, si se ha utilizado una herramienta de escaneo de puertos , de detección de vulnerabilidades, fuerza bruta, o métodos desconocidos, dando lugar a los encargados de la seguridad de la red a desplegar una respuesta y contramedida contra este tipo de amenazas.

Como conclusión, un honeypot es una herramienta que recoge información y evidencias sobre las potenciales amenazas y vulnerabilidades de redes y sistemas, y obtiene el conocimiento para ejecutar una contramedida ante estos ataques.

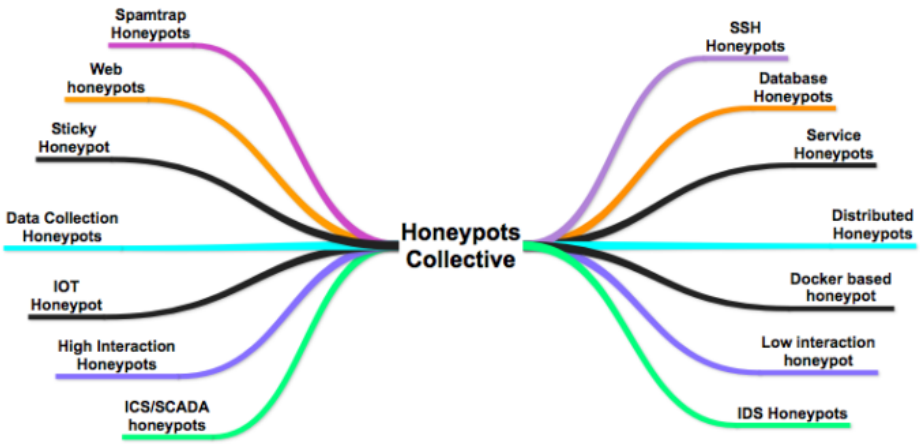

Figura 12: Tipos de honeypots según el tipo de información que se quiera recopilar

La honeynet, como se puede deducir, es una arquitectura de red formada por honeypots y su función es la misma: ser atacada para posteriormente recopilar información. Los Honeypots pueden jugar diferentes roles en la red de una organización: mejora de IDS y Firewalls, investigación, respuesta a incidentes, análisis forense, engaño a los atacantes y disuasión.

Este tipo de herramientas deben ser desplegadas y mantenidas por manos expertas, ya que una mala configuración del sistema puede llevar a generar una entrada al sistema de producción o red de la empresa, o a que el honeypot sea usado para comprometer otras empresas externas usándolo como un proxy. Por lo tanto, los honeypots/honeynet deben ser atendidos y actualizados constantemente y suelen ir respaldados por otros dispositivos de seguridad como firewalls, IDS/IPS, antivirus, UTM, etc, así como políticas de seguridad y procesos auditados.

4.2. Clasificación de Honeypots

Dentro de los honeypots, se pueden diferenciar en varias categorías: según su utilidad **(producción o investigación)** o según la interacción del atacante **(baja, media o alta)**. [**key33**]

4.2.1. Según su utilidad

- **Honeypots de producción:** Se usan para proteger la infraestructura interna de la red. Es necesario que estén correctamente diseñados e implementados ya que puede llegar a ser un punto vulnerable de la red, aunque su naturaleza sea ser probados y atacados, se debe tener precaución.
  La finalidad de este tipo de honeypots es desviar o mitigar el riesgo de cualquier red contra los ciberataques. El despliegue e implementación de los honeypots de producción son más sencillos ya que requieren menos funcionalidad y servicios ejecutándose.

- **Honeypots de investigación:** Se usa para acumular evidencias e información con el fin de analizar el comportamiento, patrones, firmas y motivos por los que el atacante ha perpetrado este ataque contra la red o sistema.
  La información que podemos obtener a través de esta herramienta son qué herramientas ha utilizado para el ataque, el sistema operativo, tipos de exploits utilizados, si se trata de un hacker individual, un grupo terrorista o de un país, la metodología usada, etc.
  Con este análisis podemos sacar conclusiones que ayuden a otras empresas a alertar y mitigar estos ataques, advertir a entidades que estén en el punto de mira según el motivo de los atacantes (información de personas, empresas, gobiernos, etc..)

Como conclusión, si una organización quiere proteger sus activos de ataques informáticos, con un honeypot de producción se pueden bloquear los ataques potenciales y perseguir a los atacantes. En cambio, si se quiere conocer las técnicas, metodologías, comportamientos, herramientas, exploits, grupo de atacantes, etc. para aprender y poderlo aplicar a sistemas de seguridad en modo de actualización o parchear fallos, el honeypot de investigación es el más adecuado.

4.2.2. Según su interacción con los atacantes

Otro tipo de honeypot que podemos distinguir son por el nivel de interacción con los atacantes y la capa de la pila de componentes de un servicio que sea emulado en el honeypot.

- **Honeypots de baja interacción:** Los aspectos para la implementación son sencillos, ya que son procesos de instalación, configuración, despliegue y mantenimiento, que no llevan ninguna complejidad. En estos sistemas se emulan los sistemas operativos, protocolos y servicios básicos sobre un host con Microsoft Windows o Linux, y puede ser en una máquina física o entorno virtualizado como VMware, Promox, KVM. Estos sistemas deben ser bastionados para prevenir que el atacante pueda obtener acceso al sistema que soporta la virtualización o sistema host.
  El nombre de baja interacción viene dado porque el atacante solo tendría capacidad para conocer el sistema operativo que se está emulando, aprovechar alguna vulnerabilidad básica o escalada de privilegios sobre algún servicio básico del honeypot, pero nada más, ya que no se está ejecutando ningún servicio real.
  Algunos de los resultados que podemos obtener, son si el atacante ha utilizado fuerza bruta para acceder al servicio FTP, SSH, correo, con qué datos ha probado los intentos, la fecha y hora a la que se realizó el ataque, protocolo, herramienta, dirección IP, etc. El nivel de riesgo de estos honeypots es bajo, ya que no pueden ser utilizados como puente para atacar otras partes de las redes, puesto que simulan el servicio y no es real.
  Algunos ejemplos de honeypots de baja interacción son Specter, Honeyd, SF Sensor.

- **Honeypots de media interacción:** El despliegue del sistema operativo y servicios es más real, proveyendo de más información al atacante. Un ejemplo es que pasamos a poder emular servicios de base de datos o servidores de aplicaciones.
  El honeypot debe estar configurado para responder parcialmente a los ataques ya que, los protocolos, puertos y funciones deben estar emulados de forma correcta. Como este despliegue es parcial y los servicios y aplicaciones no son completamente reales, el ataque no se puede ejecutar del todo, por lo que obtenemos más información que el honeypot de baja interacción pero sin llegar a que el intruso pueda terminar su ataque.
  Este tipo de honeypots requiere más conocimientos para implementarlos, por el simple hecho que debe saber cómo funcionan los protocolos y qué hacen los servicios y aplicaciones a emular. Además es importante asegurar que el sistema operativo real sea robusto contra la intrusión para que en caso de ser comprometido no afecte a la red.

- **Honeypots de alta interacción:** En este sistema no se emula nada, todos los servicios y aplicaciones son reales. Por lo tanto, hay un aumento del riesgo pero a su vez se amplía el alcance de la información que se puede obtener de un atacante.
  Al ser los sistemas reales, la instalación, despliegue y configuración pasan a ser muy complejas y necesariamente realizadas por alguien experto en la materia y con un gran conocimiento en las aplicaciones que se van a instalar.
  El fin de este tipo de honeypot es que el atacante se haga con el control del honeypot, de manera que obtenga privilegios de super usuario. La información que podemos obtener de estos atacantes, a parte de la mencionada en los de interacción baja y media, es el poder monitorizar la interacción con todos los componentes del servicio, como el sistema operativo, middleware, aplicaciones, etc..
  Al ser un riesgo muy elevado, este honeypot debe estar monitorizado 24 horas por un experto y aislados de la red de producción de la empresa para reducir la capacidad del atacante mediante un firewall o IDS que bloquee el tráfico saliente a otra red.

4.3. Arquitectura de honeypots

A la hora de crear una arquitectura de honeynet, existe una absoluta libertad de qué topología de red diseñar y de qué herramientas usar para capturar la información, monitorizar, controlar y analizar el tráfico del atacante.
Sin embargo existen tres tipos de arquitecturas básicas llamadas GEN I, GEN II o GEN III (virtual).

4.3.1. GEN I

Esta arquitectura es la más clásica de todas, conformada por un firewall fronterizo al exterior para filtrar el tráfico y redirigirlo o a la honeynet o a la red de producción, un router detrás del firewall que separa la red de producción y la honeynet y ayuda a reforzar la seguridad junto al firewall, y un IDS conectado a la red de producción y a la red de honeynet mediante **port mirroring**.

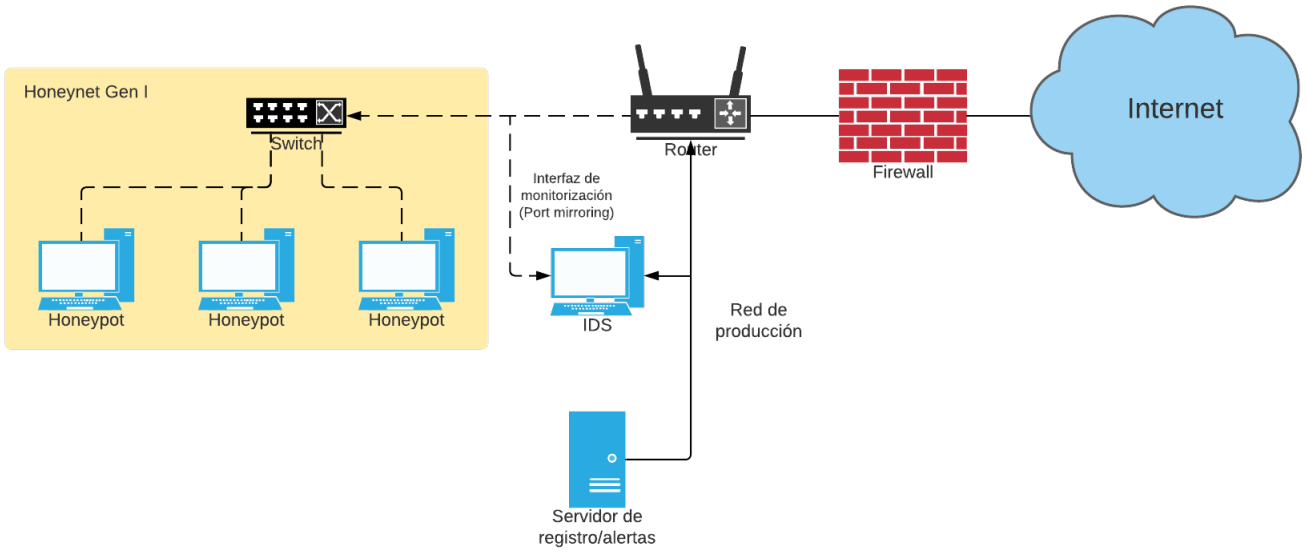

Figura 13: Infraestructura de HoneyNet de generación I

El firewall controla las conexiones entrantes y salientes, pero además de eso, tiene que controlar el límite de conexiones que se realiza de un honeypot a Internet, cerrando conexiones cada x número de conexiones por hora. Esto se puede hacer automáticamente o manualmente ya que, si el técnico encargado de vigilar quiere aprender más sobre el ataque que se está realizando, puede levantar la mano un poco al atacante.

Esto se hace para que el atacante no realice ataques DDoS o fuerza bruta usando los honeypots como equipo intermediario.

El router está después del firewall para:

1. Si el atacante está dentro del honeypot que no sea capaz de ver el firewall y piense que detrás del router está el acceso a Internet. Esto se hace configurando las dos interfaces del firewall perimetral como bridge y ninguna dirección IP, lo que le hace transparente al traceroute o TTL del paquete.

2. La segunda para complementar la seguridad del firewall, actúa como un control de acceso en la capa dos. Esto nos protege contra ataques de spoofing, DoS ataques basados en ICMP y bloquea el tráfico ICMP saliente, y así limitamos los ataques que se puedan lanzar a otros sistemas.

En cuanto a la captura de información, disponemos de varios dispositivos que capturan el tráfico:

1. **Firewall:** Primera capa de control de datos de conexiones entrantes y salientes hacia y desde la Honeynet, alertándonos en caso de nuevas conexiones. El firewall también nos avisa de backdoors en el sistema, conexiones por puertos aleatorios o altos y nos avisa cuándo una honeynet inicia una conexión saliente (reverse shell).

2. **IDS:** Tiene 2 propósitos en la arquitectura:

   a) Capturar toda la actividad. El IDS reside en un **puerto de monitorización** o **port mirroring**. El puerto espejo o port mirroring es utilizado en un switch de red o hub para enviar copias de los paquetes de red, que atraviesen los puertos del switch específicamente configurados, a una conexión de red monitoreada en otro puerto. De esta manera, estamos monitorizando el tráfico de la honeynet sin estar conectados directamente a ella.

   b) Cuando un paquete concuerda con una firma o patrón de ataque, se genera una alerta para avisar al administrador.

3. **Los propios sistemas:** Deben capturar los logs que se generan en el sistema. Además de tenerlo en local debemos tenerlo en un servidor remoto. No trataremos de ocultar el syslog, ya que si el atacante descubre e intenta desactivar el syslog, obtendremos como lo ha realizado. Estas son técnicas avanzadas para conseguir el control y limpiar los registros.

Como conclusión, esta arquitectura GEN I funciona correctamente contra ataques automatizados y atacantes de nivel básico. El entorno no es útil para ataques avanzados por su poco atractivo consistiendo en instalaciones por defecto del sistema operativo.

4.3.2. GEN II

La honeynet Gen II mejora la flexibilidad, gestión y seguridad respecto a la Gen I.

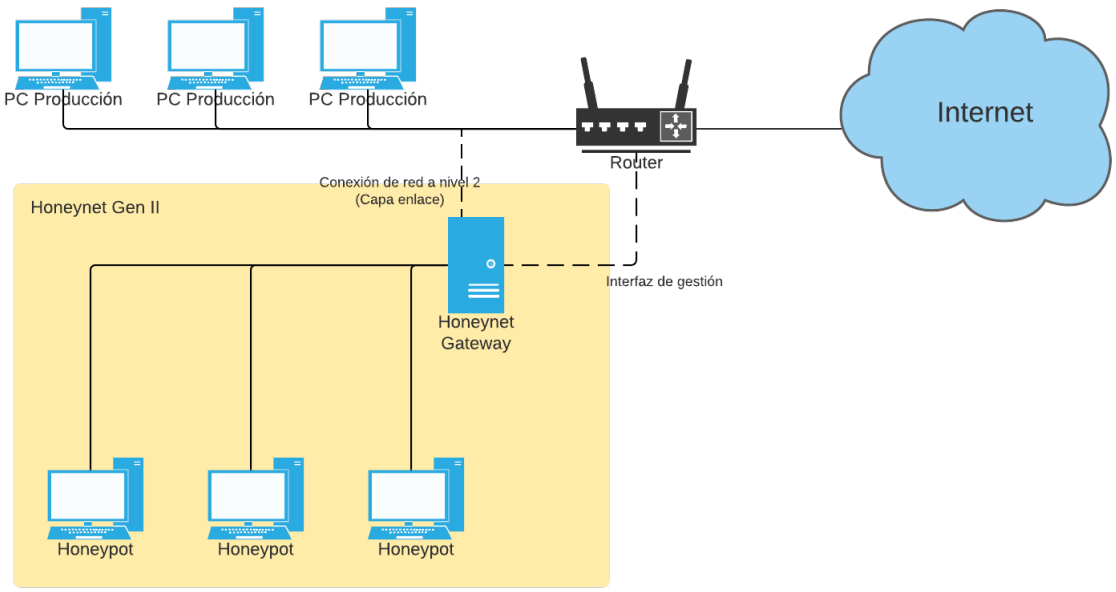

Figura 14: Infraestructura de HoneyNet de generación II

Al tratarse de una puerta de enlace de nivel dos (bridge), el Honey Gateway se encuentra dentro de la red interna de producción y la Honeynet al contrario que la Gen I. El Honey Gateway como está en modo Bridge no tiene IPs asignadas por lo que es más difícil detectarlo.
La otra interfaz eth2 es para gestionar y realizar tareas remotas del Honey Gateway vía VPN. Es una red aislada del resto para la administración.
Algunas de las ventajas que tiene son: La dificultad de detectar el Honey Gateway ya que no hay saltos de enrutamiento, decremento de TTL, ni direcciones MAC asociadas al gateway. Además combinamos el control del tráfico y la captura de datos en un solo equipo, el Honey Gateway.

Para configurar el Honey Gateway necesitamos una distribución que soporte el modo bridging e IP-tables. Estas características son críticas para el buen funcionamiento aunque no suelen ser muy compatibles. Para facilitar su implementación, existe un script **rc.firewall** que implementa toda la configuración del Honey Gateway incluyendo el bridging, firewall y la configuración de la interfaz de gestión.

Dentro del Honey Gateway necesitamos realizar un conteo que permita limitar las conexiones salientes de un honeypot e integrar un sistema de prevención de intrusiones de red (NIPS) para bloquear los ataques conocidos. En el script del rc.firewall es donde especificamos cuántas veces puede un atacante iniciar una conexión saliente (TCP, UDP, ICMP, . . .). Es importante saber que delimitar las conexiones hace suponer al atacante que se encuentra en una honeynet, ya que puede comprobar por ejemplo cuántas conexiones salientes puede hacer en un tiempo antes de que se bloquee. Esto puede arreglarse con un conteo dinámico sin un límite de conexiones fijo. Un atacante necesita realizar una conexión saliente para descargar toolkits, generar una reverse shell, configurar bots automáticos, enviar correos, etc.
NIPS investiga todos los paquetes que viajan a través de la gateway y si alguno concuerda con alguna de las reglas IDS o firmas, genera una alerta y bloquea o modifica el paquete. Esto solo funciona en ataques conocidos. La idea del NIPS es la de bloquear o deshabilitar cualquier ataque identificando esas 15 primeras conexiones.

En cuanto a la captura de información, tenemos varios sistemas:

- **Registros del firewall:** Primera indicación de lo que está haciendo un atacante.
- **Tráfico de red:** Consiste en capturar cada paquete y su contenido. Configuramos y ejecutamos un proceso que capture y vuelque todo a un fichero tcpdump.
- **Actividad del sistema:** Capturar lo que ocurre en el propio honeypot como comandos ejecutados, procesos abiertos, exploits o herramientas utilizadas dentro del sistema.

4.3.3. GEN III (Virtual Híbrida)

La tercera generación posee la misma arquitectura que la Gen II, pero experimenta ciertas mejoras de capacidad de gestión y análisis avanzado de datos. Para minimizar la inversión de recursos económicos y físicos, ofrecer seguridad, escalabilidad, flexibilidad y una implementación sencilla, dicha arquitectura se efectúa por medio de una Honeynet Virtual Híbrida.

Una Honeynet Virtual Híbrida está formada por dos equipos; uno cumple las funciones de HoneyWall como firewall, permitiendo las conexiones entrantes y salientes, y otro equipo que contiene las máquinas virtuales que constituyen los Honeypots. Este tipo de arquitectura ofrece ventajas semejantes a las proporcionadas por una Honeynet con dispositivos físicos. La Honeynet Virtual Híbrida de Tercera Generación se ubica en la red interna separada de la red de producción por el HoneyWall.

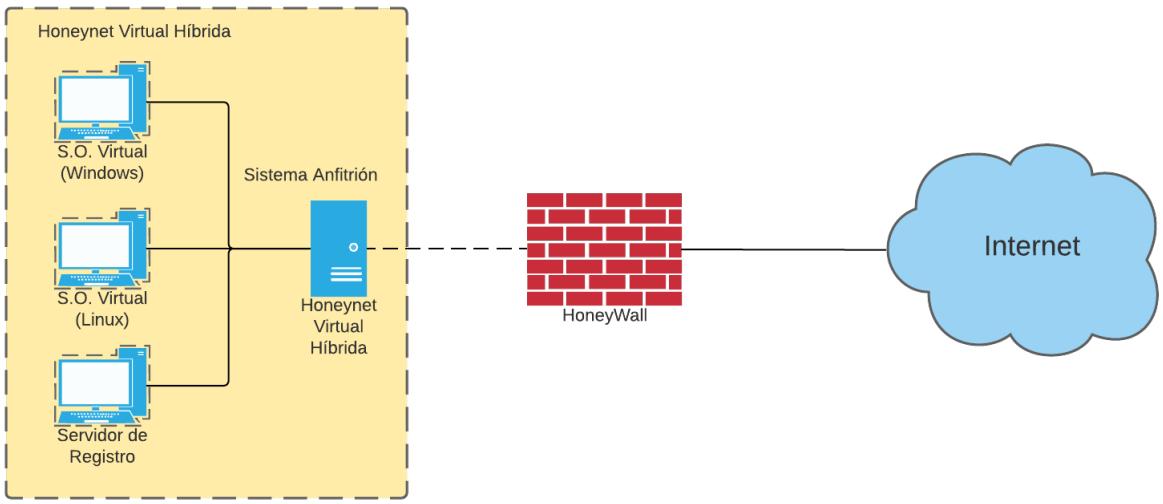

Figura 15: Infraestructura de HoneyNet Virtual Híbrida de Tercera Generación

Algunas ventajas de esta arquitectura frente a las otras son:

- Los atacantes no dañan sistemas reales. En caso de que se corrompa una máquina, es tan sencillo como volver a cargar una instancia de ella.
- Utilizar sistemas trampas similares a los de producción permite identificar fallos de seguridad existentes en el entorno real.
- Mayor flexibilidad a la hora de utilizar software para el control y captura de los datos de red.

4.4. ¿Dónde se ubica la Honeynet en la red de la empresa?

Ahora que ya tenemos un conocimiento mínimo de que es una Honeynet, es importante saber donde puede estar ubicada esta red. Lo normal, es situarla dentro de la propia intranet de la empresa, detrás del firewall exterior perimetral y separada del resto de subredes de producción para proteger de posibles fugas en un ataque.

Una empresa para poder exponer sus servicios al público alquila una IP pública a un ISP que será la asociada a un router externo con grandes prestaciones. Sobre esta IP colgaran varios dominios que serán los que utilizaran los usuarios para acceder a los servicios de la empresa. Y la pregunta es, ¿Como sabe el router o firewall externo a donde redirigir el tráfico de cada usuario, si al sistema real o a la Honeynet?

Mediante la herramienta **'Bait and Switch'** que se instala en un sistema con tres interfaces de red (la interfaz externa, la interfaz hacia la red de producción y la interfaz hacia la Honeynet) y redirecciona el tráfico entrante que detecta como hostil hacia el sistema trampa. El atacante no se percatará de que está atacando un sistema falso, porque accedió a través de la IP pública original y dado su comportamiento malicioso fue redirigido por dentro de la intranet. Claramente, el atacante no se percatará si los servicios de la Honeynet son idénticos a los reales.

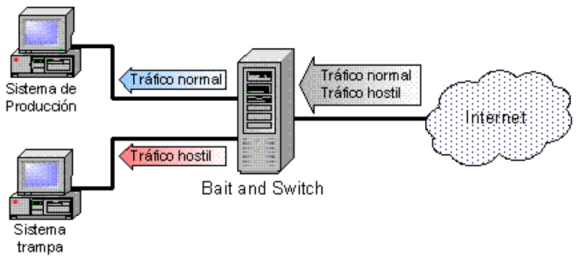

Figura 16: Funcionamiento de Baite and Switch

4.5. Tecnologías usadas durante el proyecto

4.5.1. IPtables

IPtables [**key30**] es una aplicación de nivel de usuario que permite la gestión, configuración y manejo del filtrado de tráfico de red en un sistema tanto entrante como saliente. Posee la capacidad de pasar los paquetes IPv4 a través de tablas, que estas a su vez clasifican y organizan las reglas de acuerdo al tipo de decisiones que se deben tomar según los paquetes que atraviesen.

Las reglas en IPtables están agrupadas en cadenas. Una cadena es un conjunto de reglas utilizadas para determinar qué hacer con los paquetes que entran, salen o traspasan el firewall. Estas cadenas se agrupan en tablas que definen el tipo de regla que mantienen como filtrado, NAT, manipulación de paquetes, etc..
A continuación, se enumeran los pasos del funcionamiento de IPtables [**key7**]:

1. Las reglas se agrupan en cadenas y estas cadenas están contenidas por tablas.

2. Los paquetes pasan por todas las tablas y cadenas configuradas.

3. Cuando un paquete coincide con la condición de una regla puede ser ACCEPT o DROP dependiendo de lo definido en la regla.

4. Cuando un paquete no coincide con ninguna regla, se le aplica la regla por defecto de la cadena por la que este pasando.

Las decisiones de enrutamiento entran en juego a la hora de aplicar las reglas, ya que se debe elegir si aplicar a la salida o entrada de donde este conectado la máquina. Se podrá aplicar antes o después de enrutar a otra red.

Existen cuatro tablas predeterminadas, pero se pueden añadir más tablas con algún módulo:

- **Filter:** Encargada del filtrado de paquetes, deja o no pasar los paquetes dependiendo de las reglas contenidas en sus cadenas. Existen las siguiente cadenas dentro de esta tabla:
  - INPUT: es la cadena que define las reglas para los paquetes que reciben un proceso local de la máquina. Los paquetes que tengan como destino final la máquina local de las IPtables.
  - OUTPUT: Igual que INPUT, pero por aquí pasarán los paquetes generados por la máquina local de las IPtables.
  - FORWARD: Esta cadena es por la que pasan los paquetes que no van dirigidos a la máquina local. Se usa cuando la máquina configurada con IPtables posee la capacidad de enrutar paquetes.
- **NAT:** Proceso de NAT:
  - PREROUTING: cadena donde se realiza DNAT (Destination NAT). Se hace NAT para port forward de IP pública a LAN.
  - POSTROUTING: cadena donde pasan los paquetes después de la decisión de enrutado. Se usa para hacer SNAT (Source NAT). Enmascara los paquetes con la IP de la interfaz de la salida. Por ejemplo, la transformación de IPs locales en IP pública.
  - OUTPUT: cadena usada para realizar operaciones de nateo para paquetes generados por un proceso local de la máquina que contiene las IPtables.
- **Mangle:** Es una tabla en la que se pueden manipular determinados aspectos del paquete. Pueden ser usadas para tomar decisiones de enturado dependiendo del tipo de tráfico (PREROUTING, INPUT, FORWARD, OUTPUT y POSTROUTING).
- **RAW:** Suele establecer una marca (NOTRACK) para evitar que el netfilter realice un seguimiento de paquete (PREROUTING y OUTPUT).

4.5.2. Docker

Docker [**key8**] es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos. Estos contenedores ligeros y portables se podrán ejecutar en cualquier máquina que tenga Docker instalado, facilitando el despliegue, portabilidad e independencia del sistema operativo que tenga por debajo.

Una gran diferencia que tiene Docker frente a las VM es que para cada instancia de VM es necesario virtualizar el entorno del sistema operativo, reduciendo así su rendimiento. Sin embargo, los contenedores Docker se ejecutan en el propio sistema operativo y solo virtualizan la aplicación que se despliega y no un sistema operativo completo. De esta manera, el consumo de recursos de hardware mediante Docker es muchísimo menor que usando máquinas virtuales.

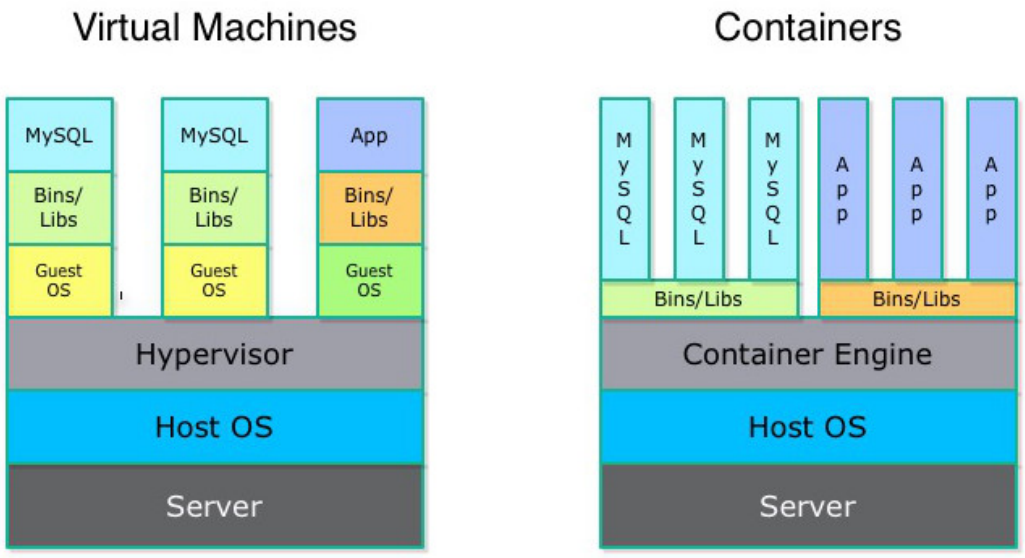

Figura 17: Virtual Machine vs Docker

Un contenedor Docker se crea a partir de una imagen de Docker, que es un "template" de un entorno que cuenta con todos los elementos necesarios para la app (libs, config files, software, code, ...). A partir de esta imagen se pueden crear diferentes versiones del contenedor como si se tratara de Git.

Docker puede crear imágenes automáticamente leyendo las instrucciones de un archivo Dockerfile. Dockerfile es un documento de texto que contiene todos los comandos que un usuario puede llamar en la línea de comandos para ensamblar una imagen. Para utilizar Dockerfile usamos el siguiente comando:

```
$ docker build -f [/path/dockerfile]
```

Las ventajas de los contenedores Docker son las siguientes: [**key9**]

- **Modularidad:** Docker permite dividir las funcionalidades de la aplicación en contenedores individuales. Un ejemplo de esto es tener la base de datos en un contenedor y una aplicación Node.js en otro.
- **Capas y control de versiones de la imagen:** Cada archivo de imagen de Docker está conformado por una serie de capas. Estas capas se combinan en una única imagen y una capa se crea cuando la imagen cambia. Docker reutiliza estas capas para construir nuevos contenedores mejorando la rapidez en el proceso de construcción. Cada vez que hay un cambio nuevo, existe un registro de cambios integrado con control total de las imágenes del contenedor.
- **Restauración:** Docker tiene una gran capacidad de restaurar a versiones anteriores gracias a la creación de capas de una manera ágil y sencilla.

Las desventajas de los contenedores Docker: [**key10**]

- **Seguridad:** El daemon de Docker puede ser un problema si está mal configurado. Docker utiliza este daemon que requiere permisos de acceso al directorio raíz, por lo que se debe prestar especial atención a quiénes obtienen acceso al proceso y en dónde reside éste. [**key11**]
- **Almacenamiento persistente:** El sistema Docker está diseñado para que cuando se apague el contenedor se borre todo lo que hay en su interior.

- **Incompatibilidad con ciertas aplicaciones:** Las aplicaciones con ventanas o gráficas no son compatibles con los contenedores o incluso algunas aplicaciones por la manera en que Docker encapsula una aplicación con sus librerías esenciales y configuraciones. Si se requiere una librería adicional se complica la configuración del contenedor Docker.

4.5.3. Pila ELK

La pila ELK [**key12**] es un conjunto de herramientas de gran potencial de código abierto que se combinan para crear una herramienta de administración de registros permitiendo la monitorización, consolidación y análisis de logs generados en múltiples servidores. ELK está compuesto por las herramientas: Elasticsearch, Logstash y Kibana.

La pila ELK se implementa en muchos centros de respuesta ante incidentes de seguridad como CERT, CSIRT y SOC, ya que ofrece la detección de incidencias en tiempo real, almacenamiento de gran cantidad de información, escalabilidad de un sistema, centralización y búsqueda compleja de la información de los logs. Sin este conjunto de herramientas sería bastante difícil o casi imposible monitorizar o esclarecer los incidentes que ocurran en una red donde puede haber más de 100 dispositivos generando logs. [**key13**]

A continuación, se hace una explicación de cada componente de la pila ELK:

- **Logstash** es la herramienta que se utiliza para recolectar, analizar (parsear) y guardar los logs para futuras búsquedas. Logstash soporta :

  - Entradas: Son las fuentes de datos de dispositivos que generar los logs en la red como puede ser un firewall, endpoint o servidor.
  - Codecs: Convierten un formato de entrada en un formato aceptado por Logstash.
  - Filtros: Se utilizan para procesar (parsear) los eventos y que se vean con un formato más adecuado y estructurado.
  - Salidas: Son los destinos donde los datos ya procesados serán enviados, normalmente será al Elasticsearch.

  Los problemas que soluciona Logstash son la descentralización de los logs, ya que por cada dispositivo se genera uno o varios logs diferentes por cada aplicación que esté ejecutando. Con Logstash se pueden recibir todos estos logs y parsearlos para que tengan un formato más legible y más fácil de analizar. Cada log puede venir con un formato de tiempo distinto.

- **Elasticsearch** es un motor de búsqueda que se basa en Lucene el cual nos permite realizar búsquedas por una gran cantidad de datos de un texto específico. También se puede definir como una base de datos NoSQL orientada a documentos JSON y que nos permite indexar grandes volúmenes de datos para poder consultarlos posteriormente. Elasticsearch permite acceder a los datos en tiempo real.
- **Kibana** es un software de panel de visualización de datos para Elasticsearch. Proporciona capacidades de visualización además del contenido indexado en un clúster de Elasticsearch. Los usuarios pueden crear gráficos de barras, de líneas y de dispersión, o mapas sobre grandes volúmenes de datos.

  Kibana se relaciona con Elasticsearch para buscar, ver y visualizar datos indexados en Elasticsearch y analizarlos a través de la creación de gráficas de barras, tablas, histogramas y mapas. Desde Kibana se

pueden crear Dashboard que son recopilaciones de gráficos, grafos, métricas, búsquedas y mapas que se recopilaron en un solo panel, pudiendo así echar un vistazo sobre datos desde varias perspectivas y permitiendo a los usuarios explorar los detalles de estos [**key14**].

Beat es la herramienta que se utiliza en los servidores o dispositivos de la red para enviar los logs que generan al servidor ELK. Normalmente van directos al Elasticsearch o si necesitan otro formato pasan primero por el Logstash para parsearlos. Dicho de otra manera, son los agentes encargados de envíar datos o logs de cientos o miles de máquinas y sistemas a Logstash o Elasticsearch.

Existen diferentes tipos de Beat, los más utilizados son:

- **Filebeat**: permite la recolección, parseo y envío de datos de ficheros logs.
- **Metricbeat**: permite la recolección y envío de métricas a nivel de sistema, como uso de CPU, memoria, sistema de ficheros, accesos a disco y a red,...
- **Packetbeat**: permite la monitorización de servicios y aplicaciones en tiempo real, obteniendo métricas como latencia, tiempo de respuesta, errores, patrones de acceso,...
- **Winlogbeat**: permite la recolección y envío de eventos de sistemas Windows.
- **Auditbeat**: permite la recolección y envío de métricas de auditoría de sistemas.
- **Heartbeat**: permite la monitorización de la disponibilidad y los tiempos de respuesta de los servicios.

Ahora que tenemos un conocimiento de cada componente, explicaremos el funcionamiento en conjunto de la pila ELK combinado con Filebeat para obtener los logs del sistema. Lo primero es instalar Filebeat en todas las máquinas que se quieran monitorizar y después configurarlos para que envíen los datos a una máquina central. En esta máquina central, estará Logstash recopilando todos los logs que llegan, analizándolos y filtrando para enviárselos al Elasticsearch. Cuando nosotros realicemos una consulta será Elasticsearch el que buscará los datos que coincidan con la consulta, sobre enormes cantidades de datos, devolviéndonos el resultado. Por último, Kibana nos mostrará estos datos que indexó Elasticsearch para que el usuario final pueda visualizarlos en modo de gráficos, tablas, etc..

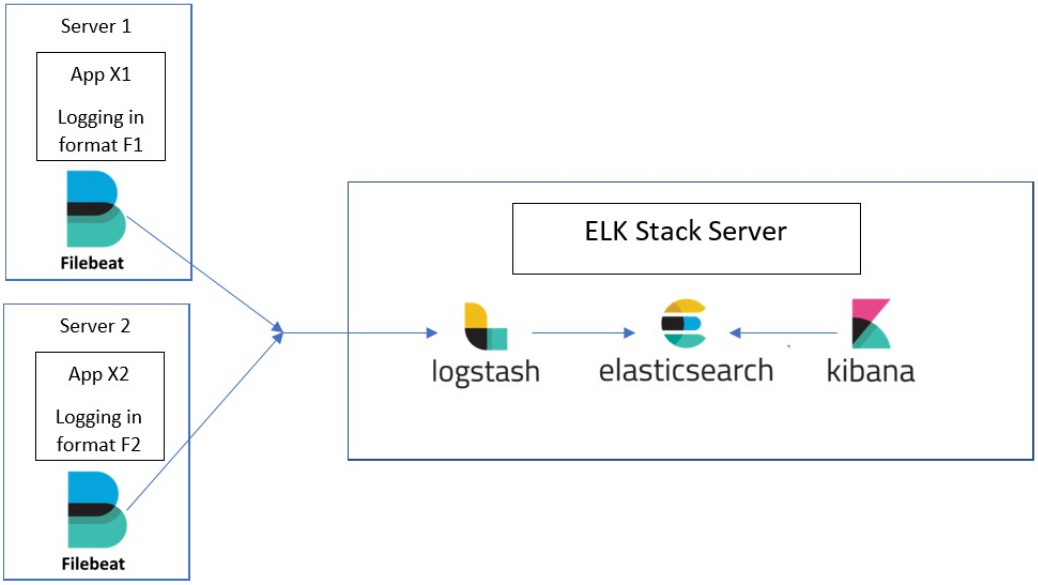

Figura 18: Proceso de Stack ELK

4.5.4. Suricata

Suricata [**key15**] es un detector de red de alto rendimiento IDS (Intrusion Detection System), IPS y seguridad de red. Está basado en un conjunto de reglas desarrolladas para supervisar el tráfico de la red y proporcionar alertas al administrador del sistema cuando se producen ciertos eventos que coinciden con las firmas o reglas.

Algunas de sus características son:

- Se puede integrar en componentes de seguridad de redes existentes.
- A diferencia de Snort, Suricata es Multi-Threaded que permite procesar una mayor cantidad de paquetes de forma simultánea.
- Con Suricata se pueden escribir reglas independientemente del puerto que un protocolo use por defecto.
- Genera logs de estadísticas y análisis de rendimiento.
- Al ser código abierto, hay muchos repositorios de alertas community para aplicarlas sobre nuestra red.

El modo de ejecución por defecto de Suricata es **autofp**. Este modo balancea automáticamente la carga de flujo, es decir, que los paquetes de cada flujo distinto se asignan a un solo hilo de detección. Los flujos se asignan a los subprocesos con el número más bajo de paquetes no procesados. Para ejecutar Suricata desde la línea de comandos sobre una interface como "ens38": [**key16**] [**key17**]

```
$ suricata -i ens38 -c /etc/suricata/suricata.yaml -s /etc/suricata/
  rules -l /var/log/suricata -D -user suricata -group suricata
```

Si necesitamos configurar el Suricata lo haremos a través del fichero */etc/suricata/suricata.yaml*. Para añadir reglas personalizadas usaremos el fichero */etc/suricata/rules/custom-rules*. Si tenemos un fichero con reglas lo copiamos en la carpeta */etc/suricata/rules*.

Los logs que genera el Suricata se pueden dividir en varios ficheros (/var/log/suricata):

- **suricata.log**: Eventos propios del servicio Suricata como inicializaciones, errores, reinicios ...
- **stats.log**: Estadísticas regulares acerca del tráfico que se ha ido analizando hasta el momento.
- **fast.log**: Eventos disparados por las reglas. Es muy útil para hacerse una idea rápida de qué ha ocurrido en la red.
- **eve.json**: Alertas detectadas y todos los detalles del tráfico capturado en formato JSON.

El formato para la creación de una regla es el siguiente:

action tcp $HOME_NET any ->$EXTERNAL_NET any (msg: "message";)

La parte roja corresponde con la acción que se debe realizar cuando se cumpla la regla (alert, drop, reject, rejectsrc, rejectdst, rejectboth). La parte verde se refiere al protocolo (tcp, udp, http, ftp, smtp,...), IP origen y IP destino, incluyendo los puertos que se deben cumplir para que salte la regla. Por último, la parte azul detalla las opciones como el mensaje de la alerta, el identificador de la regla, el tipo de classtype, opciones sobre el protocolo de la regla, etcétera. Un ejemplo de una regla:

drop http $EXTERNAL_NET any ->$HOME_NET 22 (msg:“SSH Detected”; ssh.proto; sid:1;)

Esta regla comprueba que si una petición SSH se realiza desde fuera hacia dentro sea descartada, y cuando salte muestre el mensaje “SSH Detected” en los logs.

Para que no sea muy tedioso agregar reglas una a una, existe una herramienta como **suricata-update** [**key18**] que se encarga de actualizar las reglas a través de repositorios de reglas de empresas de Threat Intelligence como Proofpoint.

5. Honeynet UVa

5.1. Arquitectura de la red Honeynet UVa

El despliegue se ha desarrollado en un entorno totalmente virtualizado mediante VMware Workstation Pro para reducir costes y centralizar el proyecto. La idea del proyecto es poderlo implementar físicamente en la red de la Universidad de Valladolid en un futuro.

El diseño de la Honeynet corresponde con una estructura de tipo híbrida virtual compuesta por el HoneyWall y HoneyDocker, soportado por un Debian 10 Buster como sistema operativo, y el equipo del intruso, que tiene como sistema operativo Kali Linux.

En este caso, el HoneyWall y el HoneyDocker están virtualizados, y los honeypots que contendrá la red serán los Dockers virtualizados dentro del propio HoneyDocker. De esta manera, tendremos una red muy flexible y de fácil escalabilidad.

El **HoneyWall** será donde recaerá el mayor peso del proyecto, debido a que este sistema tendrá la funcionalidad de filtrar, alertar y actuar contra las conexiones entrantes y salientes de la Honeynet; recopilar, centralizar y analizar la información de la red y los sistemas del HoneyDocker.
Este sistema a parte de las características que se han expuesto anteriormente, tiene la funcionalidad de router, ya que interconecta la red del atacante (WAN) con el sistema de HoneyDocker que esta dentro de la LAN.

El **HoneyDocker** estará ejecutando servicios replicados de la UVa para ser vulnerados sobre los contenedores Docker. Recopilará la información que sea generada por parte del atacante en sus contenedores y a continuación la enviará al HoneyWall.

El intruso o atacante **Kali Linux** simulará los ataques informáticos realizados desde la WAN o Internet para ver y analizar el funcionamiento de la Honeynet.

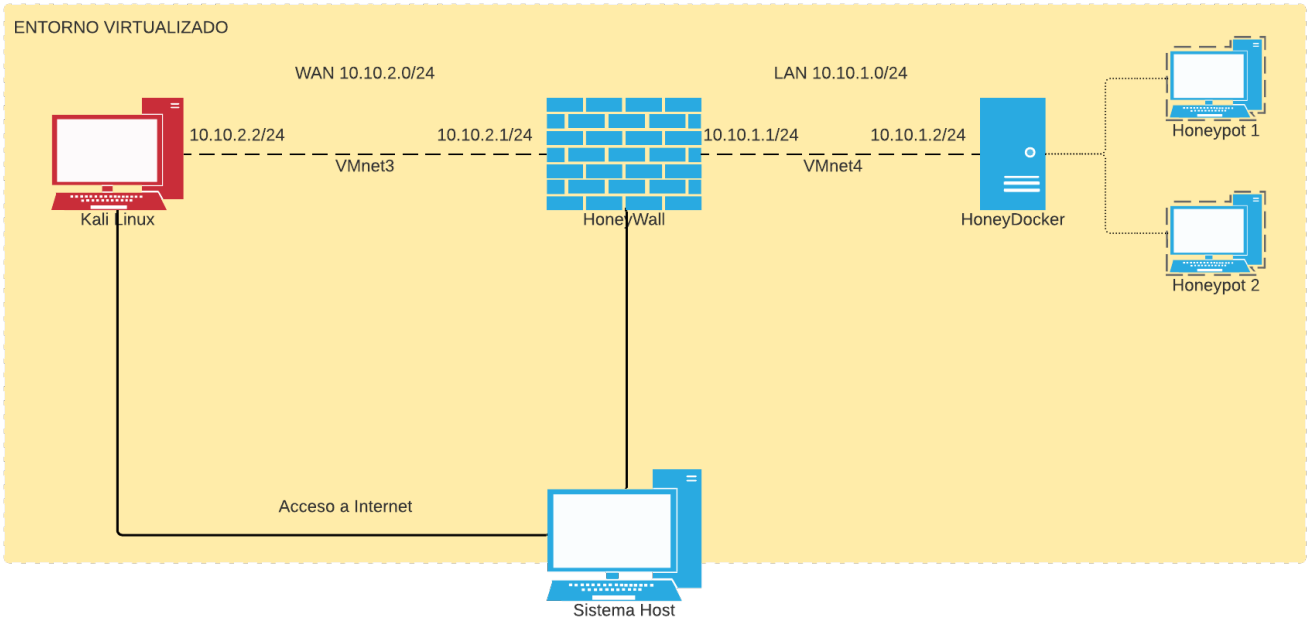

Figura 19: Diagrama de red del proyecto HoneyNet UVa

En el diagrama de red se identifican dos redes virtualizadas:

- La red que simula una WAN o Internet (10.10.2.0/24) conformada por el intruso (Kali Linux / 10.10.2.2), que atacará al HoneyDocker a través del HoneyWall (10.10.2.1).
- La red que simula una LAN (10.10.1.0/24) dentro de la UVa formada por el HoneyWall (10.10.1.1) y el HoneyDocker (10.10.1.2).
- Conexión al exterior a través del sistema Host. Es necesario que estas máquinas estén conectadas al exterior para la descarga de paquetes, actualizaciones, etcétera. El sistema HoneyDocker se conecta a Internet a través del HoneyWall una vez configuradas las IPtables.

5.2. Especificaciones de hardware

Teniendo en cuenta que este proyecto se esta desarrollando sobre un entorno totalmente virtualizado, el propio sistema host debe tener unas altas especificaciones de hardware para poder soportar 3 máquinas ejecutándose al mismo tiempo. Cada máquina virtualizada, requiere un determinado rendimiento para su buen funcionamiento, por ejemplo, la máquina HoneyDocker al ejecutar contenedores Docker y no máquinas virtuales con su sistema operativo completo, se maximiza el rendimiento de la máquina con el mínimo de características hardware. En cambio, el HoneyWall al soportar una estructura de tipo pila ELK y demás tecnologías de captura de tráfico necesita un mayor rendimiento de memoria RAM.

Nombre	Total cores por procesador	Memoria RAM	Capacidad HDD	Interfaces	Sistema Operativo
HoneyWall	4	4 GB	30 GB	VMnet4, VMnet3, VMnet0 (host-only)	Debian 10 Buster
HoneyDocker	4	2 GB	30 GB	VMnet4	Debian 10 Buster
Kali Linux	4	2 GB	40 GB	VMnet3, VMnet0 (host-only)	Kali Linux 2020.4
Sistema Host	Intel Core i5-6600K 3.50 GHz	16 GB	1 TB HDD	Realtek PCIe GbE Family Controller	Windows 10 Pro 64 bits

Figura 20: Especificaciones Hardware

5.3. Configuración de las conexiones de red

Para configurar las conexiones entre los diferentes sistemas, se ha modificado el fichero */etc/network/interfaces* de cada sistema Linux (HoneyWall , HoneyDocker y Kali Linux). En la red WAN que contiene al Kali y HoneyWall esta definida la dirección IP de red 10.10.2.0/24 y en la red LAN que se encuentra el HoneyDocker y HoneyWall esta definida la dirección IP de red 10.10.1.0/24. Para que el Kali Linux pueda acceder a los servicios del HoneyDocker es necesario configurar lo siguiente:

- El HoneyWall tiene que ser capaz de hacer enrutamiento por NAT para cambiar paquetes entre dos redes diferentes.

- El HoneyWall debe tener configurado port forwarding para redirigir un puerto de red de un nodo de red a otro.
- Se tiene que configurar un DNS en el HoneyWall para resolver los nombres de dominio.

5.3.1. Configuración NAT en el HoneyWall

Para configurar NAT en el HoneyWall, lo haremos a través de **IPtables**, que es una herramienta de Linux que se encarga de filtrar los paquetes de red. Si queremos que el HoneyWall funcione como router, debemos poner la variable del sistema **'IP forwarding'** a 1 para IPv4 [**key30**].

```
$ echo "1" > /proc/sys/net/ipv4/ip_forward
```

Es importante, tener en cuenta que cada vez que reiniciemos el sistema, debemos reactivarlo. Por lo tanto, se puede crear un script en bash para cuando necesitemos activar o desactivar el ip forward y las reglas del firewall. En el transcurso de este proyecto iremos añadiendo nuevas iptables sobre estos script que llamaremos **iptablesUp.sh** para activar y **iptablesDown.sh** para desactivar el firewall.

Por defecto, la política que tendrá el firewall es de aceptar todo tipo de tráfico y después se añaden reglas explícitamente para filtrar los paquetes.

```
$ iptables -P INPUT ACCEPT
$ iptables -P OUTPUT ACCEPT
$ iptables -P FORWARD ACCEPT
$ iptables -t nat -P PREROUTING ACCEPT
$ iptables -t nat -P POSTROUTING ACCEPT
```

Debemos añadir las siguientes reglas para poder redireccionar mediante NAT los paquetes que vienen de nuestra red interna 10.10.1.0/24 y salen al exterior hacia la WAN del Kali (ens37) y hacia Internet a través de nuestro sistema host (ens33).

```
$ iptables -t nat -A POSTROUTING -s 10.10.1.0/24 -o ens37 -j MASQUERADE
$ iptables -t nat -A POSTROUTING -s 10.10.1.0/24 -o ens33 -j MASQUERADE
```

Con esta configuración el HoneyDocker es capaz de comunicarse con el Kali Linux que se encuentra en otra red distinta.

```
root@HoneyDocker:~# ping 10.10.2.2
PING 10.10.2.2 (10.10.2.2) 56(84) bytes of data.
64 bytes from 10.10.2.2: icmp_seq=1 ttl=63 time=1.34 ms
64 bytes from 10.10.2.2: icmp_seq=2 ttl=63 time=2.46 ms
64 bytes from 10.10.2.2: icmp_seq=3 ttl=63 time=3.69 ms
64 bytes from 10.10.2.2: icmp_seq=4 ttl=63 time=3.84 ms
^C
--- 10.10.2.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 14ms
rtt min/avg/max/mdev = 1.343/2.833/3.839/1.014 ms
root@HoneyDocker:~#
```

Figura 21: Comprobación de conectividad de HoneyDocker a Kali Linux

Vemos que conecta perfectamente con Kali Linux. Sin embargo, si lo hacemos al revés no funcionaría, ya que la conexión de Kali Linux a HoneyDocker debe ser redirigida por el HoneyWall hacia un puerto concreto del HoneyDocker. Esto se puede implementar mediante IPtables con port forwarding NAT.

5.3.2. Configurar Port Forwarding en el HoneyWall

Port Fordwarding es el proceso de redireccionar con NAT un puerto específico de un host, en este caso el HoneyWall, hacia otro puerto de otro equipo de la red, como puede ser el HoneyDocker.

Como todavía no tenemos ejecutando ningún contenedor en el HoneyDocker, vamos a redireccionar el puerto 333 del HoneyWall hacia el puerto 80 del HoneyDocker para ver el funcionamiento del Port Forwarding.

```
$ iptables -A FORWARD -m state -p tcp -d 10.10.1.2 - -dport 80 - -state
  NEW,ESTABLISHED,RELATED -j ACCEPT

$ iptables -t nat -A PREROUTING -p tcp --dport 333 -j DNAT --to-
 destination 10.10.1.2:80
```

Si activamos estas reglas, desde Kali Linux podemos ejecutar desde el navegador http://10.10.2.1:333 que nos mostrará la página que está alojada en el puerto 80 del HoneyDocker. **Estas reglas son un ejemplo o comprobación de que funciona y no se añadirán al script final de iptablesUp.sh** .

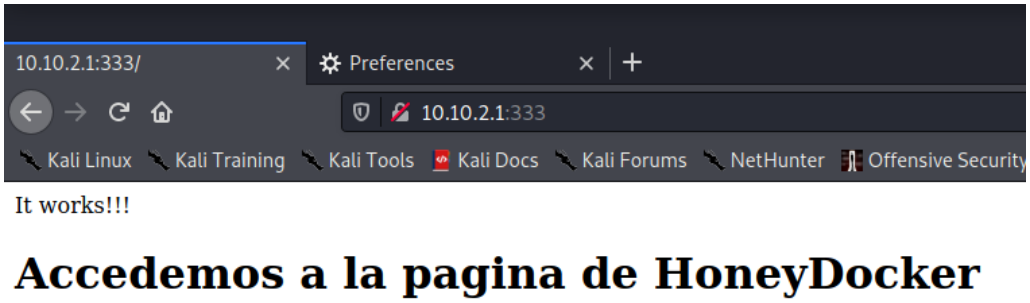

Figura 22: Port Forwarding 10.10.2.1:333 ->10.10.1.2:80

Como se puede ver, el atacante no conoce la IP final ni el recorrido que hace dentro de la LAN, pero mediante traceroute sí que puede saber los saltos que realiza el paquete. Si realizamos traceroute sobre el puerto 333 nos dará dos saltos por el HoneyWall y el HoneyDocker y si probamos con otro puerto como el 334 solo hace un salto, ya que no hay redirección aplicada a ese puerto.

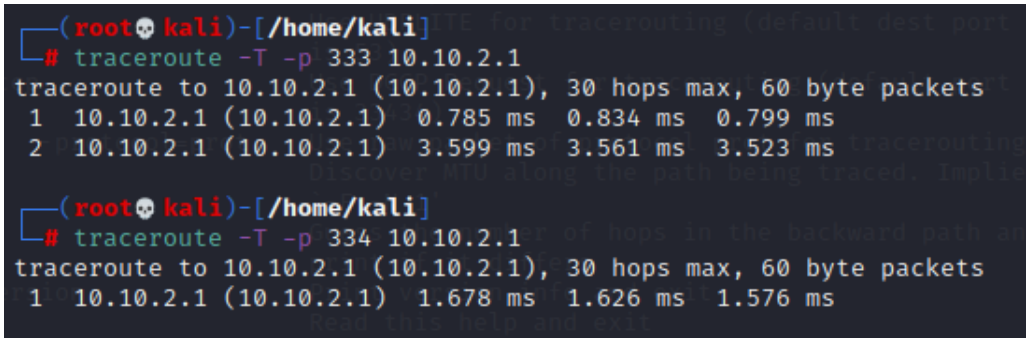

Figura 23: Traceroute sobre puerto con redirección / Traceroute sobre puerto normal

5.3.3. Configuración DNS en el HoneyWall

Es necesario tener configurado un DNS en la red que resuelva los nombres de dominio y los asocie con una IP. El HoneyWall se va a encargar de esto mediante **bind9**, que es un software de código libre para implementar DNS en servidores Linux.

Con este servicio DNS conseguiremos resolver los dominios de **aulas.inf.uva.es** y **jair.lab.inf.uva.es** para que se asocien con la IP del HoneyDocker. Tendremos que configurar los ficheros:

- */etc/bind/named.conf.options* : Sobre este fichero tenemos que añadir los forwarders para que en caso de que nuestro servidor DNS no pueda resolver esa petición se envíe a un servidor externo que sí pueda, como pueden ser los de google (8.8.8.8 y 8.8.4.4).
- */etc/bind/named.conf.local* : En este fichero añadiremos las zonas inf.uva.es y 1.10.10.in-addr.arpa que queremos que resuelva de forma directa e inversa respectivamente.
- */etc/bind/zones/db.inf.uva.es y /etc/bind/zones/db.10.10.1* : Sobre el fichero db.inf.uva.es configuramos la resolución directa añadiendo las líneas de name server (NS) y direcciones a resolver (A). En cambio, sobre el db.10.10.1 añadimos la línea de name server (NS) y para resolver las direcciones de forma inversa (PTR).

Además de configurar el servidor DNS del HoneyWall, debemos cambiar el DNS sobre el que resuelven las peticiones los equipos de la red (Kali Linux, HoneyWall y HoneyDocker). Para ello, añadimos en el fichero */etc/resolv.conf* lo siguiente:

```
nameserver 10.10.2.1
```

Con la parte del DNS configurada, ya podemos resolver los dominios que queremos simular en nuestra HoneyNet.

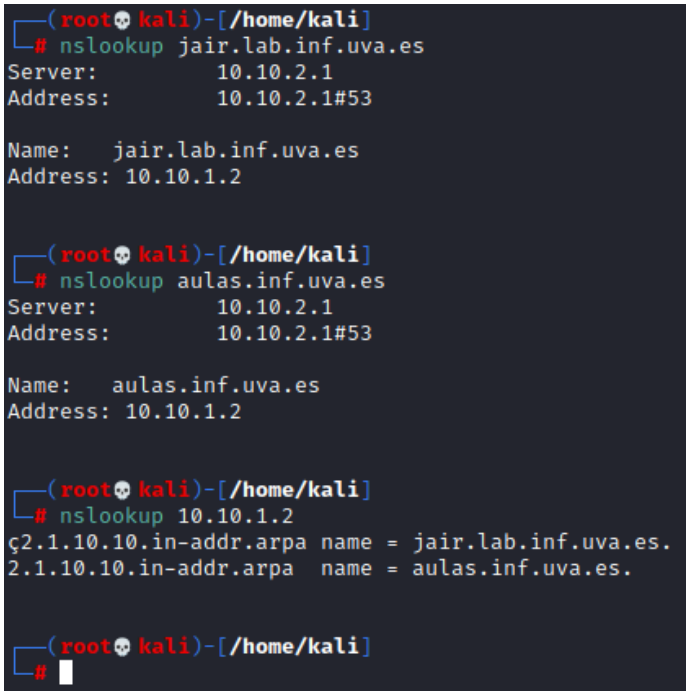

Figura 24: Resolución de aulas.inf.uva.es y jair.lab.inf.uva.es desde Kali Linux

5.4. Instalación de Docker y sus contenedores en el HoneyDocker

5.4.1. Instalación de Docker

La instalación de Docker [**key34**] se realiza mediante los repositorios oficiales de Debian y la última versión de Docker es la 20.10.5. Primero instalamos las dependencias que nos permitan usar paquetes sobre HTTPS:

```
$ sudo apt-get install apt-transport-https ca-certificates curl gnupg2
software-properties-common
```

Añadimos la GPG key del repositorio oficial de Docker a nuestro sistema:

```
$ curl -fsSL https://download.docker.com/linux/debian/gpg | sudo
apt-key add -
```

Añadimos el repositorio de Docker a /etc/apt/sources.list:

```
$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com
 /linux/debian $(lsb_release -cs) stable"
```

Volvemos a actualizar los paquetes con apt-get update e instalamos el paquete docker-ce:

```
$ sudo apt-get install docker-ce
```

Una vez instalado comprobamos si está encendido, en caso de que no lo esté es así lo ejecutamos con systemctl start docker y vemos su estado con systemctl status docker.

```
root@HoneyDocker:/# systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2021-03-16 17:30:47 CET; 19h ago
     Docs: https://docs.docker.com
 Main PID: 751 (dockerd)
```

Figura 25: Status de servicio Docker

5.4.2. Docker nginx de aulas.inf.uva.es

Para hacer el honeypot de aulas.inf.uva.es sobre un Docker necesitamos la estructura y el código de la web para replicar la web sobre el Docker. Para obtener estos recursos, utilizamos HTTrack Website Copier que sirve para la captura del código público de sitios web.

Descargamos el programa, creamos un nuevo proyecto y añadimos la url de la página que queremos que nos descargue todo su contenido, en este caso "http://aulas.inf.uva.es".

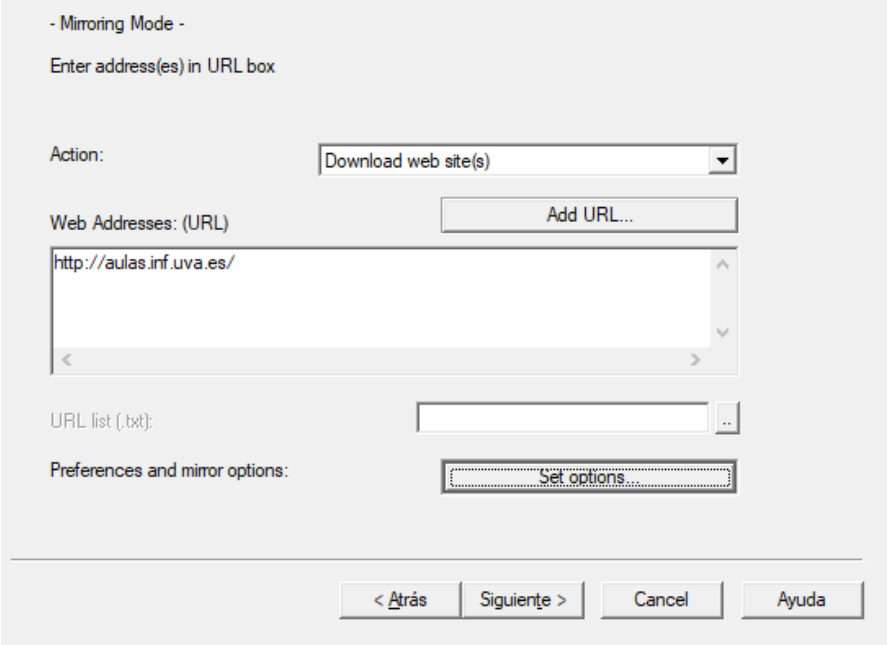

Figura 26: HTTrack Website Copier

Si vamos a la ruta donde se ha descargado la página y abrimos el index desde el navegador, nos muestra un moodle idéntico al de aulas. Esta página no tiene ninguna conexión con la base de datos por detrás, por lo que el intruso no podrá loguearse con ningún usuario. Además, puede que algunas funciones del moodle no funcionen y esto haga sospechar al intruso. Es por esto que el Honeypot que montaremos será de baja interacción.

Después de descargar la página, la alojaremos en un servicio web montado en un contenedor Dockercon una imagen httpd de nginx. Primero descargamos la imagen mediante el siguiente comando:

```
$ docker pull httpd
```

Copiamos el contenido de la carpeta de aulas dentro del HoneyDocker en la ruta /opt/aulas. Creamos el contenedor de httpd enlazando el puerto 3000 del HoneyDocker con el puerto 80 del contenedor httpd y, con la opción -v, montamos la carpeta /opt/aulas del HoneyDocker en la ruta /usr/local/apache2/htdocs del contenedor.

```
$ docker run -d --name aulas -p 3000:80 -v /opt/aulas/:/usr/local
/apache2/htdocs httpd
```

Nuestro contenedor estará activo en el puerto 3000 del HoneyDocker y para que el Kali Linux acceda a él debemos añadir IPtables a modo de port forwarding. Redirigiremos el tráfico entrante del puerto 80 del HoneyWall al puerto 3000 del HoneyDocker:

```
$ iptables -t nat -A PREROUTING -p tcp --dport 80 -j DNAT
--to-destination 10.10.1.2:3000

$ iptables -A FORWARD -m state -p tcp -d 10.10.1.2 --dport 3000
--state NEW,ESTABLISHED,RELATED -j ACCEPT
```

Si ahora buscamos aulas.inf.uva.es desde el navegador del Kali Linux, accederíamos a la página que está alojada en el contenedor del HoneyDocker y el intruso vería una página casi idéntica a la original.

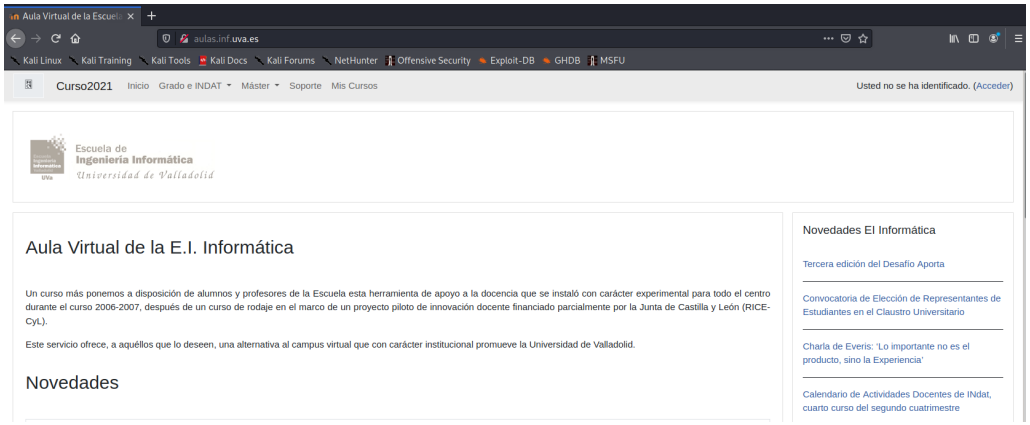

Figura 27: Réplica de aulas.inf.uva.es vista desde el Kali Linux

5.4.3. Docker SSH de jair.lab.inf.uva.es

La idea de este contenedor es replicar el dominio de jair.lab.inf.uva.es que contiene un servicio SSH para poder conectarse al sistema Linux de cada alumno. Para replicar el dominio descargaremos una imagen llamada Cowrie [**key19**]. Este contenedor está diseñado para ofrecer una interacción media de servicios SSH/Telnet de manera segura y creíble.
Cowrie puede simular un sistema de archivos completo con la posibilidad de crear, modificar y borrar archivos de este sistema simulado. El atacante podrá acceder a este sistema simulado usando SSH y tendrá la impresión de que está en un sistema operativo real.
La configuración por defecto de Cowrie es que el verdadero puerto de SSH para administrar el sistema se encuentra en el puerto 22222. Dejando libre el puerto SSH(22) y Telnet(23) por defecto. Una vez los bots y atacantes consigan acceder al sistema se les redirigirá a los puertos donde se encuentra simulado el sistema operativo falso.

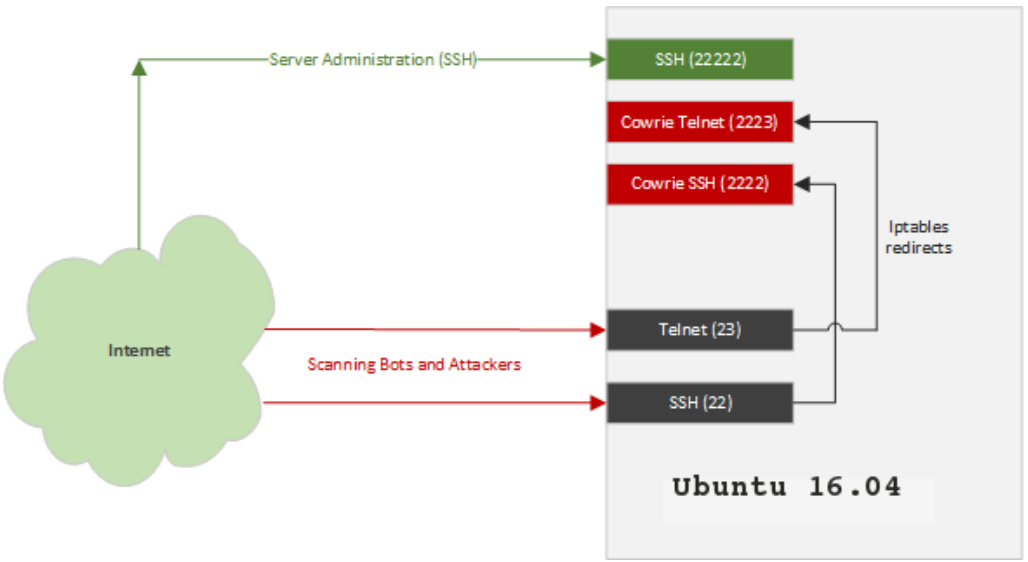

Figura 28: Funcionamiento de HoneyPot Cowrie [**key20**]

Ejecutamos este contenedor para enlazar los puertos 22 y 22222 del contenedor, con los puertos 3001 y

22222 del HoneyDocker, respectivamente.

```
$ docker pull cowrie/cowrie
$ docker run -d --name jair -h "jair" -p 3001:22 -p 22222:22222 cowrie/
 cowrie
```

Con esto iniciaremos el contenedor. Para acceder por primera vez al sistema real lo haremos mediante un comando de Docker ya que debemos cambiar la contraseña de root para acceder por SSH:

```
$ docker exec -it --user root jair /bin/bash
```

Una vez dentro, las configuraciones que se han realizado para este TFG son:

- **Cambiar contraseña de root a "JairDocker":**

  ```
  $ passwd root
  ```

- **Añadir usuarios válidos a /etc/userdb.txt:**

  Modificar o crear el fichero /etc/userdb.txt (el fichero userdb.example es una copia de ese fichero). En este fichero podemos añadir los usuarios válidos para nuestro sistema simulado y las contraseñas que estarán permitidas para acceder por SSH, en nuestro caso añadimos:

  - root:x:JairRoot
  - test:x:test

- **Añadir la línea del usuario test en el /cowrie/cowrie-git/honeyfs/etc/passwd:**

  - test:x:1000:1000::/home/test:/bin/sh

- **Añadir la línea del grupo test en el /cowrie/cowrie-git/honeyfs/etc/group:**

  - test:x:1000:

Como hemos hecho con el contenedor httpd anterior, debemos aplicar port forwarding sobre el HoneyWall. Para esto, se ha cambiado el puerto del SSH(22) del HoneyWall al 2222 para dejarlo libre. Ahora con el puerto 22 libre añadimos las siguientes IPtables del port forwarding:

```
$ iptables -t nat -I PREROUTING -p tcp -d 10.10.2.1 --dport 22 -j DNAT
--to 10.10.1.2:3001

$ iptables -A FORWARD -m state -p tcp -d 10.10.1.2 --dport 3001
--state NEW,ESTABLISHED,RELATED -j ACCEPT
```

Ahora desde Kali Linux, nos podemos conectar al contenedor SSH que está en el HoneyDocker simulando a jair con el usuario root y test.

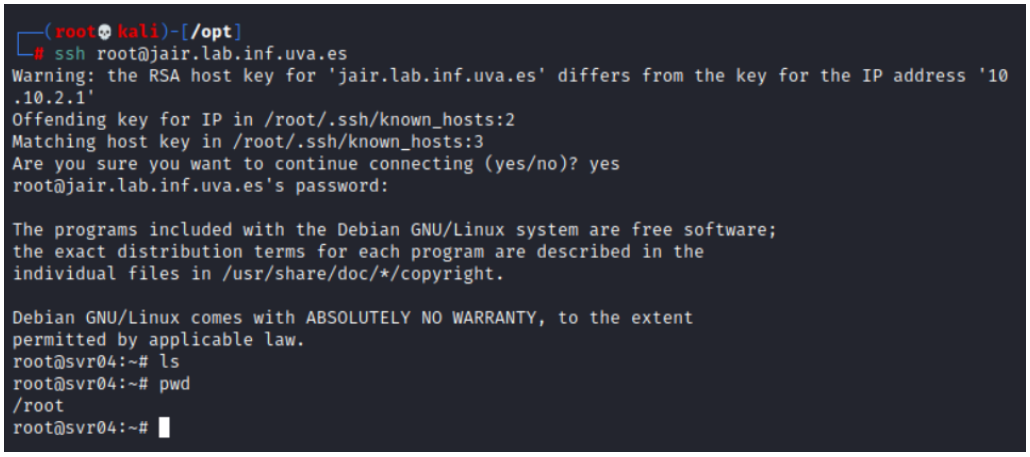

Figura 29: Conexión SSH al contenedor Docker de Cowrie desde el Kali Linux

5.5. Instalación de Suricata

Como se ha explicado anteriormente, Suricata [**key21**] es un sistema de detección y prevención de intrusos que registra todos los eventos generando alertas cuando se cumplen las reglas configuradas. Primero instalaremos sus dependencias:

```
$ apt-get install libpcre3 libpcre3-dbg libpcre3-dev build-essential
libpcap-dev libnet1-dev libyaml-0-2 libyaml-dev pkg-config zlib1g
zlib1g-dev libcap-ng-dev libcap-ng0 make libmagic-dev libjansson-dev
libnss3-dev libgeoip-dev liblua5.1-dev libhiredis-dev libevent-dev
python-yaml rustc cargo
```

Agregamos los paquetes de Suricata de OSIF (Open Information Security Foundation) con la última versión estable.

```
$ add-apt-repository ppa:oisf/suricata-stable
```

Actualizamos la cache de paquetes e instalamos el paquete suricata y suricata-update:

```
$ apt-get install suricata suricata-update
```

El siguiente paso es ajustar la configuración de Suricata mediante el fichero **suricata.yaml**.

- **Cambiar la interfaz de red:** Por defecto, la interfaz de red que esta configurada es eth0. Reemplazaremos eth0 de toda configuración en el archivo suricata.yaml y pondremos, en nuestro caso **ens38**, la interfaz de la INTRANET (10.10.1.0/24). Si se quiere añadir otra interfaz más para monitorizar, añadimos una etiqueta **-interface:** [**nueva_interface**] debajo de la "principal".

- **Definir el HOME_NET y EXTERNAL_NET:** Las reglas funcionan teniendo en cuenta si el origen/destino viene de fuera de la red o de dentro. Para ello debemos definir en la variable HOME_NET qué direccionamiento IP es el interno de la red y para EXTERNAL_NET podemos utilizar la negación con una exclamación para referirnos a todas las redes que no estén en HOME_NET.
  En este proyecto, la HOME_NET tiene el valor **HOME_NET: "[10.10.1.0/24]"** y la EXTERNAL_NET tiene **EXTERNAL_NET: "!$HOME_NET"**.

- **Configurar los tipos de logs de Suricata:** En el fichero **suricata.yaml** decidimos los logs (output) que generará Suricata cada vez que registre un tipo de evento o alerta. Por ejemplo, si tenemos activo el **dns-log**, podremos recopilar las peticiones y respuestas DNS que capture la interfaz del Suricata. Sabiendo esto, es necesario seleccionar solo los que se vayan a utilizar para evitar un almacenamiento masivo de logs innecesarios y para que, a la hora de usar ELK, no veamos ruido en los logs y sea más eficientes las búsquedas.
  Los tipos de logs que se han seleccionado son: **eve.json, http.log, fast.log y suricata.log**

Una vez configurado suricata.yaml, iniciamos el servicio Suricata y comprobamos su estado:

```
$ systemctl start suricata
$ systemctl status suricata
```

Otro punto importante a parte de configurar Suricata, es actualizar las firmas o reglas que utiliza Suricata para la detección. Con la herramienta **suricata-update** podemos realizar estas tareas con el siguiente comando:

```
$ suricata-update
```

Al ejecutar este comando, por defecto se obtiene el conjunto de reglas de ETOpen (Open Source), instalándolas en **/var/lib/suricata/rules**. Como resultado podemos ver las reglas que nos ha agregado, eliminado, modificado, desactivadas, etc..

```
10/5/2021 -- 20:58:47 - <Info> -- Loaded 36704 rules.
10/5/2021 -- 20:58:47 - <Info> -- Disabled 2538 rules.
10/5/2021 -- 20:58:47 - <Info> -- Enabled 0 rules.
10/5/2021 -- 20:58:47 - <Info> -- Modified 0 rules.
10/5/2021 -- 20:58:47 - <Info> -- Dropped 0 rules.
10/5/2021 -- 20:58:48 - <Info> -- Enabled 292 rules for flowbit dependencies.
10/5/2021 -- 20:58:48 - <Info> -- Backing up current rules.
10/5/2021 -- 20:58:55 - <Info> -- Writing rules to /var/lib/suricata/rules/suricata.rules: total: 36
704; enabled: 24279; added: 154; removed 26; modified: 3756
10/5/2021 -- 20:58:56 - <Info> -- Testing with suricata -T.
10/5/2021 -- 20:59:37 - <Info> -- Done.
```

Figura 30: Comando suricata-update

Los conjuntos de reglas son las listas de reglas que disponen los proveedores de seguridad para empresas [**key22**]. Si necesitamos ver qué conjuntos de reglas utilizamos usaremos el comando **suricata-update list-source-enabled** y, para ver todos los conjuntos de reglas que dispone Suricata, usamos **suricata-update list-source**.

```
Name: scwx/security
  Vendor: Secureworks
  Summary: Secureworks suricata-security ruleset
  License: Commercial
  Parameters: secret-code
  Subscription: https://www.secureworks.com/contact/ (Please reference CTU Countermeasures)
Name: sslbl/ssl-fp-blacklist
  Vendor: Abuse.ch
  Summary: Abuse.ch SSL Blacklist
  License: Non-Commercial
```

Figura 31: Ejemplo de conjunto de reglas Commercial y Non-Commercial

Añadimos 3 conjuntos de reglas Non-Commercial a la lista de nuestro Suricata:

```
$ suricata-update enable-source oisf/trafficid
$ suricata-update enable-source sslbl/ssl-fp-blacklist
```

```
$ suricata-update enable-source ptresearch/attackdetection
```

Por defecto Suricata tiene habilitado el conjunto de reglas et/pro que es Commercial, por lo que podemos deshabilitarla (suricata-update disable-source et/pro) o eliminarla (suricata-update remove-source et/pro).

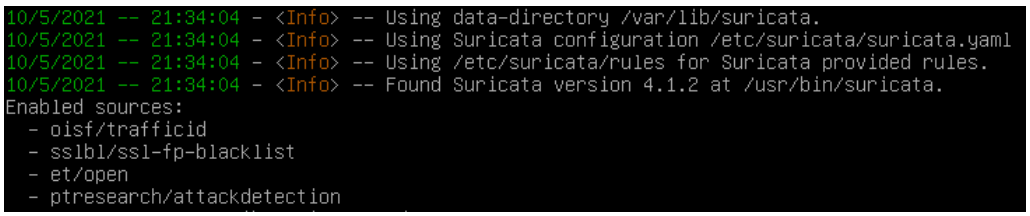

Figura 32: Conjunto de reglas de nuestro Suricata (suricata-update list-source-enabled)

Reiniciamos el servicio Suricata para que se actualicen los nuevos cambios.

5.6. Instalación de Stack ELK

ELK [**key23**] requiere Java para funcionar, así que podemos instalar la versión de Java 8 o 11, en este caso, instalamos Java 11:

```
$ apt install openjdk-11-jdk -y
```

Para añadir el repositorio de Elastic Stack debemos instalar la clave de firma PGP de Elastic Stack y después el repositorio ELK APT:

```
$ wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | apt-
 key add -
$ apt install apt-transport-https
$ echo "deb https://artifacts.elastic.co/packages/7.x/apt stable main"
 | tee /etc/apt/sources.list.d/elastic-7.x.list
```

Actualizamos la caché de la lista de paquetes con **apt update**.

Una vez agregado el repositorio ELK, iremos instalando y configurando cada componente por partes, empezando con el Elasticsearch:

```
$ apt install elasticsearch
```

El archivo de configuración principal del Elasticsearch se encuentra en */etc/elasticsearch/elasticsearch.yml*. Configuramos las siguientes etiquetas **network.host** que indican por dónde va a ser accesible el Elasticsearch, se añade **0.0.0.0** para que sea accesible desde la 10.10.1.0/24 y la 192.168.1.0/24, y **http.port** que indica dónde va a estar escuchando Elasticsearch, por defecto es el **9200**.

Debemos establecer el tamaño del heap de memoria de JVM que usaremos en */etc/elasticsearch/jvm.options*. En nuestro caso usaremos 1 GB RAM por lo que cambiamos de -Xms512m y -Xmx512m a **-Xms1g** y **-Xmx1g**, respectivamente.

Ahora iniciamos el servicio de Elasticsearch y debemos ver el estado de este:

```
$ systemctl enable elasticsearch
$ systemctl start elasticsearch
$ systemctl status elasticsearch
```

```
● elasticsearch.service - Elasticsearch
   Loaded: loaded (/lib/systemd/system/elasticsearch.service; enabled; vendor preset: enabled)
   Active: active (running) since Sun 2021-04-25 19:54:28 CEST; 1 weeks 2 days ago
     Docs: https://www.elastic.co
 Main PID: 61918 (java)
    Tasks: 109 (limit: 4673)
   Memory: 1.4G
   CGroup: /system.slice/elasticsearch.service
           ├─61918 /usr/share/elasticsearch/jdk/bin/java -Xshare:auto -Des.networkaddress.cache.ttl=
           └─62105 /usr/share/elasticsearch/modules/x-pack-ml/platform/linux-x86_64/bin/controller

Warning: Journal has been rotated since unit was started. Log output is incomplete or unavailable.
```

Figura 33: Status del servicio Elasticsearch

Una vez instalado el Elasticsearch, instalamos el Kibana:

```
$ apt install kibana
```

El archivo de configuración principal de Kibana se encuentra en */etc/kibana/kibana.yml*. Configuramos la dirección IP (**server.port: "0.0.0.0"**) y el puerto (**server.port: 5601**) para acceder al Kibana. Además cambiaremos la configuración de como se conecta Kibana a Elasticsearch con **elasticsearch.hosts: ["http:127.0.0.1:9200"]**.

Se puede implementar una autenticación básica para acceder al Kibana mediante **elasticsearch.username: "user"** y **elasticsearch.password: "pass"**. En este caso, se decidió no utilizar credenciales.

Igual que hicimos con el Elasticsearch, iniciamos el servicio de Kibana y vemos su estado:

```
$ systemctl enable kibana
$ systemctl start kibana
$ systemctl status kibana
```

```
● kibana.service - Kibana
   Loaded: loaded (/etc/systemd/system/kibana.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2021-05-05 17:50:06 CEST; 15min ago
     Docs: https://www.elastic.co
 Main PID: 108166 (node)
    Tasks: 11 (limit: 4673)
   Memory: 342.7M
   CGroup: /system.slice/kibana.service
           └─108166 /usr/share/kibana/bin/../node/bin/node /usr/share/kibana/bin/../src/cli/dist --…

may 05 17:50:06 HoneyWall systemd[1]: Started Kibana.
```

Figura 34: Status del servicio Kibana

Ahora se puede acceder al Kibana desde el PC Host (http://192.168.1.45:5601)

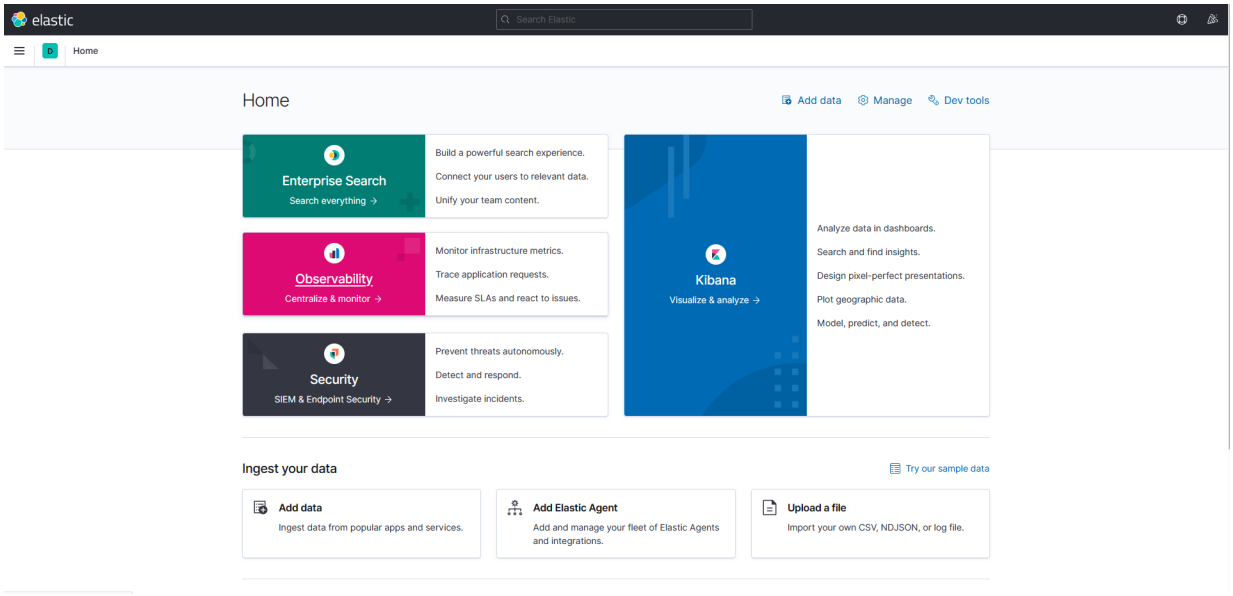

Figura 35: Página inicial Kibana

Por último, instalaremos el servicio Logstash:

```
$ apt install logstash
```

De momento no configuraremos ninguna entrada, filtro o salida del Logstash, por lo que iniciamos el servicio y revisamos su estado:

```
$ systemctl enable logstash
$ systemctl start logstash
$ systemctl status logstash
```

```
● logstash.service - logstash
   Loaded: loaded (/etc/systemd/system/logstash.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2021-05-05 18:36:06 CEST; 16s ago
 Main PID: 115716 (java)
    Tasks: 18 (limit: 4673)
   Memory: 630.7M
   CGroup: /system.slice/logstash.service
           └─115716 /usr/share/logstash/jdk/bin/java -Xms1g -Xmx1g -XX:+UseConcMarkSweepGC -XX:CMSIn

may 05 18:36:06 HoneyWall systemd[1]: Started logstash.
may 05 18:36:06 HoneyWall logstash[115716]: Using bundled JDK: /usr/share/logstash/jdk
may 05 18:36:06 HoneyWall logstash[115716]: OpenJDK 64-Bit Server VM warning: Option UseConcMarkSwee
may 05 18:36:22 HoneyWall logstash[115716]: Sending Logstash logs to /var/log/logstash which is now
may 05 18:36:22 HoneyWall logstash[115716]: [2021-05-05T18:36:22,961][INFO ][logstash.runner
may 05 18:36:22 HoneyWall logstash[115716]: [2021-05-05T18:36:22,971][INFO ][logstash.runner
```

Figura 36: Status del servicio Logstash

Dada la configuración del Stack ELK, se han dejado abiertos los puertos 5601 (Kibana), 9200 (Elasticsearch) y 9300 (Java) accesibles desde la red del Kali Linux. Por lo tanto, creamos unas reglas con IPtables para bloquear cualquier petición dirigida a esos puertos desde esa red. Además de bloquear estas peticiones, especificamos que nos genere un evento por cada petición en los logs del sistema para que más tarde lo visualicemos desde Kibana.

```
 $ iptables -A INPUT -i ens37 -p tcp --destination-port 5601 -j LOG --
log-prefix 'Access Port Blocked (5601)'
 $ iptables -A INPUT -i ens37 -p tcp --destination-port 5601 -j DROP

 $ iptables -A INPUT -i ens37 -p tcp --destination-port 9200 -j LOG --
log-prefix 'Access Port Blocked (9200)'
 $ iptables -A INPUT -i ens37 -p tcp --destination-port 9200 -j DROP

 $ iptables -A INPUT -i ens37 -p tcp --destination-port 9300 -j LOG --
log-prefix 'Access Port Blocked (9300)'
 $ iptables -A INPUT -i ens37 -p tcp --destination-port 9300 -j DROP
```

5.7. Instalación de Filebeat

Como ya se explicó, ELK puede usar Beats para enviar datos de varias fuentes y presentárselos a Logstash o Elasticsearch. Instalamos el paquete Filebeat:

```
$ apt install filebeat
```

El archivo de configuración predeterminado se encuentra en */etc/filebeat/filebeat.yml*. Si queremos que los datos que envíe Filebeat vayan directos al Elasticsearch, no debemos modificar el fichero. En cambio, si queremos que Filebeat envíe sus datos al Logstash para que los procese, debemos descomentar las líneas **output.logstash:** y **hosts:** [**"127.0.0.1:5044"**], y comentar las líneas **output.elasticsearch:** y **hosts:** [**"127.0.0.1:9200"**]. En nuestro caso, el flujo será Filebeat → Elasticsearch por lo que nos quedamos con las líneas del output.elasticsearch.

Iniciamos el servicio Filebeat y listamos los módulos que se pueden activar:

```
$ systemctl start filebeat
$ filebeat modules list
```

El primer módulo que usaremos será el **Suricata**, por lo que para activarlo ejecutamos el siguiente comando:

```
$ filebeat modules enable suricata
```

Figura 37: Lista de módulos Filebeat (Enabled/Disabled)

Desde */etc/filebeat/modules.d/suricata.yml*, configuramos el módulo Suricata para que sepa qué logs de Suricata enviar. Los logs que revisaremos son eve.json, http.log y fast.log, pero si se necesitan más logs se añaden a este fichero. Desde Kibana, tenemos que crear un patrón para los índices (index pattern) para que Kibana sepa qué datos debe procesar. Para definir el primero accedemos a **Management → Stack Management → Index Patterns → Create index pattern** y escribimos una expresión regular que seleccione todas las fuentes relacionadas con Filebeat como **"filebeat*"**.

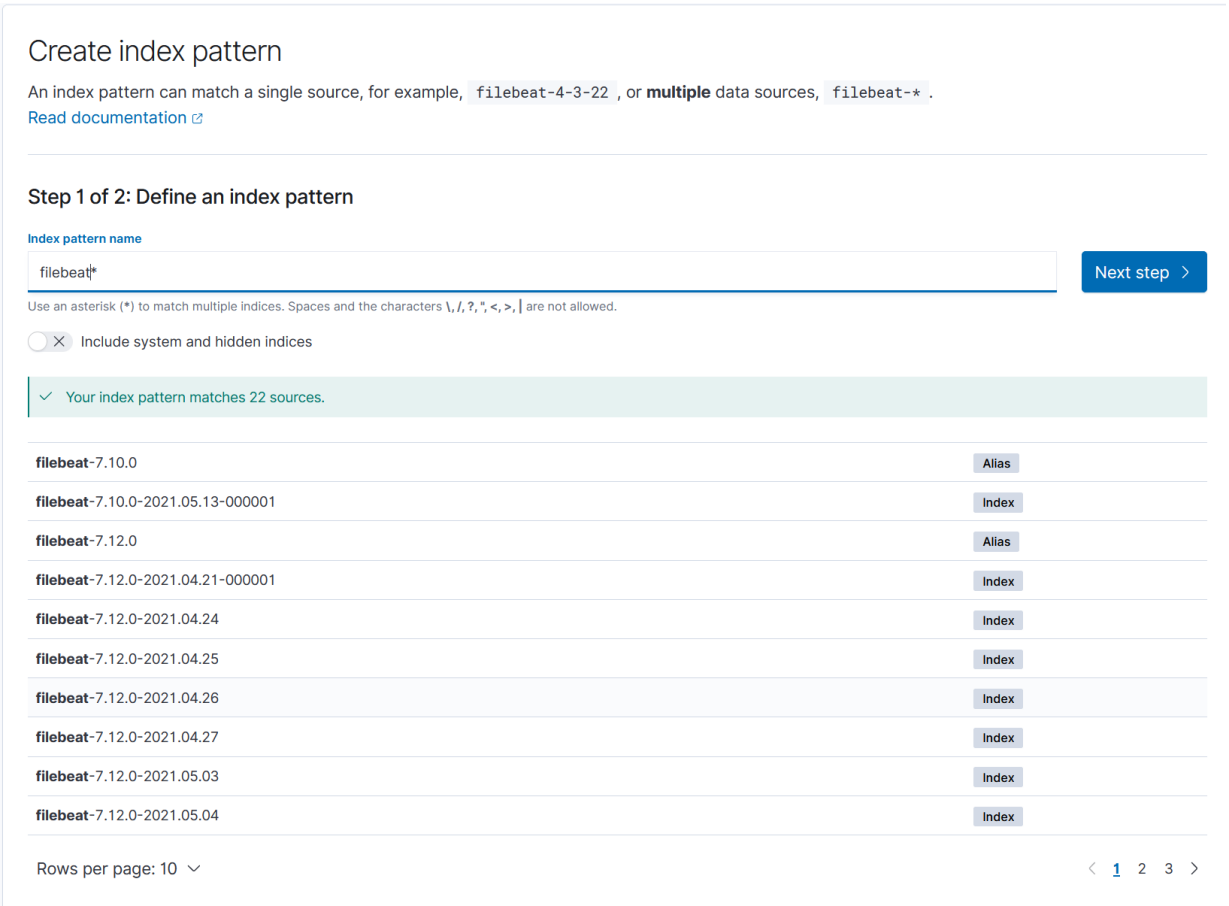

Figura 38: Index pattern Filebeat*

Con esto ya tendríamos creado el index pattern. Si generamos algo de tráfico desde el Kali Linux, y accedemos al discover "filebeat*"de Kibana, podremos ver lo siguiente:

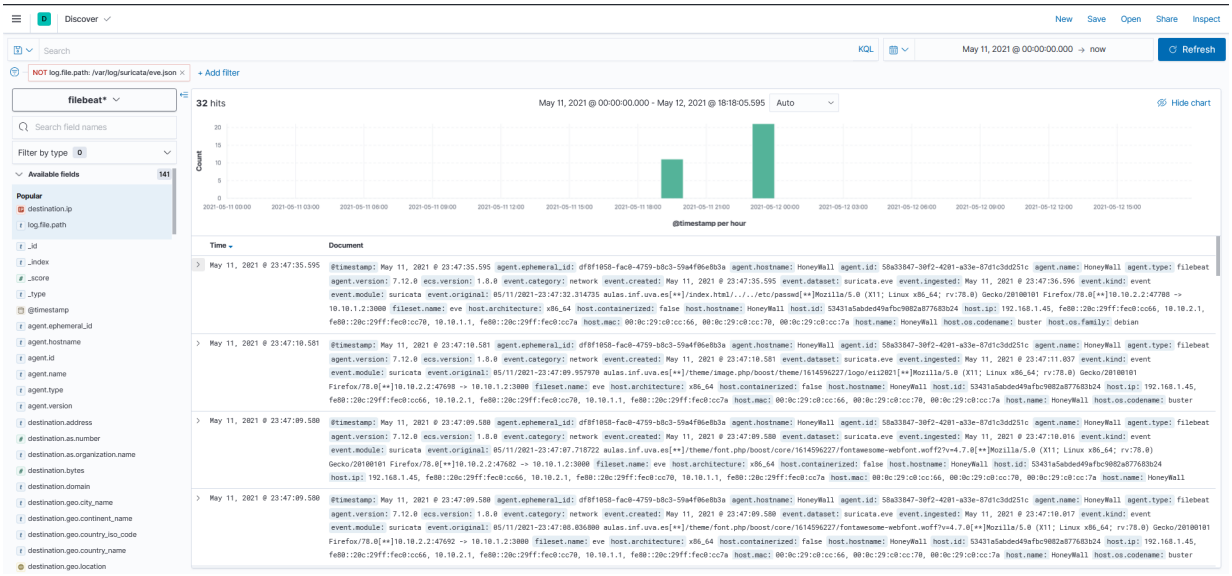

Figura 39: Eventos Suricata visualizados en el Kibana

Con esta configuración, podremos ver los eventos que genera el Suricata y filtrarlos, por ejemplo, por la IP de origen o destino, la petición HTTP, el User-Agent, el sistema operativo y versión del origen, etc...

Filebeat tiene dashboards por defecto que se pueden cargar en Kibana si tenemos activada la opción **setup.kibana** en el fichero de configuración filebeat.yml. Para cargar los dashboard ejecutamos el siguiente comando y ya podemos visualizarlos desde Kibana:

```
$ filebeat setup ——dashboards
```

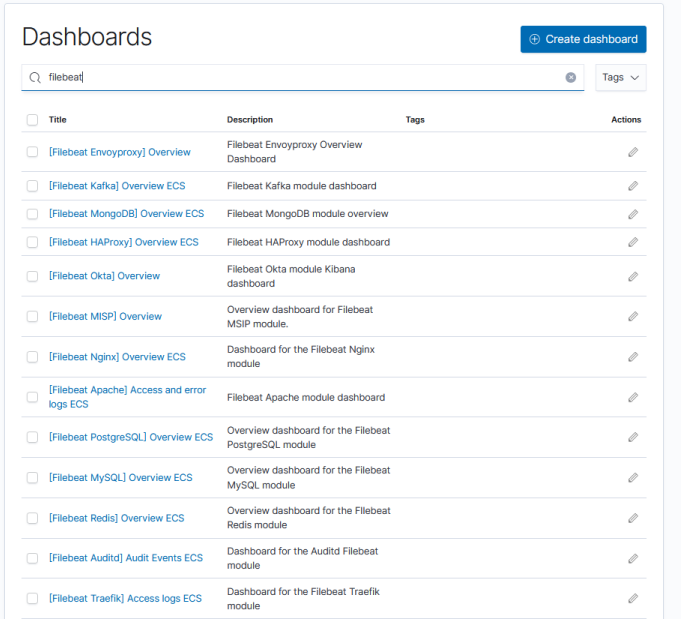

Figura 40: Dashboards predeterminados de Filebeat para Kibana

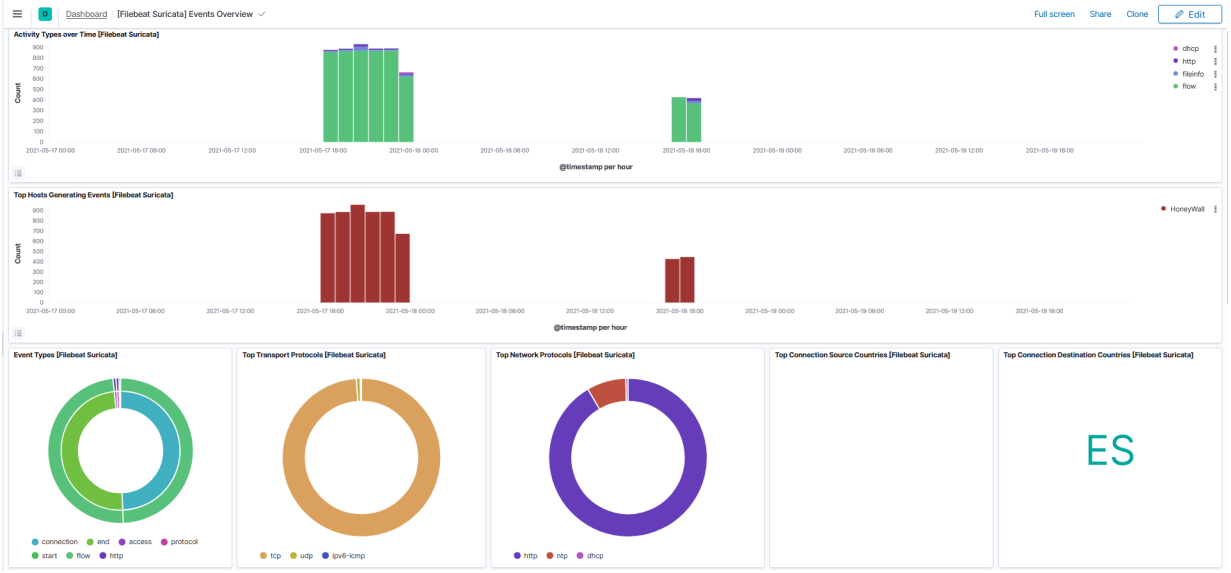

Figura 41: Dashboard Suricata Events

Los dashboards predeterminados son sencillos y cómodos, pero si necesitamos crear uno más personalizado para explorar o analizar mejor los eventos, desde el editor de dashboard se pueden crear de manera muy sencilla.

5.8. Creación de Dashboard Kibana para Logs de Suricata

Como se ha explicado antes, Kibana puede formar gráficos o tablas a partir de estos datos para perfeccionar la búsqueda. El conjunto de estos gráficos se llama ”dashboard”. Desde Kibana se pueden hacer dashboard tanto sencillos o complejos. En este proyecto, vamos a realizar un dashboard con los campos más útiles del Suricata. [**key24**]

Accedemos a la página Dashboard → Create dashboard y nos encontraremos un editor donde podemos ir añadiendo paneles, gráficas y tablas pulsando ’Create panel’ → ’Lens’.

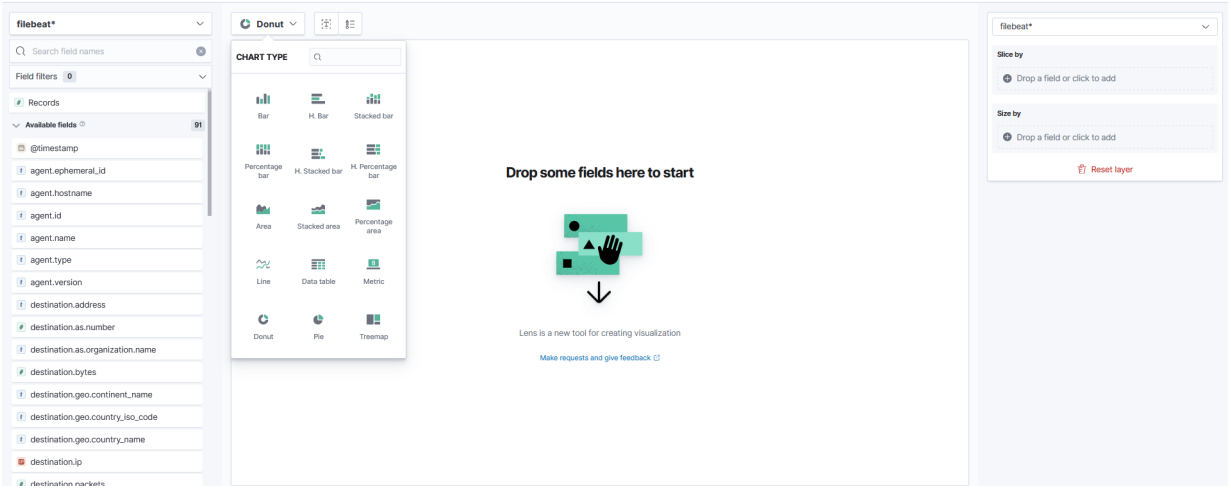

Figura 42: Creador de panel

Una vez en el editor del panel, se seleccionará un campo y el gráfico con el que vamos a trabajar, por

ejemplo, un gráfico de barras verticales con el campo @timestamp. Guardamos este panel en la esquina de arriba a la derecha y volvemos al editor del dashboard:

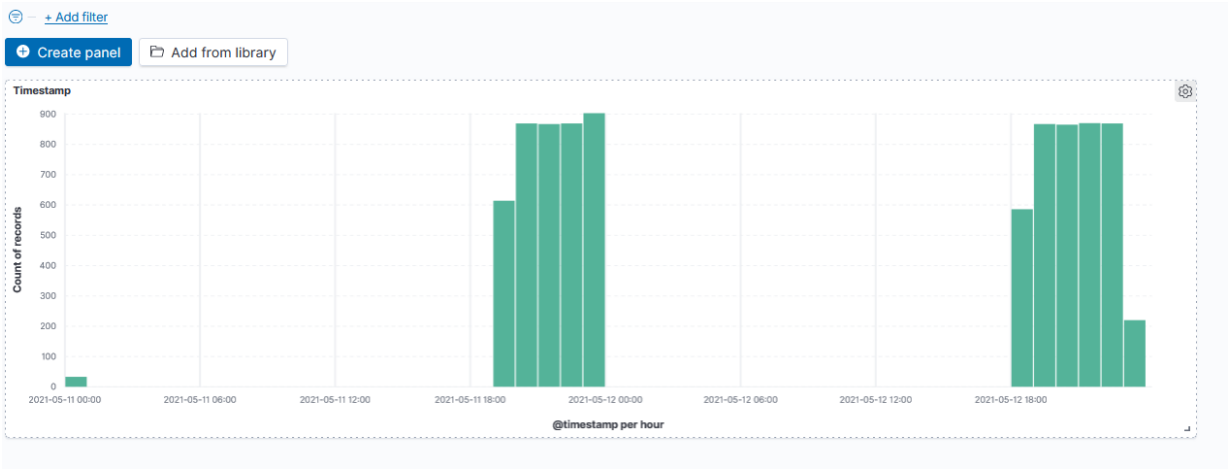

Figura 43: Gráfico de barras verticales de campo @timestamp

Ahora podemos cambiar el nombre al panel, maximizarlo, posicionarlo donde nosotros queramos dentro del dashboard. Para completar más el dashboard de Suricata añadimos unos gráficos sobre los campos: suricata.eve.src_ip, suricata.eve.dest_ip, http.request.method, http.response.status_code, file.path y un panel con los eventos tal cual está en la pestaña Discover.

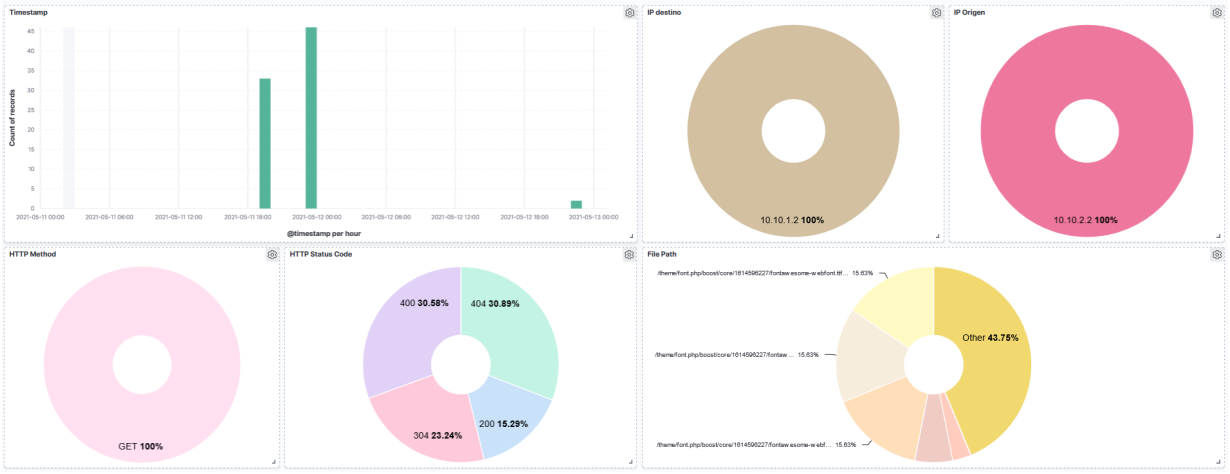

Figura 44: Dashboard Suricata parte 1

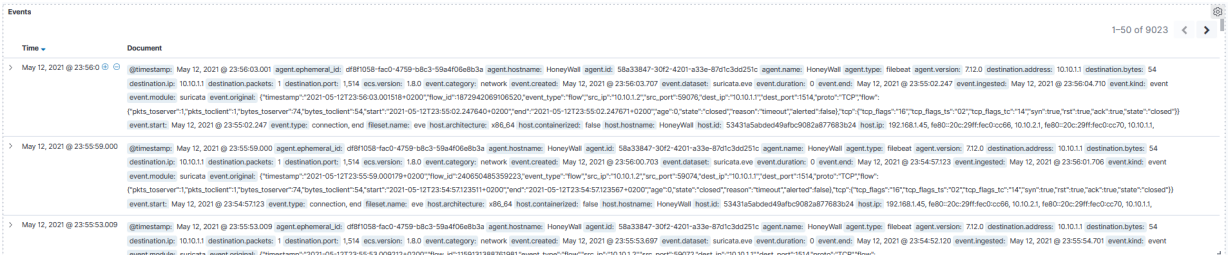

Figura 45: Dashboard Suricata parte 2

Con este dashboard podremos filtrar y precisar la búsqueda de los datos.

5.9. Recolección de diferentes logs con Filebeat

Ahora mismo nuestro recolector solo recoge los eventos de Suricata. Es importante en sistemas como Honeynet y SIEM recoger todos los eventos que puedan ocurrir ya que, si no recolectamos algún tipo de evento, los atacantes pueden usar ese sistema que no estamos controlando como vector de ataque y no nos hayamos dado cuenta de ello.

Los eventos que nos faltan por visualizar son:

- Eventos del sistema de HoneyWall.
- Eventos del sistema de HoneyDocker.
- Tráfico de red para complementar eventos que no capture Suricata.
- Eventos de los contenedores Docker: aulas.inf.uva.es y jair.lab.inf.uva.es

Es necesario tener instalado y configurado Filebeat en el HoneyDocker para que éste pueda enviar sus logs al HoneyWall. Se instala el paquete Filebeat y se configura en el fichero principal la salida hacia el Elasticsearch del HoneyWall como **hosts: ["10.10.1.1:9200"]**. Por último, reiniciamos el servicio Filebeat.

5.9.1. Eventos del sistema

Para los eventos del sistema HoneyWall y HoneyDocker utilizaremos el módulo de Filebeat **System** que será el encargado de recolectar y parsear los logs creados por el servicio de registros del sistema Unix/Linux. Cuando se ejecuta el módulo, realiza algunas tareas por debajo como establecer las rutas predeterminadas a los archivos de registros como auth.log, syslog, etcéra, y asegurar de que cada evento de registro de varias líneas se envíe como un solo evento. Utiliza el nodo de ingesta para analizar y procesar las líneas de registro para luego formar una estructura adecuada y visualizarlo en Kibana. Para activar el módulo usamos el siguiente comando:

```
$ filebeat modules enable system
```

Por defecto, el módulo System envía todos los logs que se encuentren en la ruta /var/log/ y tengan la extensión *.log. Por lo tanto, algunos ficheros que registra son [**key25**]:

- **auth.log**: Proporciona un registro de todas las actividades que implican un proceso de autenticación. Por ejemplo registro de usuarios logeados con su día, hora, usuario y órdenes que se han ejecutado con el comando sudo.
- **syslog**: Contiene la totalidad de logs capturados por rsyslogd. Este log es muy difícil de consultar y filtrar por su extenso tamaño, por eso se distribuye en otros ficheros siguiendo la configuración del fichero /etc/rsyslog.conf.
- **messages**: Contiene mensajes informativos y no críticos de la actividad del sistema operativo. Errores que se registran en el arranque del sistema no relacionados con el Kernel.
- **kern.log**: Proporciona información detallada de mensajes del kernel como mensajes de error y advertencias al compilar el kernel o detectar problemas del hardware.

Para no visualizar demasiados eventos o ruido en el Kibana, desde el fichero de configuración del módulo System (/etc/filebeat/modules.d/system.yml) solo seleccionaremos los ficheros imprescindibles para este proyecto: **/var/log/auth.log, /var/log/messages, /var/log/kern.log, /var/log/faillog y /var/log/lastlog en HoneyWall y en HoneyDocker**

Las reglas de IPtables que hayamos definido para que nos generen un evento de log, se registrarán tanto en los logs **messages** como **kern.log** de la siguiente forma:

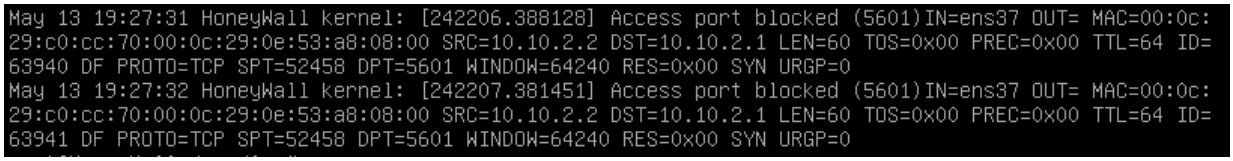

Figura 46: Logs IPtables visualizados en kern.log

En el Kibana podemos visualizar estos eventos estructurados en los diferentes campos para poderlos filtrar más fácilmente según su fecha, IP origen, de qué fichero procede ese evento, etcétera.

Figura 47: Ejemplo de bloqueo de peticiones por IPtables en Kibana

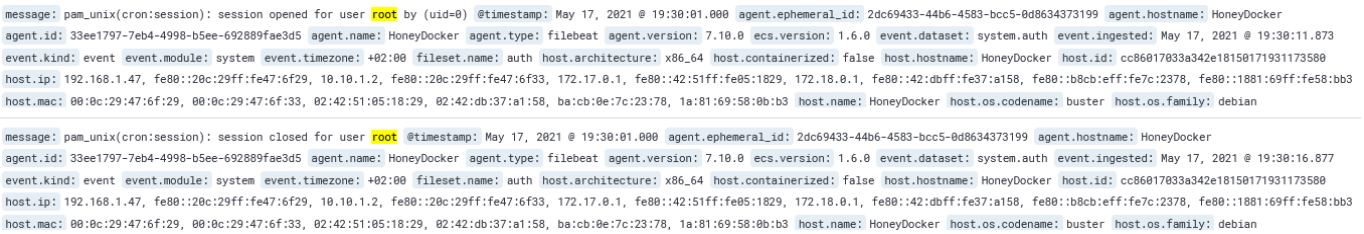

Figura 48: Ejemplo de sesión iniciada y cerrada por el usuario root en Kibana

Figura 49: Ejemplo de inicio sesión fallido en Kibana

5.9.2. Tráfico de red Packetbeat

Una buena práctica para los sistemas SIEM, o Honeynet como es en este caso, es recolectar el tráfico de red desde más de una fuente. En nuestro ejemplo, si Suricata ha devuelto un error o directamente no ha capturado cierto tráfico, estamos expuestos a que haya habido una fuga de información o que un equipo haya quedado comprometido sin percatarnos. Por lo tanto, es necesario instalar otra fuente de monitorización de tráfico de red.

El software complementario que instalaremos es **Packetbeat** [**key26**], que es un analizador de paquetes de red ligero que envía datos desde sus hosts a Logstash o Elasticsearch. Packetbeat es un tipo de Beat como Filebeat perteneciente a la empresa Elastic Stack. El proceso que sigue Packetbeat es capturar el tráfico de red, decodificar los protocolos de red, correlacionar las peticiones con sus respuestas, extraer los campos como el tiempo de respuesta, estado, etc.. y agrupar los JSON para enviárselos a Logstash o Elasticsearch.

Con Packetbeat monitorizaremos la interfaz externa del HoneyWall (10.10.2.1). Como Filebeat y ELK requieren repositorios ya incluidos en Packetbeat, solo faltaría instalarlo y ponerlo en ejecución:

```
$ apt install packetbeat
$ systemctl enable packetbeat
$ systemctl start packetbeat
```

Configuramos el fichero predeterminado que se encuentra en /etc/packetbeat/packetbeat.yml. Definimos la interfaz que va a monitorizar con **packetbeat.interfaces.device: ens37**, añadimos el puerto 22 en la sección TLS ya que es tráfico cifrado, configuramos que la salida sea hacia Elasticsearch y, por último, descomentamos la sección **setup.kibana** para poder cargar los dashboards predeterminados de Packetbeat.

Reiniciamos el servicio Packetbeat y cargamos los dashboards de Packetbeat:

```
$ packetbeat setup ——dashboards
```

Igual que con Filebeat, necesitamos crear un index pattern para que Kibana pueda procesar los datos del Packetbeat. Por lo tanto, lo creamos utilizando **"packetbeat*"** como expresión regular. Los eventos de Packetbeat los veremos desde **"packetbeat*"** para que no se junten con los del Suricata, que están en el **"filebeat*"**, y no se refleje el tráfico dos veces a la hora de explorar.

Solo se usará Packetbeat cuando en Suricata se hayan perdido eventos o no se llegue a apreciar suficientemente bien el comportamiento del ataque. Algunos ejemplos de eventos de Packetbeat:

Figura 50: Ejemplo de petición HTTP GET contra aulas.inf.uva.es en Kibana

Figura 51: Ejemplo de Ping ICMP contra jair.lab.inf.uva.es

5.9.3. Eventos de los contenedores Docker

Para recoger los logs de los contenedores Docker [**key27**] hay dos maneras: la compleja, que sería r enviar desde el propio contenedor Docke los logs al Elasticsearch del HoneyWall; o la sencilla, que sería enviar el log que genera cada contenedor Docker en el HoneyDocker al Elasticsearch del HoneyWall. Para no complicar más el proyecto, se decidió utilizar la forma sencilla y enviar los logs que se encuentran en **/var/lib/docker/containers/<container_id>/<container_id>-json.log**. En este caso, serían dos logs: uno del Docker de aulas.inf.uva.es (httpd) y otro del Docker de jair.lab.inf.uva.es (cowrie/cowrie).

Al tratarse de logs que no corresponden con ningún módulo de Filebeat, tenemos que configurarlos desde el fichero de configuración filebeat.yml como log input. En la sección **filebeat.inputs:** → **paths:** añadimos que nos lea todos los ficheros que sigan la condición **'/var/lib/docker/containers/*/*.log'**.

Reiniciamos el servicio Filebeat y, como configuramos antes Filebeat para que nos enviará los logs del HoneyDocker al HoneyWall, podremos ver los eventos de los contenedores Docker desde Kibana. El log aparecerá sin pasar ni por el módulo de Filebeat ni por Logstash.

t host.name	HoneyDocker
t host.os.codename	buster
t host.os.family	debian
t host.os.kernel	4.19.0-14-amd64
t host.os.name	Debian GNU/Linux
	Multi fields
	host.os.name.text: Debian GNU/Linux
t host.os.platform	debian
t host.os.version	10 (buster)
t input.type	log
t json.log	10.10.2.2 - - [19/May/2021:17:47:18 +0000] "GET /theme/font.php/boost/core/1614596227/fontawesome-webfont.woff2?v=4.7.0 HTTP/1.1" 404 196
t json.stream	stdout
t json.time	2021-05-19T17:47:18.487536577Z
t log.file.path	/var/lib/docker/containers/49577c1066b14577cb7d5476d07abba464a0e2e6950d692e5a5948d755767d3c/49577c1066b14577cb7d5476d07abba464a0e2e6950d692e5a5948d755767d3c-json.log
# log.offset	45,381

Figura 52: Ejemplo de petición HTTP capturada en el Docker httpd de aulas.inf.uva.es

Una conexión SSH está cifrada por lo que desde el Suricata y Packetbeat no podemos observar nada interesante sobre las acciones que se están realizando. Para ver los comandos que se están ejecutando, utilizamos el log del Docker. Algunos eventos del log Docker SSH Cowrie visualizados en Kibana:

t host.name	HoneyDocker
t host.os.codename	buster
t host.os.family	debian
t host.os.kernel	4.19.0-14-amd64
t host.os.name	Debian GNU/Linux
	Multi fields
	host.os.name.text: Debian GNU/Linux
t host.os.platform	debian
t host.os.version	10 (buster)
t input.type	log
t json.log	2021-05-23T21:27:41+0000 [cowrie.ssh.factory.CowrieSSHFactory] New connection: 10.10.2.2:51516 (172.17.0.2:2222) [session: c8fc0b671c08]
t json.stream	stdout
t json.time	2021-05-23T21:27:41.660520567Z
t log.file.path	/var/lib/docker/containers/323dce1984514365f52dc14aa572c45f293ef2707fa55d19632ee10ff531b49f/323dce1984514365f62dc14aa572c45f293ef2707fa55d19632ee10ff531b49f-json.log
# log.offset	131,196

Figura 53: Ejemplo de conexión establecida por SSH desde Kali Linux visualizado en Kibana

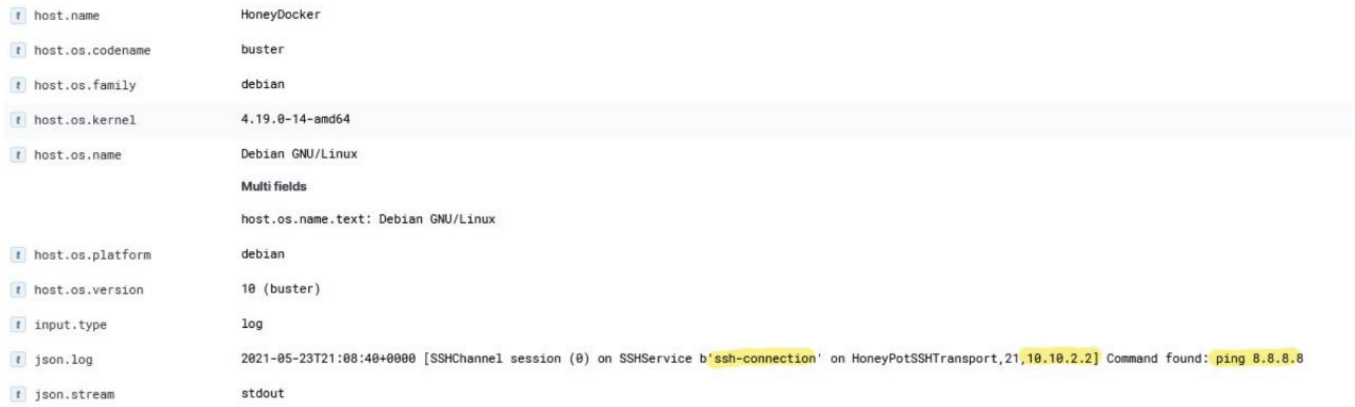

Figura 54: Ejemplo de captura de comando 'ping 8.8.8.8' desde Kali Linux visualizado en Kibana

6. Batería de pruebas

Para comprobar la eficacia de la Honeynet simularemos varios vectores de ataque desde el Kali Linux situado en la red WAN (10.10.2.2). Se usarán los siguientes vectores de ataque:

- **Escaneo de puertos con NMAP**
- **Escaneo de vulnerabilidades con OpenVAS**
- **Fuerza bruta SSH contra Cowrie**
- **Comandos Linux dentro del Docker Cowrie**

6.1. Ataque: Escaneo de puertos con NMAP

Uno de los primeros vectores de ataque que emplean los ciberdelicuentes es escanear dispositivos expuestos a Internet en busca de puertos abiertos que tengan alojado algún servicio vulnerable. Desde el Kali Linux ejecutamos el siguiente comando para realizar un escaneo con NMAP de todos los puertos con la opción -sV, para indicarnos la versión del software de cada puerto:

```
$ nmap -p- -sV 10.10.2.1
```

```
┌──(root💀kali)-[~]
└─# nmap -p- -sV 10.10.2.1
Starting Nmap 7.91 ( https://nmap.org ) at 2021-05-27 10:33 EDT
Nmap scan report for aulas.inf.uva.es (10.10.2.1)
Host is up (0.0014s latency).
Not shown: 65528 closed ports
PORT     STATE    SERVICE      VERSION
22/tcp   open     ssh          OpenSSH 6.0p1 Debian 4+deb7u2 (protocol 2.0)
53/tcp   open     domain       ISC BIND 9.11.5-P4-5.1+deb10u3 (Debian Linux)
80/tcp   open     http         Apache httpd 2.4.46 ((Unix))
2222/tcp filtered EtherNetIP-1
5601/tcp filtered esmagent
9200/tcp filtered wap-wsp
9300/tcp filtered vrace
MAC Address: 00:0C:29:C0:CC:70 (VMware)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 31.66 seconds
```

Figura 55: Resultado del escaneo con NMAP desde Kali Linux

En los resultados del escaneo de puertos, obtenemos siete puertos de los cuales tres están abiertos (SSH, DNS, HTTP) y cuatro están filtrados por el firewall, por lo que el atacante no puede conectarse a ellos ni ver que servicios están ejecutándose en estos. Una buena práctica de seguridad es cambiar los puertos por defecto de los servicios. Aunque el atacante no sepa qué servicio hay, sí que se puede hacer una idea por el número del puerto.

Visualizamos la captura del tráfico que generó este escaneo en el Kibana tanto por parte de Packetbeat como de Suricata.

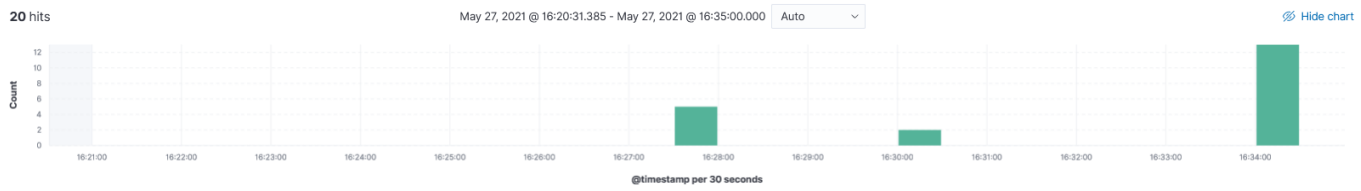

Figura 56: Tráfico del escaneo capturado por Suricata

Figura 57: Tráfico del escaneo capturado por Packetbeat

En la esquina superior izquierda aparecen los eventos registrados por cada componente. Suricata registró 20 eventos mientras que Packetbeat 98687 eventos. Esto se debe a que Suricata se encuentra escuchando el tráfico en la interfaz interna del HoneyWall mientras que el Packetbeat está escuchando en la interfaz externa conectada a la red WAN o Internet. Las únicas peticiones que capturó el Suricata son las que atraviesan el firewall a través de los puertos abiertos (DNS, HTTP y SSH). Por otro lado, Packetbeat capturó un total de una o más peticiones por puerto escaneado. Una máquina tiene un rango de puertos de 0 a 65535.

En este ejemplo observamos la importancia de elegir bien la interfaz que estará en modo monitorización, si queremos capturar todo el tráfico o simplemente el que atraviese el firewall.

6.2. Escaneo de vulnerabilidades con OpenVAS

OpenVAS [**key28**] es un escáner de vulnerabilidades de uso libre utilizado para la identificación y corrección de fallos de seguridad. Se trata de un framework que tiene como base servicios y herramientas para la evaluación de vulnerabilidades y que puede utilizarse de forma individual o como parte del conjunto de herramientas de seguridad incluidas en OSSIM (Open Source Security Information Management).

Para simplificar el TFG, se ha optado por no explicar la instalación y la configuración del OpenVas.

Primero añadimos el Target, que será la IP del HoneyWall (10.10.2.1), desde la pestaña Configuration → Targets. Después sobre la pestaña Scans → Tasks, si pulsamos 'New Task' y elegimos el target que hemos añadido antes, se realizará el escaneo de vulnerabilidades contra esa IP y veremos el progreso del escaneo.

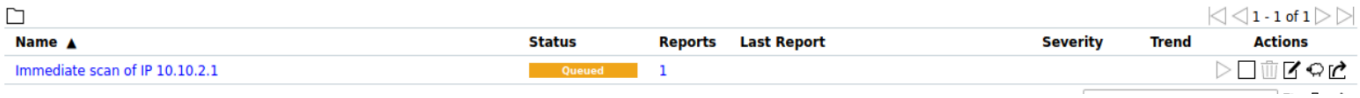

Figura 58: Escaneo a la IP 10.10.2.1 en cola

Figura 59: Escaneo a la IP 10.10.2.1 en progreso

Se observa el volumen de tráfico generado por el escaneo en el Kibana que en total son 65191 eventos

generados. A diferencia de un escaneo de puertos el escaneo de vulnerabilidades tarda más en ejecutarse, un total de 15 minutos.

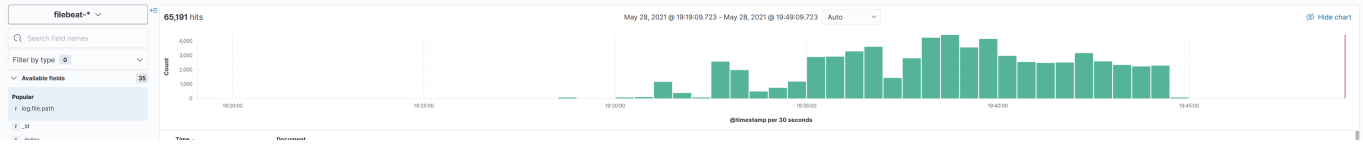

Figura 60: Reporte de OpenVAS

Si observamos el reporte obtenido en OpenVAS, vemos las vulnerabilidades detectadas con su severidad y el puerto del servicio vulnerable. Si queremos conocer el CVE correspondiente vamos a la pestaña 'CVEs' del reporte.

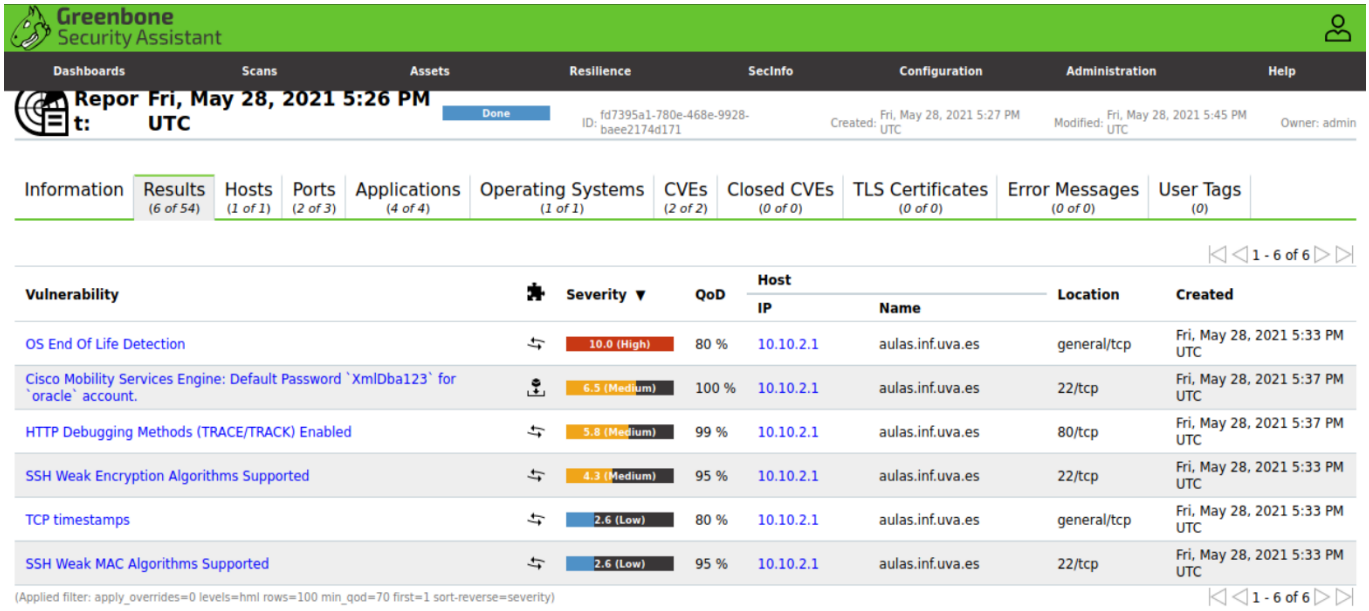

Figura 61: Reporte de OpenVAS

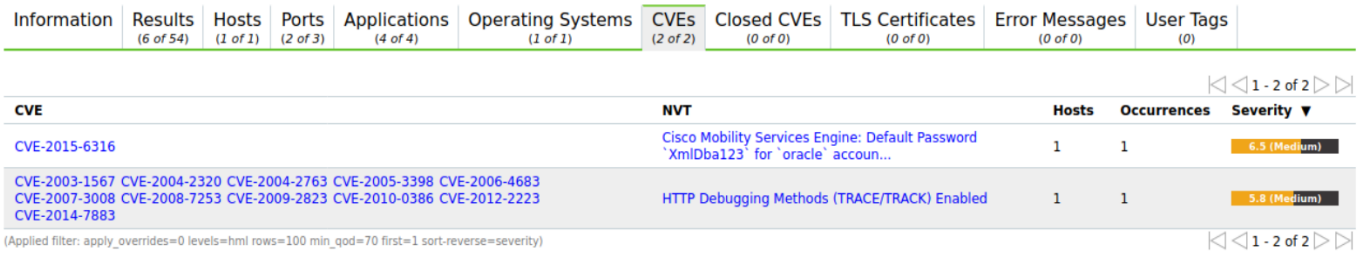

Figura 62: CVEs detectados durante el escaneo

Analizamos los CVEs detectados para ver si realmente son falsos positivos o una vulnerabilidad aplicable. La vulnerabilidad Cisco Mobility Services Engine Static Credential Vulnerability (CVE-2015-6316) podría permitir que un atacante remoto no autenticado inicie sesión via SSH con la cuenta de 'Oracle' predeterminada. Se debe a que la cuenta contiene la contraseña por defecto y estática (XmlDba123). Si intentamos explotar dicha vulnerabilidad en nuestro sistema, vemos que es vulnerable y se consigue iniciar sesión con esta cuenta.

No vamos a parchear esta vulnerabilidad porque el atacante puede pensar que es un despiste del administrador y ofrece algo más de realismo a la HoneyNet y, como el inicio de sesión es contra el sistema falso de Cowrie, esta controlado.

La vulnerabilidad HTTP Debugging Methods (TRACE/TRACK) Enabled (CVE-2003-1567) permite utilizar el método TRACK que devuelve el contenido de la petición original en el cuerpo de la respuesta. Esto facilita a atacantes remotos el robo de cookies y credenciales o evitar el mecanismo de protección HttpOnly usando TRACK para leer los contenidos de las cabeceras HTTP de los paquetes de respuesta. Una técnica similar al rastreo de sitios cruzados (XST) usando HTTP TRACE.

Dado que en nuestra página web no hay conectada ninguna base de datos ni ningún sistema de credenciales, podemos dejar sin parchear esta vulnerabilidad para dar más realismo.

Lo que hemos observado desde el OpenVAS es la información que habría obtenido el atacante. Por otro lado, desde el Kibana discover o dashboard podemos visualizar el escaneo completo, como por ejemplo las alertas que han saltado por parte del Suricata.

Top Alert Signatures [Filebeat Suricata]

Export

Alert Signature	Alert Category	Count
ET SCAN OpenVAS User-Agent Inbound	Attempted Information Leak	690
GPL EXPLOIT iissamples access	Web Application Attack	1
GPL WEB_SERVER iisadmin access	Web Application Attack	1
SURICATA HTTP Host header ambiguous	Generic Protocol Command Decode	1

Figura 63: Alertas detectadas por el Suricata desde Kibana

Como se puede ver, destaca la regla 'ET SCAN OpenVAS User-Agent Inbound' indicando que se esta realizando un escaneo desde fuera de la red con OpenVAS. También podemos observar las otras 2 alertas sobre accesos a contenidos como '/iissamples' y '/iisadmin'.

Otra información relevante son los códigos de error HTTP o el HTTP Status Code, que es la respuesta devuelta por el servicio web a las peticiones. Por ejemplo, el famoso error **404 (Not Found)** que es que no ha encontrado ese recurso solicitado, **501 (Not Implemented)** significa que el servidor no soporta esa funcionalidad, **301 (Moved Permanently)** que indica que el recurso ha sido movido a otra dirección permanentemente o **400 (Bad Request)** que el servidor no ha podido procesar la petición por algún error en la petición por parte del cliente.

HTTP Status Code	Count
404	36
400	1
404	37,033
400	360
301	105
403	35
501	26
404	16,750
403	245
400	137
405	35

Figura 64: Códigos de error HTTP Status

Este ha sido el conteo de los códigos de error. Si filtramos por el código **200 (OK)**, que significa petición procesada exitosamente, veremos algunas peticiones exitosas como a recursos '/rss', '/pluginfile.php/153/user/', '/libhtp::request_uri_not_seen' y más. Aquí es donde entra el analista de ciberseguridad para ver el impacto del escaneo, replicando estas peticiones y siguiendo los mismos pasos que el atacante.

Figura 65: Ejemplo de petición 200 OK desde Kibana

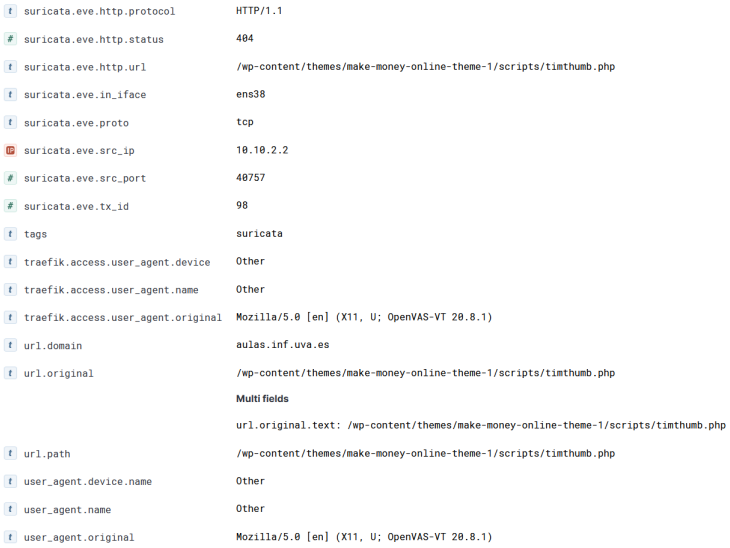

Figura 66: Ejemplo de petición 404 Not Found desde Kibana

6.3. Fuerza bruta SSH contra Cowrie

Se realizará un intento de fuerza bruta de SSH mediante hydra con un diccionario que contiene 900 posibles contraseñas, incluida la correcta. Las credenciales a intentar encontrar son "test/test", con el siguiente comando ejecutamos la fuerza bruta:

```
$ hydra -l test -P /usr/share/wordlists/dirb/small.txt ssh://jair.lab.
  inf.uva.es -t 4
```

Después de 10 minutos ejecutándose, el ataque finaliza encontrando la contraseña correcta del usuario test.

```
└─# hydra -l test -P /usr/share/wordlists/dirb/small.txt ssh://jair.lab.inf.uva.es -t 4  255 ✘
Hydra v9.1 (c) 2020 by van Hauser/THC & David Maciejak - Please do not use in military or secre
t service organizations, or for illegal purposes (this is non-binding, these *** ignore laws an
d ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2021-06-03 10:02:41
[DATA] max 4 tasks per 1 server, overall 4 tasks, 959 login tries (l:1/p:959), ~240 tries per t
ask
[DATA] attacking ssh://jair.lab.inf.uva.es:22/
[STATUS] 108.00 tries/min, 108 tries in 00:01h, 851 to do in 00:08h, 4 active
[STATUS] 108.00 tries/min, 324 tries in 00:03h, 635 to do in 00:06h, 4 active

[STATUS] 92.00 tries/min, 644 tries in 00:07h, 315 to do in 00:04h, 4 active
[22][ssh] host: jair.lab.inf.uva.es   login: test   password: test
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2021-06-03 10:11:34
```

Figura 67: Bruteforce con hydra con el usuario test

Con una media de 100 intentos por minutos con 4 tareas ejecutándose en paralelo, el volumen que genera en este ataque en el Kibana es de 4146 eventos. Como se puede observar, no es tan amplio como un escaneo de puertos. Esto se debe en mayor parte a que ha habido mas puertos escaneados que contraseñas intentadas pero también a que, cuando se realiza una ataque de fuerza bruta por SSH durante una sola conexión TCP establecida, se prueban hasta 4 contraseñas. Este proceso evita realizar varios handshake TCP que en un escaneo de puertos corriente no podría.

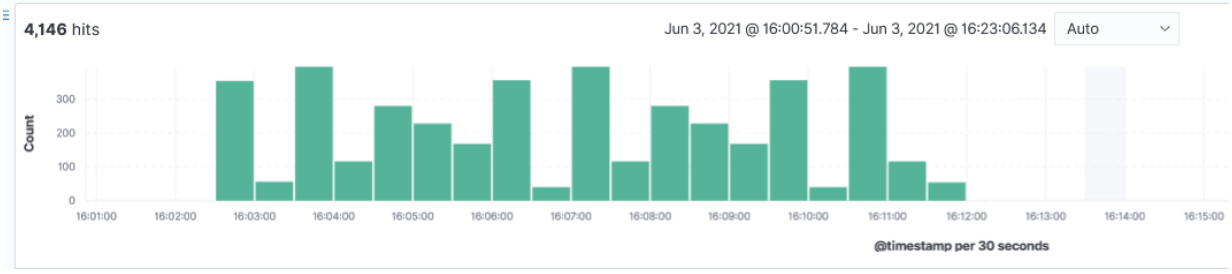

Figura 68: Intento de logeo fallido con credenciales test/table

Como se trata de una conexión SSH, el tráfico pasa cifrado, por lo que la única manera de saber el contenido del paquete es revisándolo con los logs del Docker Cowrie. Desde estos logs podemos ver cómo se han realizado intentos de inicio de sesión desde la IP 10.10.2.2 con las credenciales test/table o test/engine y han resultado fallidas.

debian

4.19.0-14-amd64

Debian GNU/Linux

Multi fields

host.os.name.text: Debian GNU/Linux

debian

10 (buster)

log

2021-06-03T14:11:29+0000 [SSHService b'ssh-userauth' on HoneyPotSSHTransport,381,10.10.2.2] login attempt [b'test'/b'table'] failed

stdout

2021-06-03T14:11:29.040773981Z

/var/lib/docker/containers/323dce1984514365f62dc14aa572c45f293ef2707fa55d19632ee10ff531b49f/323dce1984514365f62dc14aa572c45f293ef2707fa55d19632ee10ff531b49f-json.log

1,752,011

Figura 69: Intento de logeo fallido con credenciales test/table

t	host.os.kernel	4.19.0-14-amd64
t	host.os.name	Debian GNU/Linux
		Multi fields
		host.os.name.text: Debian GNU/Linux
t	host.os.platform	debian
t	host.os.version	10 (buster)
t	input.type	log
t	json.log	2021-06-03T14:05:36+0000 [SSHService b'ssh-userauth' on HoneyPotSSHTransport,355,10.10.2.2] login attempt [b'test'/b'engine'] failed
t	json.stream	stdout
t	json.time	2021-06-03T14:05:36.763764453Z
t	log.file.path	/var/lib/docker/containers/323dce1984514365f62dc14aa572c45f293ef2707fa55d19632ee10ff531b49f/323dce1984514365f62dc14aa572c45f293ef2707fa55d19632ee10ff531b49f-json.log
#	log.offset	1,340,752

Figura 70: Intento de logeo fallido con credenciales test/engine

Si filtramos en Kibana con la palabra 'succeeded' encontramos un evento de un intento de inicio de sesión exitoso con las credenciales test/test, por lo que podemos deducir que el ataque tiene las credenciales de ese usuario y, por lo tanto, acceso a la máquina Cowrie.

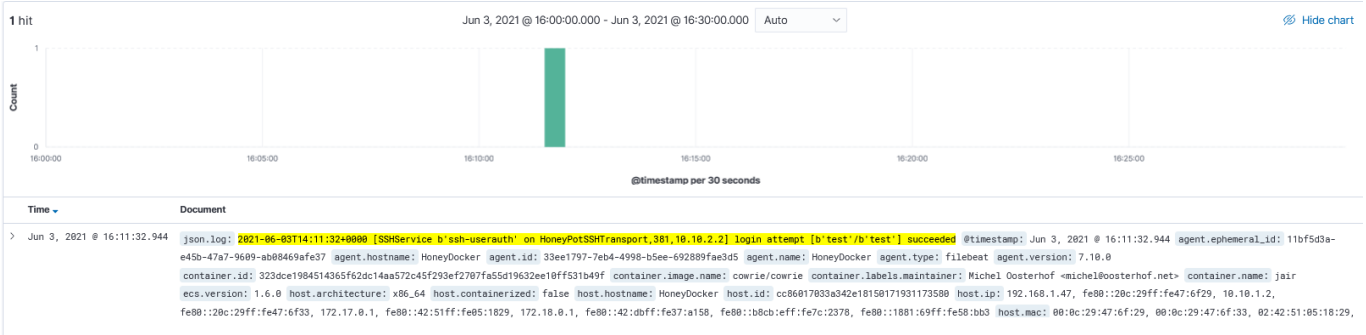

Figura 71: Intento de logeo exitoso con credenciales test/test

6.4. Comandos Linux dentro del Docker Cowrie

Como se ha mencionado anteriormente el tráfico SSH esta cifrado, por lo que vemos los comandos que se ejecutan desde los logs del Cowrie Docker. A continuación probamos a conectarnos con la cuenta de root al Docker Cowrie y ejecutamos un par de comandos:

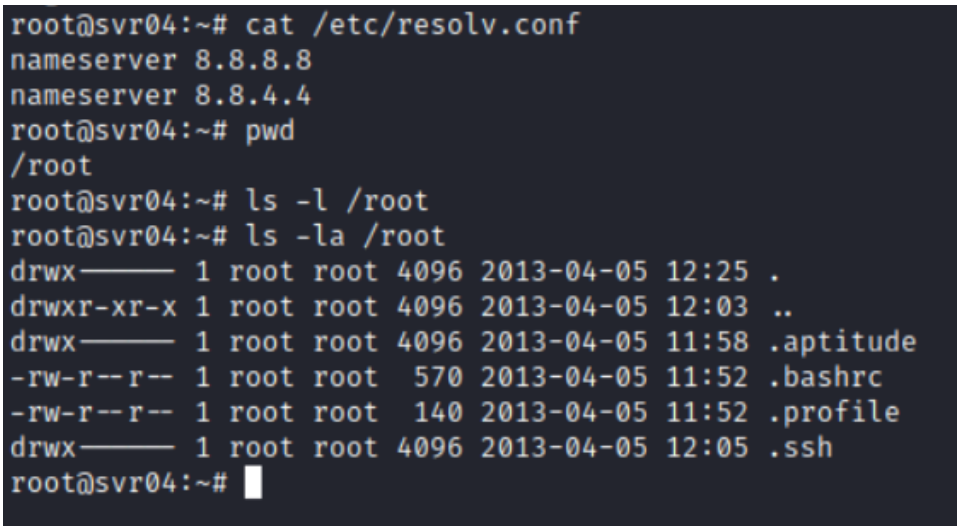

Figura 72: Ejecución de algunos comandos en una sesión SSH

Si observamos los resultados obtenidos desde Kibana veremos el comando que ha ejecutado, aunque no el resultado devuelto por el comando. Una manera conocer el resultado es replicando los comandos contra el mismo sistema.

Figura 73: Comando 'cat /etc/resolv.conf'

Figura 74: Comando 'pwd'

Figura 75: Comando 'ls -la /root'

7. Conclusión y posibles mejoras a futuro

7.1. Conclusiones

Durante el desarrollo del trabajo de fin de grado hemos aprendido sobre qué es una red Honeynet y su funcionamiento como su posterior utilización contra atacantes. Seguidamente, se enumeran los objetivos que se han conseguido durante el proyecto desarrollado:

- Se ha adquirido un conocimiento amplio sobre las redes Honeynet y su funcionamiento.
- Se ha implementado una Honeynet basada en servicios reales de la Universidad de Valladolid.
- Se ha desplegado un laboratorio para comprobar la funcionalidad de la Honeynet.
- Se ha desarrollado una Honeynet sobre un entorno seguro y virtualizado siguiendo un patrón de GEN III Virtual Híbrida.
- Se ha estudiado e implementado las diferentes tecnologías SIEM como pila ELK, Suricata e IPtables.
- Se ha desplegado un entorno escalable con la ayuda de la tecnología Docker.
- Se han simulado los diferentes vectores de ataque inicial que usaría un atacante.
- Se ha aprendido cómo trabaja un analista de ciberseguridad mediante la investigación de alertas.

Querría remarcar uno de los objetivos principales propuestos al inicio de este proyecto que era adquirir los conocimientos de ciberseguridad que se podrían implementar en un CERT o SIEM como es el uso y despliegue de la pila ELK, la implementación de reglas Suricata o el uso de contenedores Docker para virtualizar; y puedo manifestar que he aprendido mucho sobre estos temas. A pesar de que en la carrera existen asignaturas como Administración de Sistemas Operativos, Gestión de la Seguridad de la Información y Redes que me han ayudado a que me resulte más sencillo el desarrollo del proyecto, he tenido que trabajar duro e investigar cada una de las tecnologías y componentes que se han utilizado para obtener este resultado final.

Desde otra perspectiva, me gustaría indicar que, a pesar de que haya sido duro, he disfrutado desarrollando un proyecto relacionado con el campo de la ciberseguridad ya que es mi pasión y mi meta como trabajo.

7.2. Posibles mejoras a futuro

Durante la investigación y desarrollo de este trabajo de fin de grado, se han recopilado posibles mejoras a futuro sobre la Honeynet que, debido al poco tiempo para finalizar el proyecto y el desconocimiento, no se han realizado. A continuación se nombran las posibles mejoras:

- **Mejorar el nivel de interacción de los atacantes con los contenedores Docker**: Actualmente en el proyecto tenemos dos contenedores Docker de baja interacción, uno basado en una web y otro en un sistema Linux SSH. Una mejora es añadir más interacción a los contenedores existenes como añadir una base de datos, sistema de credenciales o configurar un sistema más real que el Docker Cowrie. Además de esto, agregar más contenedores como un servidor de correo, FTP o un Windows AD.
- **Separar la funcionalidad del HoneyWall**: Otra mejora es separar la funcionalidad del HoneyWall. Por una parte crear otro sistema Host que sea el router con sus IPtables, port forwarding y que separe las redes; y por otra, que otro sistema se encargue de monitorizar toda la red dentro de la propia LAN. Con esto ganamos menos carga sobre un equipo y más seguridad ya que, en caso de que se infecté el equipo HoneyWall no podrá acceder a los logs que están almacenados en él. Además se podría configurar una interfaz bridge en el 'nuevo' firewall.

- **Mejora del parseo en los logs**: En este proyecto, no se ha dado mucha importancia a cómo llegaban estructurados los logs al Elasticsearch. Es por esto que una mejora podría ser mejorar el parseo mediante Logstash y grok.
- **Añadir más medidas preventivas**: Actualmente en nuestro proyecto se han desplegado 2 tecnologías de seguridad preventiva: IPtables y IPS Suricata. Únicamente estamos utilizando la parte de prevención en IPtables para que bloquee los puertos descritos en iptablesUp.sh. Una mejora sería que para ataques conocidos, que no nos interese registrar o que generen mucha carga de tráfico a la Honeynet, sean bloqueados por el IPS Suricata.
- **Implementación del proyecto en un entorno real**: Una de las principales idea del proyecto es poder implementar esta Honeynet en un entorno real y, más en concreto, en la red de la Universidad de Valladolid.

- Anexos -

8. ANEXO

8.1. Configuración de red

```
source /etc/network/interfaces.d/*

#The loopback network interface
auto lo
iface lo inet loopback

#Interface para acceder a Internet
auto ens33
iface ens33 inet static
    address 192.168.1.45
    netmask 255.255.255.0
    network 192.168.1.0
    broadcast 192.168.1.255
    gateway 192.168.1.1

#Interface WAN con Kali
auto ens37
iface ens37 inet static
    address 10.10.2.1
    netmask 255.255.255.0
    network 10.10.2.0
    broadcast 10.10.2.255

#Interface LAN con HoneyDocker
auto ens38
iface ens38 inet static
    address 10.10.1.1
    netmask 255.255.255.0
    network 10.10.1.0
    broadcast 10.10.1.255
```

Listing 1: /etc/network/interfaces de HoneyWall

```
source /etc/network/interfaces.d/*

#The loopback network interface
auto lo
iface lo inet loopback

#Interface LAN con HoneyDocker
auto ens34
iface ens34 inet static
    address 10.10.1.2
    netmask 255.255.255.0
```

```
12      network 10.10.1.0
13      broadcast 10.10.1.255
14      gateway 10.10.1.1
```

Listing 2: /etc/network/interfaces de HoneyDocker

```
1  source /etc/network/interfaces.d/*
2
3  #The loopback network interface
4  auto lo
5  iface lo inet loopback
6
7  #Interface para acceder a Internet
8  auto eth1
9  iface eth1 inet static
10     address 192.168.1.55
11     netmask 255.255.255.0
12     network 192.168.1.0
13     broadcast 192.168.1.255
14     gateway 192.168.1.1
15
16 #Interface WAN con HoneyWall
17 auto eth0
18 iface eth0 inet static
19     address 10.10.2.2
20     netmask 255.255.255.0
21     network 10.10.2.0
22     broadcast 10.10.2.255
```

Listing 3: /etc/network/interfaces de Kali Linux

8.2. Configuración DNS del HoneyWall

```
1  options {
2      directory "/var/cache/bind";
3
4      forwarders {
5          8.8.8.8;
6          8.8.4.4;
7      };
8
9      dnssec-validation auto;
10     listen-on-v6 { any };
11 };
```

Listing 4: /etc/bind/named.conf.options

```
zone "inf.uva.es" {
    type master;
    file "/etc/bind/zones/db.inf.uva.es";
};

zone "1.10.10.in-addr.arpa" {
    type master;
    file "/etc/bind/zones/db.10.10.1";
};
```

Listing 5: /etc/bind/named.conf.local

```
$TTL    604800
@       IN      SOA     HoneyWall.inf.uva.es.   admin.inf.uva.es. (
                              3         ; Serial
                         604800         ; Refresh
                          86400         ; Retry
                        2419200         ; Expire
                         604800 )       ; Negative Cache TTL
;

        IN      NS      Honeywall.inf.uva.es.

HoneyWall.inf.uva.es.   IN      A       10.10.1.1
aulas.inf.uva.es.       IN      A       10.10.1.2
jair.lab.inf.uva.es.    IN      A       10.10.1.2
```

Listing 6: /etc/bind/zones/db.inf.uva.es

```
$TTL    604800
@       IN      SOA     HoneyWall.inf.uva.es.   admin.inf.uva.es. (
                              3         ; Serial
                         604800         ; Refresh
                          86400         ; Retry
                        2419200         ; Expire
                         604800 )       ; Negative Cache TTL
;

        IN      NS      Honeywall.inf.uva.es.

1       IN      PTR       HoneyWall.inf.uva.es.
2       IN      PTR       aulas.inf.uva.es.
2       IN      PTR       jair.lab.inf.uva.es.
```

Listing 7: /etc/bind/zones/db.10.10.1

8.3. Configuración IPtables

```
#!/bin/bash

#PERMITIR EL ENRUTAMIENTO DEL HONEYWALL
echo "1" > /proc/sys/net/ipv4/ip_forward

#ESTABLECER POLITICA POR DEFECTO TODO ACEPTAR
iptables -P INPUT ACCEPT
iptables -P OUTPUT ACCEPT
iptables -P FORWARD ACCEPT
iptables -t nat -P PREROUTING ACCEPT
iptables -t nat -P POSTROUTING ACCEPT

#INTRODUCIENDO SOURCE NAT (DINAMICO) CON IPTABLES
iptables -t nat -A POSTROUTING -s 10.10.1.0/24 -o ens37 -j MASQUERADE
iptables -t nat -A POSTROUTING -s 10.10.1.0/24 -o ens33 -j MASQUERADE

#INTRODUCIENDO PORT FORWARDING A WEB AULAS.INF.UVA.ES (80)
iptables -A FORWARD -m state -p tcp -d 10.10.1.2 --dport 3000 --state NEW
    ,ESTABLISHED,RELATED -j ACCEPT
iptables -t nat -A PREROUTING -p tcp --dport 80 -j DNAT --to-destination
    10.10.1.2:3000

#INTRODUCIENDO PORT FORWARDING A JAIR.LAB.INF.UVA.ES (22)
iptables -A FORWARD -m state -p tcp -d 10.10.1.2 --dport 3001 --state NEW
    ,ESTABLISHED,RELATED -j ACCEPT
iptables -t nat -I PREROUTING -p tcp --dport 22 -j DNAT --to
    10.10.1.2:3001

#BLOQUEAR PUERTOS 2222, 5601 (KIBANA), 9200 (ELASTICSEARCH), 9300
iptables -A INPUT -i ens37 -p tcp --destination-port 2222 -j LOG --log-
    prefix 'Access Port Blocked (2222)'
iptables -A INPUT -i ens37 -p tcp --destination-port 2222 -j DROP

iptables -A INPUT -i ens37 -p tcp --destination-port 5601 -j LOG --log-
    prefix 'Access Port Blocked (5601)'
iptables -A INPUT -i ens37 -p tcp --destination-port 5601 -j DROP

iptables -A INPUT -i ens37 -p tcp --destination-port 9200 -j LOG --log-
    prefix 'Access Port Blocked (9200)'
iptables -A INPUT -i ens37 -p tcp --destination-port 9200 -j DROP

iptables -A INPUT -i ens37 -p tcp --destination-port 9300 -j LOG --log-
    prefix 'Access Port Blocked (9300)'
iptables -A INPUT -i ens37 -p tcp --destination-port 9300 -j DROP
```

Listing 8: IPtables script (iptablesUp.sh)

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

```
2
3 #DENEGAR EL ENRUTAMIENTO DEL HONEYWALL
4 echo "0" > /proc/sys/net/ipv4/ip_forward
5
6 #REGLAS PARA BORRAR LAS IPTABLES
7 iptables -F
8 iptables -X
9 iptables -Z
10 iptables -t nat -F
```

Listing 9: IPtables script (iptablesDown.sh)

8.4. Configuración Suricata

```
1 ##
2 ## Step 1: inform Suricata about your network
3 ##
4
5 vars:
6   # more specific is better for alert accuracy and performance
7   address-groups:
8     HOME_NET: "[10.10.1.0/24]"
9     #HOME_NET: "any"
10
11     EXTERNAL_NET: "!$HOME_NET"
12     #EXTERNAL_NET: "any"
13 ...
14 default-log-dir: /var/log/suricata/
15
16 # global stats configuration
17 stats:
18   enabled: no
19   interval: 8
20
21
22 # Configure the type of alert (and other) logging you would like
     .
23 outputs:
24   # a line based alerts log similar to Snort's fast.log
25   - fast:
26       enabled: yes
27       filename: fast.log
28       append: yes
29
30   # Extensible Event Format (nicknamed EVE) event log in JSON
      format
31   - eve-log:
32       enabled: yes
33       filetype: regular
```

```
...
        types:
          - alert:
              # enable dumping payload in Base64
              payload: yes
              # max size of payload buffer to output in eve-log
              payload-buffer-size: 4kb
              # enable dumping payload in printable (lossy) format
              payload-printable: yes
              # enable dumping of packet (without stream segments)
              packet: yes
              # enable dumping of http body in Base64
              http-body: yes
              # enable dumping of http body in printable format
              http-body-printable: yes
          - http:
              # enable this for extended logging information
              extended: yes
          - drop:
              alerts: yes       # log alerts that caused drops
...
 # a line based log of HTTP requests (no alerts)
  - http-log:
      enabled: yes
      filename: http.log
      append: yes
...
  outputs:
  - console:
      enabled: yes
      # type: json
  - file:
      enabled: yes
      level: info
      filename: /var/log/suricata/suricata.log
      # type: json
...
af-packet:
  - interface: ens38
pcap:
  - interface: ens38
...
default-rule-path: /etc/suricata/rules

rule-files:
 - botcc.rules
 - ciarmy.rules
 - compromised.rules
 - drop.rules
```

```
83   - dshield.rules
84   - emerging-attack_response.rules
85   - emerging-chat.rules
86   - emerging-current_events.rules
87   - emerging-dns.rules
88   - emerging-dos.rules
89   - emerging-exploit.rules
90   - emerging-ftp.rules
91   - emerging-imap.rules
92   - emerging-icmp_info.rules
93   - emerging-icmp.rules
94   - emerging-malware.rules
95   - emerging-misc.rules
96   - emerging-mobile_malware.rules
97   - emerging-netbios.rules
98   - emerging-p2p.rules
99   - emerging-policy.rules
100  - emerging-pop3.rules
101  - emerging-rpc.rules
102  - emerging-scan.rules
103  - emerging-smtp.rules
104  - emerging-snmp.rules
105  - emerging-sql.rules
106  - emerging-telnet.rules
107  - emerging-tftp.rules
108  - emerging-trojan.rules
109  - emerging-user_agents.rules
110  - emerging-voip.rules
111  - emerging-web_client.rules
112  - emerging-web_server.rules
113  - emerging-web_specific_apps.rules
114  - emerging-worm.rules
115  - tor.rules
```

Listing 10: suricata.yaml

8.5. Configuración Stack ELK

```
1 #--- --- --- --- --- ---   Network --- --- --- --- --- ---
2 ...
3 network.host : 0.0.0.0
4 http.port : 9200
5 ...
```

Listing 11: elasticsearch.yml

8.6. Configuración Filebeat

```
# ==================== Kibana ====================
...
setup.kibana:
    host: "0.0.0.0:5601"
...
# ==================== Elasticsearch Output ====================
...
output.elasticsearch:
    hosts: ["127.0.0.1:9200"]
...
# ==================== Logstash Output ====================
...
#output.logstash:
    #hosts: ["127.0.0.1:5044"]
...
```

Listing 12: HoneyWall filebeat.yml

```
# ==================== Filebeat inputs ============
filebeat.inputs:

- type: log
    enabled: true

    paths:
        - '/var/lib/docker/containers/*/*.log'
# ==================== Kibana ====================
...

setup.kibana:
    #host: "0.0.0.0:5601"
...
# ==================== Elasticsearch Output ====================
...
output.elasticsearch:
    hosts: ["10.10.1.1:9200"]
...
# ==================== Logstash Output ====================
...
#output.logstash:
    #hosts: ["127.0.0.1:5044"]
...
```

Listing 13: HoneyDocker filebeat.yml

```
# Module: suricata

-module: suricata
 #All logs
 eve:
```

```
6       enabled: true
7
8       # Set custom paths for the log files. If left empty,
9       # Filebeat will choose the paths depending on your OS.
10      var.paths: ["/var/log/suricata/eve.json", "/var/log/suricata
      /http.log","/var/log/suricata/fast.log"]
```

Listing 14: Módulo Filebeat suricata.yml

```
1  # Module: system
2
3  -module: system
4   # Syslog
5   syslog:
6       enabled: true
7
8       # Set custom paths for the log files. If left empty,
9       # Filebeat will choose the paths depending on your OS.
10      var.paths: ["/var/log/lastlog", "/var/log/kern.log", "/var/
      log/messages", "/var/log/faillog"]
11
12      # Authorization logs
13  auth:
14      enabled: true
15
16      # Set custom paths for the log files. If left empty,
17      # Filebeat will choose the paths depending on your OS.
18      var.paths: ["/var/log/auth.log"]
```

Listing 15: Módulo Filebeat system.yml

8.7. Configuración Packetbeat

```
1  # ==================== Network device ======================
2
3  packetbeat.interfaces.device: ens37
4
5  # ==================== Transaction protocols ==============
6
7  packetbeat.protocols:
8  ...
9  - type: tls
10
11   ports:
12     - 22 # SSH
13     - 443 # HTTPS
14     - 993 # IMAPS
15     - 995 # POP3S
16     - 5223 # XMPP over SSL
```

```
17      - 8443
18      - 8883 # Secure MQTT
19      - 9243 # Elasticsearch
20  ...
21  # ==================== Kibana =====================
22  ...
23
24  setup.kibana:
25      host: "localhost:5601"
26  ...
27  # ================== Elasticsearch Output ==================
28  ...
29  output.elasticsearch:
30      hosts: ["localhost:9200"]
31  ...
32  # ================== Logstash Output ==================
33  ...
34  #output.logstash:
35      #hosts: ["127.0.0.1:5044"]
36  ...
```

Listing 16: packetbeat.yml

Bibliografía

[1] Antonio Salvador. *Estadística de ciberataques 2020*. 2021. URL: https://www.elindependiente.com/espana/2021/01/01/2020-ano-record-en-ciberataques/.

[2] Adnan Rahić. *Logs en Docker*. 2020. URL: https://sematext.com/blog/docker-logs-location/.

[3] Antonio Ramos Varón y col. *Seguridad perimetral, monitorización y ataques en redes*. 2014.

[4] Brian Hogan. *How To Install and Use Docker on Debian 10*. 2019. URL: https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-debian-10.

[5] *Conjunto de reglas Suricata Emerging Threats*. 2021. URL: https://rules.emergingthreats.net/open/suricata/rules/.

[6] Carlos Álvarez Martín y Pablo González Pérez. *Hardening de servidores GNU/Linux*. 2017.

[7] Carlos Carretero Aguilar. *Despliegue de una honeynet en la red de la diputación de Cádiz para la investigación de ataques informáticos*. 2018.

[8] Darkcrizt. *Suricata 4.0 supervisa el tráfico de la red*. 2017. URL: https://ubunlog.com/suricata-4-0-supervisa-el-trafico-de-la-red/.

[9] Docker. *Docker Oficial*. 2013. URL: https://docs.docker.com/.

[10] Erik Czumadewski. *Desventajas de Docker*. 2019. URL: https://es.quora.com/Cu%C3%A1les-son-las-desventajas-de-usar-Docker.

[11] Elastic. *Packetbeat Documentation*. URL: https://www.elastic.co/es/beats/packetbeat.

[12] Eduardo Malo. *Los ciberataques son ya la segunda mayor preocupación entre los CEOs españoles*. 2020. URL: https://www.muycanal.com/2020/02/06/ciberataques-preocupacion-espana/.

[13] Emiliano. *IPtables tutorial básico*. 2016. URL: https://www.linuxito.com/seguridad/793-tutorial-basico-de-iptables-en-linux.

[14] *Funcionamiento Cowrie*. 2018. URL: https://hackertarget.com/cowrie-honeypot-ubuntu/.

[15] Federación de Enseñanza de CC.OO. de Andalucía. "HoneyNets, una desconocida en la seguridad informática". En: (2009).

[16] Gobierno de España. *Información sobre RedIRIS Corporativa*. 2020. URL: https://www.rediris.es/rediris/mm/RedIRIS_corporativa_2020.es.pdf.

[17] Hartek. *Suricata IDS jugando con las reglas*. 2018. URL: https://ubunlog.com/suricata-4-0-supervisa-el-trafico-de-la-red/.

[18] Hartek. *IDS IPS Suricata entendiendo y configurando*. 2018. URL: https://fwhibbit.es/suricata-ids-jugando-con-las-reglas.

[19] itamarst. *Cheat Sheets Docker*. URL: https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Docker_Security_Cheat_Sheet.md.

[20] *Instalación y configuración de Suricata*. 2021. URL: https://blog.elhacker.net/2021/03/suricata-ids-ips-instalacion-configuracion-reglas-.html.

[21] jasonish. *suricata-update GitHub*. URL: https://github.com/OISF/suricata-update.

[22] Joan Carles. *Logs en Linux*. 2019. URL: https://geekland.eu/logs-en-linux/.

[23] caffix. *Github Amass*. URL: https://github.com/OWASP/Amass.

[24] Lourdes Peñalver Herrero. *ELK para el análisis de logs - TFG - Escuela Politécnica de Valencia*. 2020. URL: https://riunet.upv.es/bitstream/handle/10251/156901/Simarro%20-%20ELK%20para%20el%20an%C3%A1lisis%20de%20logs.pdf?sequence=1%5C&isAllowed=y.

[25] Lorna Chepkoech. *Instalación ELK en Debian 10*. 2021. URL: https://techviewleo.com/install-elastic-stack-7-elk-on-debian/.

[26] micheloosterhof. *Cowrie Honeypot Github*. 2020. URL: https://github.com/cowrie/cowrie.

[27] nmap. *NMAP Página Oficial*. URL: https://nmap.org/.

[28] OpenVAS. *OpenVAS Documentación*. URL: https://www.openvas.org/.

[29] Red Hat. *¿Qué es Docker?* 2013. URL: https://www.redhat.com/es/topics/containers/what-is-docker.

[30] Sergio Losada. *¿Qué es ELK?* 2018. URL: https://openwebinars.net/blog/que-es-elk-elasticsearch-logstash-y-kibana/.

[31] Scayle. *Información sobre la Red CAYLE*. URL: https://www.scayle.es/redcayle/infraestructura/.

[32] *Tutorial Kibana: Creación de index pattern y dashboard*. 2019. URL: https://www.ionos.es/digitalguide/online-marketing/analisis-web/tutorial-de-kibana/.

[33] Victor García. *Monitorización de aplicaciones usando ELK Stack*. 2019. URL: https://enimbos.com/monitorizacion-de-aplicaciones-usando-elk-stack/.