Facultad de Ciencias

SISTEMA DE DETECCIÓN DE INTRUSIONES (IDS) Y ANOMALÍAS EN UN CPD - INTRUSION DETECTION SYSTEMS AND ANOMALIES IN DATACENTERS

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENERÍA INFORMÁTICA

Autor: Daniel Belén Del Río

Director: Álvaro López García

Co-Director: Aida Palacio Hoz

06 - 2021

Indice

List of Figures

List of Tables

Resumen

Los centros de procesamiento de datos son utilizados ampliamente por todo el mundo y almacenan información valiosa sobre los usuarios y la propia organización, pero debido a su complejidad resulta difícil y costoso realizar labores de mantenimiento y protección de la información. Los sistemas de detección de intrusiones ayudan a simplificar estas tareas automatizando las tareas de monitorización de un sistema, identificando eventos destacables como ataques que afecten al funcionamiento del sistema.

Este trabajo propone una implementación de un sistema de detección de intrusiones y anomalías que permita automatizar la monitorización en entornos de alto rendimiento. Se describe el estado actual de los centros de procesamiento de datos, los principales problemas de seguridad que presentan y se realiza un estudio entre las diferentes tecnologías de detección de intrusiones existentes: Snort, Suricata y Zeek.

Con el fin de automatizar el despliegue del sistema se hace uso de herramientas como contenedores Docker y se somete el sistema resultante a pruebas en entornos de alto rendimiento.

Palabras clave ciberseguridad, docker, centro de procesamiento de datos, sistema de detección de intrusiones, suricata

Abstract

Datacenters are widely used around the world and store valuable information about users and the organisation itself but due to their complexity is difficult to maintain and protect the data. Intrusion detection systems help to simplify these tasks by monitoring the system and assert notable events such as attacks that affect the functioning of the datacenter.

In this work we proposes an implementation of an intrusion detection in high-performance environments. We describe the current state of art of datacenters and it is main security flaws and a study is made of the different existing intrusion detection solutions: Snort, Suricata and Zeek.

In order to automate the deployment of the system, tools such as Docker containers are used and the resulting system is tested in high-performance environments.

Keywords cibersecurity, docker, datacenter, ids, intrusion detection system, suricata

1. Introducción

Con el auge y evolución de las tecnologías de información y la transición a la era digital, es cada vez mayor el número de organizaciones que manejan grandes volúmenes de datos haciendo uso de los sistemas de información. Los centros de procesamiento de datos (CPD), o *datacenters*, son instalaciones informáticas especializadas que gestionan los datos y aplicaciones de una o varias organizaciones que los administran [3].

En la actualidad los CPD más modernos ofrecen sus recursos como servicios, incluido los recursos computacionales, pudiendo encontrarse en diferentes instalaciones interconectadas a través de internet. Para poder prestar estos servicios a todos los usuarios de la organización se utilizan modelos de computación en la nube que abstraen la infraestructura y el mantenimiento de la misma del usuario.

Aunque la computación en la nube presenta grandes ventajas en escalabilidad y disponibilidad, en realidad es un sistema complejo susceptible a problemas de seguridad si no se tratan debidamente. Al existir tantos usuarios utilizando el sistema conjuntamente no hay control sobre las acciones que ejecutan cada uno. Sumado al hecho de que múltiples usuarios comparten la infraestructura subyacente, en caso de que haya un problema de seguridad grave y un ataque, ya sea provocado por un acceso externo o interno, este puede comprometer a todos los usuarios del sistema antes de que sea descubierto y subsanado por los administradores. Las consecuencias pueden ser especialmente desastrosas pues un ataque organizado que conlleve una o varias intrusiones con éxito supone una amenaza a la privacidad de los usuarios con substracción de información confidencial, extorsiones, fraudes, daños de reputación y en la confianza de la organización y sus usuarios. Además, si se ofrecen servicios críticos como por ejemplo los que se encuentran en hospitales, gobiernos, centros de educación o centrales de energía las consecuencias afectarán también a la sociedad. El principal riesgo de seguridad es la falta de control del entorno.

Las organizaciones no siempre están debidamente preparadas para hacer frente a las oleadas de ataques dirigidas contra sus infraestructuras. Ya sea por una implementación incorrecta de las políticas de seguridad, nuevos vectores de ataque que se han ido descubriendo en los últimos años debido a los fallos intrínsecos de los sistemas, así como la propia especialización y evolución del *cibercrimen*, se ha producido un incremento innegable de los ataques y de los datos expuestos en la última década [4], [5], [6]. En la figura 1.1, se muestra el número de filtraciones de datos sólo en Estados Unidos de los que se tiene constancia pública. Este aumento no ha sido constante pero incluso aunque en algunos años el número de ataques

ha disminuido relativamente en años consecutivos, el número de datos expuestos ese mismo año ha crecido (2009 y más recientemente 2018), lo que indica ataques más efectivos:

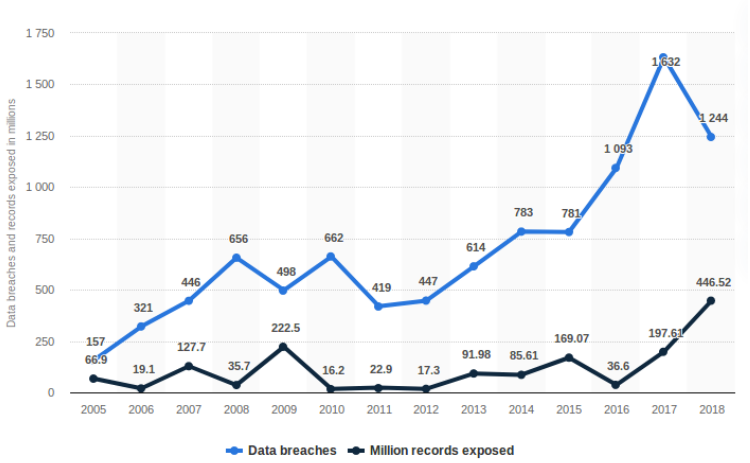


Figure 1.1: Ataques y fugas de datos anuales en Estados Unidos desde 2005 hasta 2018 [1].

En los entornos de los grandes centros de datos y sistemas en la nube, la monitorización ayuda a reducir el problema de la falta de control en el entorno. Monitorizar consiste es registrar todo lo que ocurre en un sistema en todo momento , con el fin de no sólo identificar las posibles amenazas presentes que puedan surgir en el, sino también poder analizarlas para determinar su alcance y prevenir incidencias similares en el futuro [7]. También es aplicable a incidencias no maliciosas que causen problemas de uso y si se aplica de la forma correcta mejorar el rendimiento y reducir costes de mantenimiento.

Los procesos de monitorización y análisis posterior de la información presentan grandes dificultades para llevarse a cabo en un entorno que genera grandes volúmenes de eventos al día, como lo es un CPD en la nube. Sería necesario por lo tanto un sistema que permite automatizar estos procesos y que sea capaz de filtrar los eventos que pueden suponer un riesgo para el sistema y sus usuarios.

Un sistema de detección de intrusiones (IDS) es en esencia un monitor de actividad, cuya labor consiste en recolectar y clasificar cualquier anomalía que pueda ocurrir en un sistema. Automatizan el proceso de monitorización permitiendo procesar grandes cantidades de eventos en un tiempo razonable, lo que permite reacciones más rápido ante eventos que necesiten atención urgente, minimizando o incluso deteniendo por completo sus consecuencias, ya sean intrusiones de seguridad que amenazan la integridad de los usuarios o problemas del funcionamiento del propio sistema.

1.1 Motivación, objetivos y ámbito

El principal motivo de este trabajo es comprender el estado actual de la seguridad y las vulnerabilidades en un CPD, así como de las herramientas que existen para combatirlos.También se busca simplificar las tareas de gestión y mantenimiento del CPD, mediante la automatización de tareas y una rápida detección de problemas, así como minimizar el impacto de ataques maliciosos en sus usuarios integrando herramientas que permitan responder rápidamente ante estas amenazas.

Este trabajo tiene dos objetivos principales, el primero es implementar un sistema de monitorización de red basado en un IDS y automatizar el proceso, que permita ser adaptable, escalable y fácilmente desplegable en entornos de alto rendimiento. El segundo es verificar que es capaz de detectar anomalías, ya sean maliciosas (intrusiones) o no, bajo la carga de funcionamiento esperada. Se llevará a cabo un estudio para determinar los componentes más adecuados del sistema de monitorización. Los componentes estarán basados en herramientas *opensource*. Para demostrar el segundo objetivo se evaluará la implementación mediante pruebas con ataques simulados y bajo situaciones de estrés.

1.2 Estructura de la memoria

El trabajo se compone de cinco partes principales:

La primera parte es la introducción al mismo, la enumeración de objetivos y estructura del trabajo.

En la segunda parte se hace una breve explicación de las tecnologías que forman parte en este proyecto, así como su estado actual. Se hace una introducción a los centros de procesamiento de datos, el procesamiento en la nube, su seguridad y sus limitaciones así como sus principales amenazas. Se describe el sistema de detección de intrusiones, los tipos de monitorización existentes analizando sus ventajas y limitaciones, los tipos de IDS y sus componentes. Finalmente se realiza una selección de sus componentes, escogiendo entre las principales soluciones IDS disponibles actualmente.

En la tercera parte se propone una arquitectura y un entorno de pruebas local, explicando paso a paso el proceso de implementación, automatización y configuración inicial de los componentes elegidos que permitan cumplir los objetivos. Las pruebas se llevarán a cabo en un entorno de nube privada en el centro de procesamiento de datos del IFCA [8]. Ofrece servicios basados en la nube y en computación distribuida orientados a labores y investigación científica [9].

En la cuarta parte se evalúan los componentes mediante pruebas en situaciones de estrés

midiendo la capacidad de detección y el uso de recursos, analizando y optimizando la configuración para mejorar los resultados.

En el última parte se expondrán las conclusiones del trabajo y se propondrán todas aquellas mejoras al proyecto que se crean que puedan mejorar su desempeño.

2. Estado del arte

En este apartado se exponen el contexto, herramientas y los conceptos necesarios para comprender este trabajo. Explicamos el contexto del centro de procesamiento de datos, el estado de la seguridad en ellos y las principales amenazas. Describimos los sistemas de detección de intrusiones, sus componentes, los tipos de soluciones que se plantean en estwe trabajo y una selección final con las características que se aproximen a los objetivos dados.

2.1 Centro de procesamiento de datos

Los *datacenters* o centro de procesamiento de datos (CPD) son instalaciones especializadas que centralizan las operaciones de los servicios de la información de una organización y el equipamiento necesario para almacenar, procesar y distribuir los datos valuables entre todos sus usuarios. [10]. Los centros de procesamiento de datos se utilizan para proveer servicios de alta disponibilidad y alta demanda a través de internet, almacenamiento masivo de datos a los que se les puede aplicar técnicas de *Big Data* y para fines científicos o de investigación que requieran de grandes recursos de cómputo.

Los CPD *on-premise* o tradicionales son mantenidos completamente por la organización que lo utiliza. Esto aporta ventajas como control total de la infraestructura a todos los niveles, especialización para tareas concretas y seguridad, pero cuenta con las desventaja del coste y tiempo de mantenimiento, así como una escalabilidad limitada por la organización.

En los últimos años los CPD han evolucionado junto con nuevos modelos de computación que mejoran o directamente eliminan estas desventajas, creándose los CPD en la nube (*cloud datacenters*). A diferencia de los CPD clásicos, los CPD en la nube dependen de una infraestructura virtualizada sobre varios *clusters* distribuidos en diferentes emplazamientos e interconectados a través de internet protegidos con las tecnologías de cifrado más avanzadas posibles [11]. Estos *clusters* suelen estar gestionados por una organización diferente a la que hace uso de los recursos virtuales. La ventaja de este modelo es que se eliminan por completo las necesidades de mantenimiento del CPD virtual, también las de escalabilidad, siendo esta la única responsabilidad del proveedor de los *clusters*. Esto hace de este tipo de CPD un sistema reducido en costes y flexible [12]. Como desventaja se pierde la especialización y el control característicos del los CPD *on-premise* [13].

2.1.1 Sistemas de computación distribuida

Podemos clasificar los componentes de un CPD clásico (*on-premise*) en las siguientes categorías:

- **Instalación física:** el emplazamiento habilitado especialmente para la infraestructura principal del CPD y todos los sistemas de ayuda que mantienen su correcto funcionamiento.
- **Sistemas de soporte:** sistemas de climatización y sistemas de alimentación ininterrumpida.
- **Infraestructura de cómputo y almacenamiento:** recursos computacionales y software del CPD, agrupados en *clusters*.
- **Infraestructura de interconexión y conectividad:** dispositivos de capa 2 y 3 del modelo OSI (*switches* y *routers*), cableado, balanceadores de carga y software coordinador que une los componentes.
- **Infraestructura de seguridad:** puede ser física y virtual. La física se refiere a la seguridad de la instalación, por ejemplo, sistemas de vigilancia y control de acceso por personal autorizado. La virtual se refiere a los componentes software y hardware que protegen los datos y las aplicaciones, como *firewalls*, *VPNs*, ACLs o sistemas de monitorización.
- **Personal de mantenimiento:** equipo cualificado para operar el CPD.

Un *cluster* es una agrupación de múltiples componentes físicos interconectados entre sí mediante una infraestructura basada en red, permitiendo un uso coordinado de los recursos del sistema como una única unidad lógica. Esta arquitectura presenta la ventaja de poder ampliar o restaurar la disponibilidad con sólo añadir o reemplazar las unidades informáticas necesarias en la red. Existen diferentes tipos de *clusters*:

- ***High performance computing* (HPC):** centrado en ejecutar una sola tarea con toda la potencia disponible de sus nodos. Por ejemplo en experimentos científicos.
- ***High Throughput computing* (HTC):** se centra en ejecutar el mayor número de tareas aisladas de forma concurrente, utilizando las CPU de los nodos que tenga disponible. Los modelos de computación en la nube hacen uso de este modelo para ofrecer recursos a múltiples usuarios.
- ***High Avalilability computing* (HA):** centrado en proporcionar un servicio a prueba de fallos, mediante sistemas de control y redundancia. Por ejemplo, balanceo de carga o backup. Este modelo se encuentra en sistemas de computación en la nube.

2.1.2 Sistemas de computación en la nube

El instituto nacional de estándares y tecnología estadounidense (NIST) define la computación en la nube como un modelo que permite un acceso bajo demanda, conveniente y ubicuo a través de la red a un conjunto de recursos informáticos (por ejemplo: redes y servidores, servicios, almacenamiento) altamente configurables y con la capacidad de desplegarse de forma rápida y sin apenas esfuerzo por parte del proveedor. [14]. La computación en la nube puede clasificarse de dos formas diferentes, según el servicio que ofrecen y el modo de despliegue.

Los sistemas de computación en la nube se clasifican en varios **modelos de servicio** según el tipo de recursos que ofrecen, tal y como se representa en la figura 2.1:

- **SaaS o *Software-as-a-service***: la nube ofrece una solución software como servicio. Es el modelo más restrictivo, ya que toda su gestión se reduce a una sola aplicación. El proveedor del sistema en la nube gestiona la infraestructura hardware y software y el consumidor solo puede gestionar la aplicación. Por ejemplo, servicios de Google como Gmail o servicios *streaming* como Netflix entrarían en este grupo.
- **PaaS o *Platform-as-a-Service***: la nube ofrece una plataforma software como servicio. En este modelo de servicio el consumidor gestiona la infraestructura software al completo y puede crear y ejecutar sus propias aplicaciones software. Por ejemplo, Microsoft Azure ofrece soluciones para desplegar servidores o contenedores donde los consumidores pueden alojar sus propias aplicaciones.
- **IaaS o *Infrastructure-as-a-Service***: la nube ofrece recursos hardware como servicio. Es el modelo más flexible. El consumidor tiene control de los recursos hardware y la infraestructura software mediante máquinas virtuales. Por ejemplo, Elastic Compute Cloud de Amazon o Google's Compute Engine ofrecen servicios basados en IaaS.

Uno de los principales aspectos a destacar es el nivel de control que se le otorga al usuario en cada tipo de modelo. El nivel de control se muestra en la figura 2.1 separado por una línea roja. Existe un equilibro entre qué parte de la infraestructura es controlada por el proveedor y qué parte controla el usuario. Cuanto mayor control tenga el usuario mejor puede adaptarse la nube a sus necesidades a costa de el tiempo y costes de mantenimiento. Es interesante notar que también se incrementa la superficie de ataque ante un usuario malintencionado.

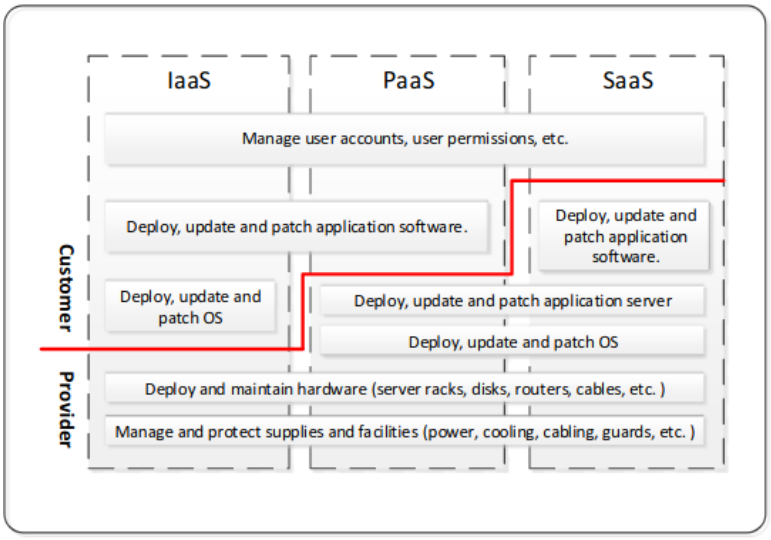


Figure 2.1: Modelos de servicio en la computación en la nube y nivel de control y responsabilidad del consumidor final sobre la infraestructura final. [2].

Según el tipo de despliegue del modelo, la nube puede clasificarse en **privada** si su uso está limitado a la organización y se encuentra completamente bajo su control, **comunal** si se comparte entre diferentes organizaciones que pertenecen a un grupo concreto y **pública** si los usuarios no poseen relación con la organización que lo gestiona. Existe un tipo **híbrido** en el que la nube es gestionada por una organización ajena a aquella que hace uso del servicio, pero el uso de servicio es privado.

Ventajas e inconvenientes [15]:

- **Abstracción de la infraestructura a los usuarios**: los usuarios no necesitan conocer la infraestructura subyacente para operar. Esto permite al proveedor desplegar o actualizar nueva infraestructura bajo demanda sin afectar al funcionamiento global del sistema, proporcionar al usuario los recursos computacionales y software que necesita sin necesidad de su interacción y a los administradores gestionar, monitorizar y actualizar los componentes de forma transparente al usuario.
- **Elasticidad y disponibilidad**: todas las capacidades del modelo son accesibles a través de la red mediante protocolos estándar que permiten su uso en múltiples plataformas heterogéneas. Esto aporta disponibilidad de los recursos completa al usuario. Los recursos son servidos a los usuarios bajo demanda y *multi-tenant*. El término *multi-tenant* se refiere a un sistema en el que las organizaciones y usuarios comparten recursos y los gestionan desde su propio entorno independiente al resto.

- **Ahorro de costes**: los recursos que componen la arquitectura suelen ser homogéneos y uniformes lo que permite simplificar las tareas de mantenimiento del sistema y operacionales. Minimiza los costes y tiempo de gestión, además de asegurar una arquitectura escalable. También permiten facilitar y reducir los costes de implementación de sistemas de replicación, aumentando la fiabilidad y la disponibilidad de los datos e infraestructura y reducir su exposición, lo que repercute en la disponibilidad.

Entre los inconvenientes destacan:

- **Pérdida y limitación de control**: un sistema complejo, con cientos de componentes y miles de usuarios interactuando entre sí pueden generar problemas de servicio o fallos de seguridad que pueden ser aprovechadas por los atacantes si no se gestiona correctamente. Esto también implica una pérdida de control por parte del consumidor y el proveedor, ya que según el modelo utilizado pueden tener menor control sobre el sistema.
- **Disponibilidad de servicio y acceso a internet**: fallos en los servicios del proveedor de internet, del propio centro de datos o simplemente incapacidad de conectarse pueden dejar a los usuarios inoperativos durante un tiempo determinado.
- **Confidencialidad y seguridad en los datos**: el tipo de despliegue de la nube y su modelo de servicio implican riesgos para la seguridad. Un compromiso de seguridad en el CPD puede afectar a todos los usuarios del sistema. En la nube, todos los componentes están expuestos a internet, lo que implica una gran superficie de ataque y un gran riesgo.

2.2 Seguridad en el CPD

La **seguridad informática** es el conjunto de políticas y protocolos que se aplican sobre los componentes de un sistema con el objetivo de evitar intrusiones que conlleven la manipulación o fuga de los datos y el uso indebido o sabotaje de los componentes informáticos. Todo sistema de seguridad debe cumplir con los siguientes principios básicos [16]:

- **Integridad**: garantiza la autenticidad y validez de la información.
- **Confidencialidad**: garantiza que los datos solo son alcanzables por las personas autorizadas.
- **Disponibilidad**: determina el grado de fiabilidad de los sistemas de la información y su capacidad de delegar los datos solicitados en el momento, tiempo y forma en los que son requeridos.

Con estos puntos en mente, podemos identificar las principales amenazas y entender las políticas existentes en un centro de procesamiento de datos. Podemos agruparlas en seguridad física y seguridad virtual. [17]. En la seguridad física identificamos riesgos como la pérdida de suministro eléctrico o desastres naturales que pueden generar una denegación de servicio temporal o permanente. También pueden ocurrir accesos por personal no autorizado que pueden desembocar en robo, pérdida o sabotaje del hardware y sus datos. Entre las principales contra-medidas existentes los centros de procesamiento de datos suelen contar con redundancia para mantener la disponibilidad y evitar pérdidas tanto de recursos como de la información, así como controles de acceso de personal autorizado y sistemas de control de integridad de hardware. Es siempre responsabilidad del proveedor implementar la seguridad física del CPD. Por ejemplo, muchas organizaciones se certifican con la ISO 27000 enfocada en el Sistema de Gestión de Seguridad de la Información (SGSI).

La seguridad virtual define las medidas a los servicios y los propios datos que gobiernan el CPD. Es la más complicada de tratar debido a la complejidad de los componentes y el gran número de formas de comprometerlos, dependiendo de la arquitectura y el modelo de computación utilizado pueden presentarse riesgos adicionales. A continuación se desarrolla el estado de la seguridad virtual de un centro de procesamiento de datos con un modelo de computación en la nube.

2.2.1 Seguridad en la nube

La computación en la nube presenta una serie de características que refuerza y complica la seguridad del centro de datos. Los recursos que componen la arquitectura suelen ser homogéneos y uniformes, lo que permite una mayor automatización de prácticas de seguridad y auditoría y por lo tanto un mayor refuerzo de la seguridad física. Del mismo modo, homogeneidad implica que una misma brecha de seguridad se manifiesta a través de toda la infraestructura, lo que puede ser muy difícil de corregir si no existe un parche efectivo.

La centralización de datos y compartir infraestructura entre varios usuarios (*multi-tenant*) también facilita su acceso a los atacantes. Debido a que el modelo de la nube se basa en separación lógica y no física de los datos, un ataque interno que pueda sobrepasar las medidas internas de contención puede tener acceso a todos los recursos de una organización y afectar a todos sus usuarios. La forma de solucionar esto es separar los recursos mediante virtualización y utilizar sistemas de control de acceso como ACLs, así como dispositivos cortafuegos a nivel de red.

En cuanto a los modelos de servicio, cuanto mayor sea el control del usuario sobre la infraestructura, mayor es la exposición de componentes a usuarios ajenos. Es más, también crece la responsabilidad del consumidor de mantener parcheado y debidamente configurado los componentes de la nube, por lo tanto existe una superficie de ataque mayor. Aquellos CPD en la nube que proporcionen acceso de infraestructura (IaaS) son más vulnerables

porque proporcionan un mayor control al usuario final.

De no implementar medidas de seguridad de la forma correcta las consecuencias pueden ser devastadoras. Por ejemplo, el CPD podría ser víctima de un ataque por Ramsomware como el conocido WannaCry [5] o los ataques producidos contra la infraestructura de los servicios de salud de Reino Unido y Irlanda [6].

2.2.2 Intrusiones

Una **intrusión** es una actividad ilícita en una red o sistema informático. Se considera actividad ilícita a cualquier acción que resulte en una violación de las políticas de seguridad del sistema en donde se producen. Algunas intrusiones pueden deberse a un simple desuso por parte de usuarios legítimos, debido a fallos de configuración por ejemplo, mientras que otras se realizan de forma malintencionada con el fin de causar daños al sistema.

Axelsson [18] clasifica las intrusiones que pueden ocurrir en un sistema como:

- **Intrusiones conocidas**: amenazas que son bien conocidas, predecibles y por lo tanto son fáciles de detectar. Tienen un grado muy bajo de variabilidad.
- **Intrusiones generalizadas**: similares a las intrusiones conocidas, pero con mayor grado de variabilidad.
- **Intrusiones desconocidas/emergentes**: amenazas poco conocidas y con alto grado de variabilidad. Son peligrosas porque interfieren en el sistema de forma impredecible. Difíciles de detectar.

Una intrusión siempre conlleva una serie de riesgos que puede comprometer seriamente la privacidad de los usuarios, sin importar si se produce por accidente, ya que expone un fallo de la política de seguridad del sistema que más tarde o más temprano será explotado por atacante. Podemos identificar varias fases o estadios de una intrusión intencionada [19]:

1. **Reconocimiento**: el principal objetivo de un atacante en esta fase es recoger tanta información como le sea posible de un sistema con el fin de detectar sus puntos débiles (vulnerabilidades, debilidades en la topología, etc). Para realizar esta tarea, el atacante hará uso de técnicas de escaneo y recolección de información, mayormente pasivos, es decir, que no generen paquetes ya que es importante evitar la detección durante el mayor tiempo posible.
2. **Infiltración inicial**: uso de *malware* como troyanos, suplantación de identidad, *spam*,

phising o *spear phising* [1]. Uso de vulnerabilidades como desbordamiento de *buffers* o fallos en los hypervisores. El objetivo es conseguir acceso al sistema o por lo menos a un componente periférico del mismo.

3. **Consolidación en el sistema**: uso de técnicas de persistencia, (rootkits, APT), puertas traseras, escalada de privilegios (*user 2 root*) que permitan al atacante tener un control mayor del sistema y lanzar nuevos ataques desde dentro. Esto implica la instalación de herramientas que permitan al atacante realizar su objetivo.

4. **Movimiento lateral**: uso de *malware* como gusanos, credenciales robadas, o técnicas similares al punto anterior para comprometer nuevos componentes de la red y aumentar su superficie de acción. Por ejemplo, el ransomware Wannacry infectaba otros equipos expandiéndose a través de la vulnerabilidad *EternalBlue* [20] que abusaba del puerto SMB de sistemas operativos Windows.

5. **Ejecución del ataque**: una vez reunido la información necesaria y tener un nivel de acceso lo suficientemente alto al sistema, el atacante ejecuta su objetivo final: fuga de datos, extorsión, daño de componentes, denegación de servicio, entre otros.

2.2.3 Principales riesgos

A continuación se describen los principales riesgos que podemos encontrar en un CPD. Los ataques llevados a cabo por un atacante se clasifican siempre en uno de estos tipos:

Denegación de servicio: es un ataque que tiene como objetivo provocar un consumo alto de recursos de computación y de red, impidiendo el desempeño de las tareas habituales y perjudicando a los usuarios. Los ataques DoS clásicos se ejecutan desde un único sistema atacante, mientras que los DDoS *(distributed denial of service)* involucran varios sistemas atacantes al mismo tiempo, siendo habitual el uso de botnets par realizarlos [21]. La solución a estos sistemas pasa por usar cortafuegos o balanceadores de carga. Un sistema de monitorización es necesario para detectar pronto este tipo de ataques de forma que pueda bloquearse o liberar los recursos afectados lo antes posible. Es necesario también tener cuidado con los sistemas de monitorización y seguridad ya que también pueden ser susceptibles de un ataque de denegación de servicio y usarse en favor del atacante para evadir la detección de futuras intrusiones.

Fuga o exposición de datos: la filtración de información de usuarios o de la organización puede tener repercusiones muy graves. Los mayores daños son pérdidas financieras y de confianza, tanto de la empresa como de los usuarios cuya información ha sido comprometida.

[1] Las técnicas *phising* consisten en suplantar la identidad de una organización o persona a través de internet con el fin de ganarse la confianza de sus víctimas, con el objetivo de obtener dinero o credenciales de acceso para realizar movimiento lateral o realizar chantaje. El *spear phising* son engaños personalizados y dirigidos a una persona o grupo concreto, con el mismo fin.

Vulnerabilidades: podría definirse como una característica o estado de un componente del sistema que puede provocar un comportamiento no autorizado. Fallos de seguridad en los componentes software y hardware de la nube o una indebida configuración de los mismos puede convertirse en vectores de entrada de un ataque. Además, dependiendo del modelo de computación, cuando mayor sea el control del consumidor, mayor probabilidad de encontrar estas vulnerabilidades.

Ataques internos: ataques desde la red interior provocados por atacante infiltrado que está realizando un movimiento lateral, escáner de puertos o violación de las políticas de seguridad físicas del CPD.

Accesos no autorizados: escalada de privilegios, mala gestión o configuración de listas ACL o robo de identidad.

Suplantación de identidad: Antiguos usuarios legítimos del sistema que todavía tienen acceso a los recursos o robo de identidad y enmascaramiento por parte de un atacante exponen un riesgo de ataques que poseen la ventaja de de realizarse en el interior del perímetro de seguridad, evitando la mayoría de defensas.

2.3 Sistemas de detección y prevención de intrusiones

Un sistema de detección de intrusiones (IDS) es un software que monitoriza el estado de un sistema a través de los eventos que ocurren en el mismo y es capaz de detectar anomalías en base a esos eventos, todo ello de forma automatizada. Los IDS actúan como una alarma de seguridad que notifica cualquier actividad sospechosa que se detecte en el sistema, registrando y guardando en logs todos los eventos para su posterior consulta por parte de administradores y auditores. Pueden monitorizar en tiempo real o de forma periódica el sistema, conectado o desconectado de la fuente de datos, pero proporcionándole una muestra cuando sea necesario.

Un sistema de prevención de intrusiones (IPS) posee todas las capacidades de un IDS convencional y además puede intervenir al sistema que monitoriza, deteniendo un ataque si lo detecta a tiempo, por ejemplo, terminar una conexión en curso o bloqueando los paquetes igual como lo haría un *firewall*. La arquitectura y posición del IPS en una red es determinante para que se puedan aplicar estas contra medidas, ya que no sirve de nada realizar acciones si la fuente de eventos no es en tiempo real, no tiene pleno control sobre el tráfico de una red o no guarda relación con los puntos de interés de la red a proteger.

Hoy en día la mayoría de soluciones de detección de intrusiones ofrecen capacidades IPS, pero pueden utilizarse como IDS dependiendo de las necesidades o de su rol en la estrategia de seguridad. A partir de este momento, nos referiremos al software IDS/IPS como IDPS.

Los IDPS realizan además otras funciones interesantes [22]:

- **Identificar problemas en las políticas de seguridad**: una aplicación incorrecta de medidas de seguridad en el sistema puede originar comportamientos en este que pueden ser identificados por un IDPS. Esto permite descubrir fallos de implementación de seguridad por parte de los administradores y corregirlos rápidamente. Por ejemplo, una mala configuración de un *firewall* provoca una fuga de tráfico o permite un acceso remoto que no debería poderse realizar. Un IDPS puede detectar tráfico de red en los puertos que deberían estar bloqueados.
- **Documentar la clase de peligros y su frecuencia en el sistema**: los IDPS registran todas las incidencias lo que permite a los analistas crear un modelo de los tipos de amenazas que sufren y desarrollar una estrategia óptima ante ellas.
- **Limitar las acciones de los atacantes del sistema o la red**: si un atacante es consciente o sospecha de la existencia de un IDPS puede reconsiderar realizar alguna acción que entrañe el riesgo de ser detectado.

2.3.1 Metodologías de detección

La principal forma de clasificar un IDS es según su método de detección o método de análisis. Determina cómo el IDS extrae la evidencia de intrusiones de la muestra de datos. Principalmente, hay tres grandes grupos de metodologías [23] [22] [24]:

Detección basada en firmas

El IDS localiza intrusiones de las que conoce un patrón o huella predefinida que identifica esa amenaza. Es el caso del análisis basado en firmas, donde las firmas son reglas que contienen estos patrones. [18]. Por ejemplo, un intento de inicio de sesión por telnet con el usuario "root" o una petición HTTP con una cabecera utilizada por un *malware*.

Este método de análisis tiene la ventaja de tener una baja probabilidad de falsos positivos mientras que los verdaderos positivos están bien determinados y es más fácil comprender su naturaleza (causa, acciones, objetivos...). Sin embargo, al ser este método de detección bastante rígido, presenta problemas para detectar amenazas cuyo patrón o huella se desconoce o ha mutado lo suficiente respecto a la versión de la base de conocimiento que no provocan una alerta [22] [25].

Otro problema es que la detección de firmas no maneja los estados de las comunicaciones, lo que es esencial para identificar comunicaciones complejas o saber discernir una amenaza de una falsa alarma basándose en el contexto. Por ejemplo, no pueden asociar una petición

con su respuesta y por lo tanto no pueden saber si esa petición ha provocado una respuesta de acceso denegado a un recurso web, lo que puede ser un indicio de un atacante intentando acceder a un recurso al que no debería tener acceso.

Detección basada en anomalías

En este tipo de análisis no existe un patrón predefinido de la intrusión, sino que se deduce al compararlo con un modelo base que el IDS posee del sistema, considerado el estado normal [18]. De este modo, las amenazas se identifican como excepciones o desviaciones del modelo.

Este método presenta la ventaja de que es capaz de detectar amenazas emergentes de las que no existen firmas o no se asemejan a ninguna parecida, aunque es propenso a generar falsos positivos, ya que el modelo no siempre se ajusta a la realidad o no responde bien ante cambios muy rápidos del patrón de uso de un sistema por parte de los usuarios legítimos lo que puede generar falsas alertas con mucha facilidad [25]. Además, si un ataque presenta un comportamiento poco variable respecto al modelo podría pasar desapercibido y tardar un tiempo en ser detectado, o no ser detectado si quiera.

El sistema de detección debe generar el modelo de referencia y eso implica entrenar y observar la actividad del sistema durante algún tiempo, pero nada garantiza que ese sistema muestre un comportamiento "normal" desde el primer momento, por ejemplo, es posible que el sistema ya haya sido comprometido y muestre actividad maliciosa que se considere como normal por parte del motor de detección, o que el atacante, siendo consciente de que los IDPS con esta metodología de detección necesitan tiempo de entrenamiento, opten por adaptar sus ataques para que escalen de una forma progresiva para confundir al IDPS y que integre este comportamiento como parte del modelo.

Otro problema es la dificultad de analizar una amenaza debido a la cantidad y complejidad de eventos que han podido disparar la alerta.

Análisis de estados de protocolo

Es el proceso de comparar los estados de una comunicación (o protocolo) con un modelo o perfil de referencia del buen funcionamiento de ese protocolo con el fin de detectar desviaciones que puedan identificar ataques que por ejemplo abusen del protocolo para esconder su actividad [22]. El modelo no es dependiente del sistema sino que se basa en reglas universales establecidas por el fabricante o las instituciones que desarrollan el protocolo.

Como ventaja este método aporta la capacidad de detectar amenazas que solo son detectables cuando se analizan todos sus estados en conjunto. También hace más fácil analizar las propiedades y procedencia de un ataque ya que permite etiquetar la actividad con un usuario,

por ejemplo poder constatar el login con un usuario o máquina con todas sus acciones posteriores.

Sin embargo, consume muchos recursos y memoria para seguir los estados de todas las sesiones de comunicaciones. Esto lo hace propenso a técnicas de denegación de servicio dirigidas al IDPS. Los modelos de referencia pueden no tener una especificación completa y la existencia de implementaciones ambiguas provocan que los modelos fallen en detectar ciertas amenazas.

2.3.2 Tipos de tecnologías IDPS

Basado en host (HIDS)

Los sistemas de detección de intrusiones basados en host monitorizan todas las actividades de un sistema operativo y aplicaciones de un host en concreto. Recogen registros del sistema, analizan los ficheros creados o accedidos y monitorizan las conexiones del nodo. Se encuentran instalados en el host que analizan. Ejemplos de HIDS: OSSEC [26], AIDE [27].

Basado en red (NIDS)

Los sistemas de detección de intrusiones basados en red monitorizan la red local a la que pertenecen. Se sitúan en un punto clave de la red, en donde pueden interceptar todas las comunicaciones posibles y recoger los paquetes de datos de esas comunicaciones. Se especializan en analizar los protocolos y las capas TCP/IP de la comunicación. En su variante preventiva (IPS) pueden funcionar como un *firewall* adaptativo. Ejemplos de NIDS: Snort [28], Suricata [29], Bro (ahora llamado Zeek) [30].

Network Behaviour Analysis (NBA)

Monitorizan el flujo de la red para identificar amenazas que generen tráfico inusual de red. Estos sistemas nacieron como una solución especializada en detectar ataques de denegación de servicio. También permite etiquetar la actividad a cada usuario. muchas soluciones IDPS de hoy en día incluyen capacidades NBA y detección basado en estado de protocolos.

Algunas soluciones IDPS incorporan funcionalidades NBA, pero también existen como soluciones hardware independientes especializadas. Algunos NBA no monitorizan la red directamente, sino que se basan en las estadísticas recopiladas por routers y switches.

Los NBA poseen la mayor capacidad de detección para ataques que generan una gran cantidad de trafico en poco tiempo o ataques que tienen patrones de tráfico inusuales.

Distributed IDPS (DIDPS)

Estos sistemas se componen de una combinación de los tipos de IDPS vistos hasta ahora (por ejemplo, NBA/NIDS para la red y HIDS para los nodos que se desean observar), que actúan como los monitores de toda la infraestructura, interconectados entre sí por una red de administración y gestionados por un servidor de control central.

2.3.3 Componentes

- **Sensor o agente**: Los sensores son instancias del motor de detección que monitorizan la actividad. Cuando se sitúan en la red, se llaman sensores, cuando se sitúan en un host, agentes. Un IDPS puede tener uno o varios sensores.
- **Servidor de administración**: dispositivo centralizado que recibe la información recolectada por los sensores o los agentes y la administra. Estos servidores, al recibir eventos de distintas fuentes y naturaleza, pueden realizan un análisis adicional sobre la información recolectada detectando anomalías que los sensores o agentes por separado no podrían detectarlas. Este proceso se conoce como correlación.
- **Servidor de base de datos**: dispositivo que proporciona el repositorio para guardar la información recolectada por sensores, agentes o servidores de administración.
- **Consolas**: interfaces de gestión del IDPS para los usuarios del mismo.

2.3.4 Limitaciones

- **Incapacidad de analizar tráfico encriptado**: comunicaciones VPN, SSL, HTTPS, SSH, por ejemplo son imposibles de analizar por el IDPS debido a que no posee la clave privada necesaria para desencriptar la comunicación. Algunos IDPS ofrecen capacidades de análisis de las fases iniciales de comunicaciones cifradas y pueden ayudar en detectar configuraciones erróneas o peligrosas, pero la única forma de realizar una análisis completo es asegurarse que el IDPS analice los *payloads* antes o después de su encriptación. Esto puede lograrse si el sensor se sitúa en una sección de la red que es privada y por donde no viaja comunicación cifrada, por ejemplo en una la red local accesible por VPN, o utilizar un IDPS a nivel *host*.

- **Gestionar altas cargas de tráfico**: los sensores pueden saturarse en situaciones de altas cargas de tráfico. Pueden llegar a descartar paquetes sin analizarlos lo que puede limitar la detección de amenazas especialmente si se utiliza *Stateful protocol analysis*. La capacidad de análisis de un sensor depende del nivel de profundidad de análisis, la longevidad de las conexiones y el número de conexiones simultáneas, así como la CPU y el hardware de captura de paquetes.
- **Ataques al IDPS**: los sensores IDPS basados en red son vulnerables a varios tipos de ataques, especialmente los ataques de denegación de servicio si los sensores no gestionan bien las altas cargas de tráfico o provocando un uso muy elevado de recursos por parte del motor de detección. Generar un número muy alto de alertas en un corto periodo de tiempo o inutilizar la capacidad de generar alertas tiene como fin ocultar las auténticas amenazas frente a los administradores o provocar fallos en el IDPS. Esta técnica se conoce como *blinding* y se aprovecha de las configuraciones por defecto.

2.4 Sistemas de detección de intrusiones basados en red

En el entorno de pruebas se trabajará con IDS basados en red (NIDS). Este tipo de IDS es el más adecuado para estructuras masivas como un CPD, ya que permiten reducir costes de instalación y mantenimiento de sensores, al monitorizar la red interna en lugar de monitorizar cada nodo individual o monitorizar grupos según su función. También pueden servir para fortificar la red perimetral del CPD y detectar los primeros indicios de intrusión o incluso cerrar puertos para evitar propagación de malware y salvar ancho de banda. A continuación se describe brevemente las principales soluciones NIDS de seguridad de código abierto disponibles actualmente.

2.4.1 Snort

Snort es uno de los IDS más maduros y populares, contando con una amplia comunidad de usuarios y respaldado y mantenido por Cisco [28]. Es de código abierto (licencia GPL), lo que conlleva que miles de personas puedan revisar y aportar mejoras al código, combinado por su versatilidad y modularidad, destaca por ser utilizado ampliamente en el ámbito académico y profesional.

Cuenta con una base de datos de reglas inicial mantenida por la comunidad, pero también dispone de bases de datos mantenidas por empresas especializadas, como por ejemplo Talos [28]. Algunas de estas bases de firmas comerciales requieren subscripción de pago para actualizaciones inmediatas pero permiten su uso gratuito a partir del mes de ser publicada. Snort posee una de las bases de datos de firmas más completa.

El método principal de detección de Snort se basa principalmente en el análisis de firmas. Cuenta con varios modos de funcionamiento: puede utilizarse como un recolector de paquetes (*sniffer*), un *logger* de paquetes creando registros de ellos en el disco o como un IDS/IPS completo. Dispone de decodificación y análisis automático de protocolos de capa IP y de aplicación, análisis HTTP, análisis de ficheros (experimental) y sistemas de reputación IP basado en listas. Snort dispone de un sistema de plugins basado en preprocesadores que consisten en módulos de análisis personalizados que pueden añadirse al motor de detección base. Pueden crearse reglas personalizadas y ajustar el comportamiento del motor de detección mediante preprocesadores.

Snort puede generar registros de actividad directamente en formato de texto, redirigirlos a Syslog o generar ficheros CSV. En su última versión incorpora soporte para generar logs en formato JSON, que se ha convertido en el estándar de facto para los recolectores de eventos. En su versiones anteriores necesita de herramientas externas para convertir los logs de unified2 (formato binario) a JSON.

Snort cuenta a día de hoy con dos versiones principales: la versión estable 2.9 y la versión 3.0. La principal diferencia entre las dos versiones es que la versión 3 cuenta con una reestructuración de la arquitectura interna que aporta soporte multi-hilo así como un motor de detección menos limitado y más configurable que lo coloca a la altura de sus homólogos NIDS, un nuevo sistema de configuración basado en scripts y soporte nativo de formato JSON para los logs.

2.4.2 Suricata

Suricata es un IDPS diseñado con la finalidad de ejecutarse en entornos de alto rendimiento. Es *open source* con licencia GNU GPL 2 y es mantenido por la Open Information Security Foundation (OISF) [29].

Suricata es compatible con la base de datos de firmas con Snort, pero no completamente debido a las diferencias de características entre los motores de detección. Suricata usa las reglas de EMT, que aprovechan las capacidades de su motor de detección al máximo, pero EMT no es tan extensa como puede ser la base combinada de la comunidad de Snort y los distribuidores comerciales. La base de datos EMT requiere subscripción pero libera las actualizaciones de firmas entre 30 y 60 días para usuarios gratuitos.

El motor de detección está basado en firmas. Aporta decodificación y detección de protocolos automática, análisis HTTP, análisis TLS/SSL parcial, extracción con análisis de ficheros y sistema de reputación IP basado en listas. Suricata no cuenta con un sistema de plugins como Snort, pero dispone de herramientas para programar su motor de detección de forma personalizada, mediante *scripting*. Por ejemplo, con este sistema se pueden añadir nuevos protocolos. Puede funcionar, además de como IDS y IPS, como analizador de paquetes en

modo offline y *packet logger* a disco duro. Como un monitor de seguridad de red (NSM), es capaz, en base a los eventos que genera como IDS, recolectar todo el contexto de los eventos, construir su estado global y recuperar toda la información sobre el estado de la red en el momento el que el IDS realizó la detección lo que permite facilitar los procesos posteriores de auditoría. Un NSM está más orientado a auditar la red que detectar intrusiones en ella.

Suricata puede generar registros en formato JSON directamente y es compatible con unified2, además de soportar Syslog y formato de texto básico. Adicionalmente, genera registros de request HTTP, peticiones y respuestas DNS, flujo de tráfico, handshakes TLS e información acerca de ficheros extraídos y analizados. Mediante scripting (LuaJIT) permite definir formatos personalizados de logs.

En cuanto a rendimiento, Suricata posee un motor de detección multi-hilo nativo. Utiliza un parseador de alto rendimiento para generar los logs en formato JSON.

2.4.3 Zeek

Zeek (antiguamente Bro) es un *framework* con capacidades IDS/IPS centrado en la monitorización de red (NSM). Orientado a ser una herramienta de automatización para el analista. *Open source*, licencia BSD. Originalmente desarrollado por Vern Paxson, ahora es mantenido por Robin Sommer y un equipo de desarrolladores de la universidad de Berkeley. Soporta únicamente un hilo, pero cuenta con soporte *clustering* [30].

Zeek combina análisis de firmas con análisis de comportamiento. Proporciona análisis de contexto de la capa de aplicación, decodificación de túnel, análisis de los ficheros y análisis de metadatos. Cuenta con un motor de eventos, que realiza una detección inicial de lo que ocurre en la red (protocolo, origen, destino acción) pero no determina si es legítimo o malicioso. Su función es abstraer el flujo de datos de la red en una serie de eventos independientes. Estos eventos son analizados posteriormente por el intérprete [30]. El intérprete, en base a los patrones configurados mediante los scripts que definen la política de seguridad de la red a aplicar, analiza y clasifica los eventos generados. Los scripts pueden derivar información adicional acerca de los eventos, como estadísticas y trazar seguimiento continuo del contexto de los eventos, relacionándolos entre sí . En definitiva, los scripts determinan cómo construye Zeek su modelo de referencia de forma abstracta y cuando algo se desvíe de ese modelo de referencia, genera una alerta.

2.5 Selección de IDS

Snort cuenta con el mayor soporte, estabilidad y madurez posible en el mundo de los IDS. Su mayor ventaja es la calidad del repositorio de firmas de la que dispone, especialmente si

se emplea el método de subscripción que te da acceso a las listas de Talos [28]. Aunque su motor de detección no es tan completo de base como el de Suricata, puede ampliarse con preprocesadores y posee gran interoperabilidad con otras aplicaciones de análisis completo como los SIEM.

Suricata, en comparación a Snort posee un rendimiento mayor y un set de configuraciones más completo, así como personalizable mediante scripts. Su capacidad para recolectar información contextual de una anomalía es mayor que la de Snort, lo que facilita mucho su posterior estudio e identificación de la raíz del problema.

Zeek destaca por disponer de una metodología de detección basada en anomalías, y puede ser útil como motor adicional junto a un motor basado en firmas, como Snort y Suricata. En concreto, Zeek puede complementarse a Snort o Suricata aportando un sistema de detección frente a anomalías desconocidas junto a los analizadores basado en firmas. Su mayor punto negativo es la dificultad de programar las reglas del modelo de referencia. Su capacidad de detección depende en buena parte de esta programación.

Finalmente se implementa Suricata en lugar de Snort. Razones:

- Mismo formato de reglas de Snort, puede utilizar las mismas fuentes. Automatización y configuración avanzada de detección.
- Soporte *multithreading*. Snort ha incorporado esta característica recientemente con Snort 3.0, pero Suricata la tenido de base.
- Mejor rendimiento en redes de alta velocidad. Aceleración hardware.
- Logging del contexto de red en todo momento y información adicional de las alertas.
- Exportación de eventos de forma nativa en un formato ampliamente soportado por recogedores y analizadores de registros.

2.6 OpenStack

El despliegue del entorno de pruebas se realiza sobre un sistema de computación de nube gestionado por OpenStack. OpenStack es un software de código abierto pensado para la nube que gestiona los recursos de un centro de procesamiento de datos y ofrece un modelo de servicio basado en infraestructura (IaaS) [31].

Openstack está compuesto de una arquitectura modular basada en componentes y add-ons que aportan una funcionalidad específica, tal y como se muestra en la figura 2.2. Estos componentes cubren funcionalidades como por ejemplo: persistencia, red, computación o

monitorización. Los servicios más relevantes para este trabajo son: *Nova* [32], el servicio que gestiona las máquinas virtuales de OpenStack y *Neutron* [33], el servicio de red que en combinación con *Nova* aporta conectividad a las máquinas virtuales.

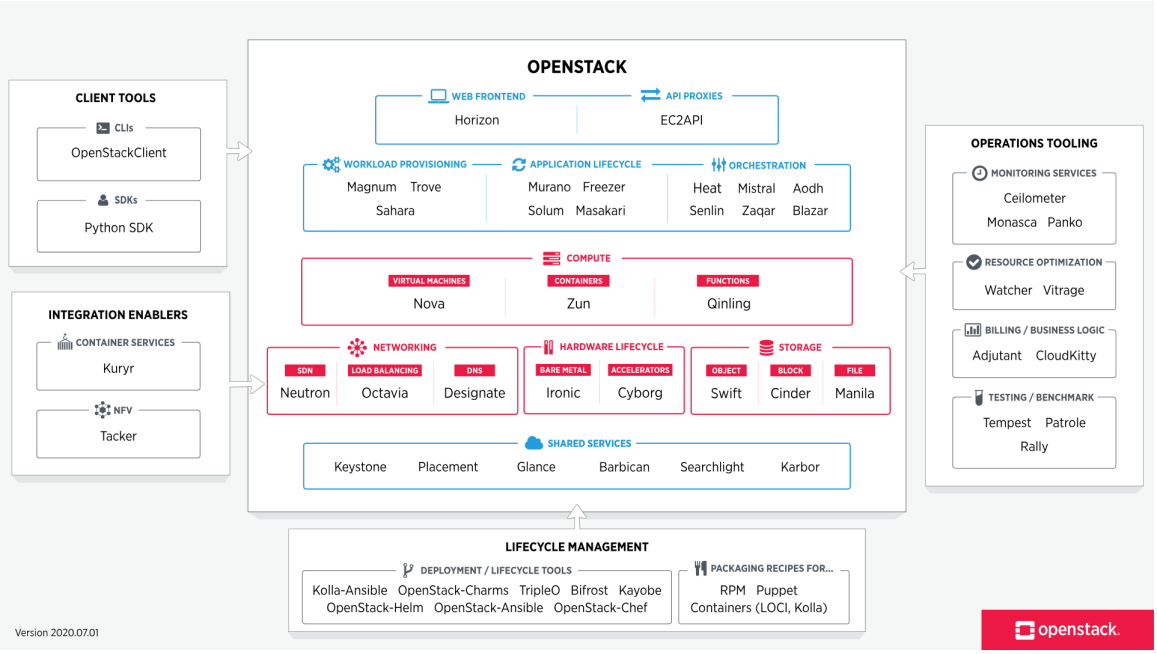


Figure 2.2: Estructura de OpenStack. Cada componente realiza una tarea concreta.

2.7 Docker

Docker [34] es una herramienta *open source* pensada para realizar despliegues de software de forma rápida y automatizada [35]. Las aplicaciones se despliegan en contenedores que de la misma manera que una VM, aislan el contenido del host, pero con una sobrecarga menor para el sistema. La ventaja de este sistema es que los contenedores son independientes del host donde se despliegan.

Los comandos de despliegue de una aplicación y su configuración se recogen en el fichero *Dockerfile*. Este sistema es útil para documentar paso a paso todos los detalles de una instalación y es ejecutable.

Docker-compose [36] permite definir despliegues multi-contenedor compuesto por varias aplicaciones. Por ejemplo: un fichero Docker-compose que tiene directrices de como desplegar y relacionar entre si dos Dockerfile: un servidor web y un servicio de bases de datos SQL. Esta función es interesante para desplegar un entorno completo para el IDS, incluyendo

aplicaciones adicionales para recolectar o mandar logs a un servidor centralizado.

3. Entorno de pruebas

En este capítulo se desplegará el IDPS elegido (Suricata) así como todos los componentes adicionales necesarios para preparar un entorno de pruebas que permita analizar todas las alertas que puedan generarse en un entorno de alto rendimiento. Para automatizar el despliegue y simplificar la configuración se hará uso de contenedores manejados con Docker [34] explicados en la sección 3.4. Las secciones posteriores se centran en la preparación y despliegue de cada componente del sistema, así como una explicación de las limitaciones encontradas y la solución por la que se ha optado.

3.1 Arquitectura del entorno IDS

La arquitectura base para el entorno de pruebas simplifica el entorno realista del CPD simulando una comunicación entre dos extremos, a los que podemos referirnos como *atacante* y *víctima*. El atacante representa un intruso desde internet o una instancia comprometida del CPD por el mismo. La víctima representa al propio CPD y sus instancias.

La posición del IDPS depende de lo que queremos que el sensor analice y su modo de funcionamiento. El modo IPS es interesante ya que trabajando conjuntamente con otros dispositivos de seguridad como cortafuegos permite tomar contra medidas rápidas, pero su instalación no es recomendable en entornos de alto rendimiento. Es necesario que pueda interceptar la comunicación entre los dos extremos, pero se transforma en un cuello de botella al tener que pasar todo el tráfico por el sensor, puede provocar denegación de servicio si clasifica actividad benigna como actividad maliciosa al poder tomar acción sobre las alertas y es más complicado configurar y mantener.

Configurar el IDPS en modo pasivo (IDS) no nos proporciona un sistema de respuesta automático, pero al no ser intrusivo el sistema es menos propenso a fallos por lo explicado anteriormente y no tiene como requisito colocar el sensor de modo que pase a formar parte del camino de datos de la red. Para estas configuraciones lo ideal es clonar el tráfico y mandar una copia al sensor IDS para su análisis.

3.2 Implementación en OpenStack

Las soluciones actuales existentes para Openstack consisten en replicar tráfico a nivel de *tenant* o instancia utilizando switches virtuales (tecnología *Open VSwitch*(OVS)) o dispositivos TAP y redirigiéndolo a una instancia VM con el sensor IDS desplegado [37]. Esta configuración es transparente para el sistema y fácilmente escalable. En las pruebas locales, la intención original era utilizar tres nodos de cómputo independientes con una red de alta velocidad a 10 Gbps: el atacante, la víctima y Suricata, conectadas a un *switch* virtual gestionado por OpenStack que permita clonar el tráfico generado entre las dos primeras instancias y reenviarlo a Suricata, pero debido a limitaciones de permisos en el servicio de red de OpenStack (neutron) configurar una red de pruebas a medida en donde se pueda redirigir el tráfico no resultaba posible. Para solucionar el problema se ha optado por un enfoque más práctico, compuesto de dos instancias virtuales:

- **atacante**: instancia desde donde se realizarán ataques simulados para poner a prueba las capacidades de detección y rendimiento de Suricata. Esta instancia cuenta con **8 núcleos virtuales** y **16 GB de RAM** con el objetivo de realizar varios tests de generación de tráfico de alto rendimiento.
- **suricata/víctima**: instancia objetivo de las pruebas del atacante. Se compone de dos partes principales: un falso servidor que actuará de víctima y una instancia de Suricata escuchando la comunicación. Ambas partes estarán separadas mediante contenedores de Docker. Esta instancia cuenta con **24 núcleos virtuales** y **48 GB de RAM** para alojar sobradamente ambos servicios sin que interfiera en las medidas de rendimiento.

Todas se basan en una imagen base con el sistema operativo Ubuntu 18.04. La figura 3.1 muestra la arquitectura final del entorno de pruebas:

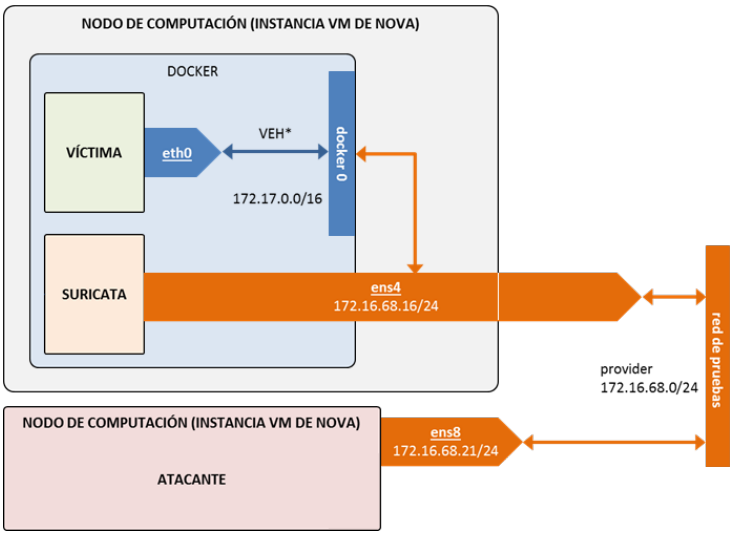


Figure 3.1: Arquitectura detallada de la implementación del entorno de pruebas local.

El camino de datos de una intrusión a analizar comienza en el nodo atacante y es dirigido a través de la interfaz *ens8* conectada a la red de pruebas (provider 2) a la instancia de Suricata, donde es recibido en la interfaz *ens4*, a velocidad de 10 gbps. Este tráfico es a su vez redirigido internamente al contenedor que recibirá el ataque, a través de la interfaz virtual puente de docker. El contenedor de Suricata esta configurado en modo *host* lo que significa que su interfaz virtual está asociada directamente a la interfaz real de la instancia, lo que le da acceso directo a todo el tráfico que es enviado y recibido en la instancia. Esto permitiría funcionar al IDS en modo cortafuegos, pero nos limitamos al modo IDS de momento.

Esta arquitectura presenta como ventaja es más fácil adaptarla para abrir un *honeypot* y realizar pruebas con intrusiones realistas ya que solo es necesario abrir la interfaz a un puerto externo.

Para administrar las instancias y dirigir las pruebas se hará uso de una red de administración proporcionada por el propio OpenStack, de tal forma que el tráfico de gestión no interfiera con el tráfico propio de las pruebas.

3.3 Configuración de Suricata

La configuración utilizada para las pruebas es la configuración original de Suricata con los cambios necesarios para activar el log extendido de eventos, ya que es interesante comprobar el rendimiento de Suricata en situaciones de estrés con un registro de eventos más completo. Toda la documentación de la configuración puede encontrarse en el código fuente o el manual

online [38]. Los ficheros más importantes de la configuración de Suricata son los siguientes:

- **classification.config**: Permite configurar la clasificación de alertas, establecer prioridades y descripciones.
- **reference.config**: contiene la lista de fuentes que a la que referencia una regla y al problema que detecta. Es una base de datos bibliográfica para los analistas que estudien las alertas detectadas por Suricata.
- **suricata.yaml**: fichero de configuración de Suricata.
- **threshold.config**: configuración de límite de registro de las alertas en un espacio de tiempo determinado. También sirve para establecer una cota de detecciones de una regla en un margen de tiempo determinado para generar una alerta.
- **update.yaml**: fichero de configuración utilizado por la herramienta de actualización y fuentes de reglas de Suricata.

La primera sección de la configuración muestra las variables utilizadas por las reglas de Suricata. El primer paso es poder diferenciar la red interna (HOME_NET) a proteger y la red externa (EXTERNAL_NET) a vigilar, en el fichero **suricata.yaml**. Para una primera aproximación se establece la IP del servidor como red interna (como la red interna se compone de un solo host, se asigna una única IP) y cualquier otra dirección como externa, invirtiendo la condición de la variable HOME_NET con "!". Esto es solo para el entorno de pruebas, pues la dirección o direcciones IP de las redes de producción a analizar determinará el valor de esta variable. En el caso del entorno OpenStack, HOME_NET sería una subred /16 o /24.

```
HOME_NET: "[172.16.68.16]"
EXTERNAL_NET: "!$HOME_NET"
```

El resto de variables indican la ubicación de ciertos servicios comunes . Se deja la configuración por defecto, indicando que se sitúan en "HOME_NET". Se mantiene la asignación de puertos a los protocolos por defecto.

El apartado de logging de suricata cuenta con numerosas opciones agrupadas en módulos para exportar la información recopilada en la detección en diferentes formatos (texto, json, pcap, etc) y con diferente cantidad de contenido. Se pueden dividir en dos grupos: logging referente a las alertas y contexto de red y logging de estado de Suricata. Dentro del logging de alertas, están los registros en texto plano y los registros en formato json. La configuración base activa por defecto el log rápido de alertas y el log en formato json de eventos de detección. También se activa por defecto el log de estado de proceso y la opción de registrar estadísticas de rendimiento de aplicación. Habilitamos el log de estadísticas de aplicación para obtener información valiosa sobre las pruebas:

```
default-log-dir: /var/log/suricata/

# Global stats configuration
stats:
  enabled: yes
  interval: 8
```

Y en la configuración específica de salida a fichero de stats, habilitamos estadísticas globales y hacemos que sean incrementales, es decir, que su valor se conserve y aumente conforme pasa el tiempo de ejecución:

```
  - stats:
      enabled: yes
      filename: stats.log
      append: yes       # append to file (yes) or overwrite it (no)
      totals: yes       # stats for all threads merged together
      threads: no      # per thread stats
```

Activamos el *output* del fichero eve.json. En este fichero se registran todos los eventos que recopila Suricata durante el análisis de tráfico, no solo aquellos que generan una alerta también aquellos eventos anómalos, estadísticas de rendimiento y eventos de tráfico por protocolo. Esto es así porque eve.json reúne todos los formatos posibles listos para ser interpretados por un software de análisis externo. Como queremos medir el rendimiento de logging, el eve.json es necesario activarlo porque sin duda será usado en producción.

3.4 Despliegue de Suricata

Para automatizar la instalación y configuración de Suricata se ha creado un fichero DockerFile personalizado (ver anexo A.1). El fichero DockerFile realiza lo siguientes pasos:

1. Descarga una imagen base para el contenedor. La imagen base contiene el sistema operativo y las utilidades básicas.
2. Configura el sistema operativo del contenedor e instala los componentes necesarios.
3. Copia la configuración de Suricata al contenedor y programa la actualización desatendida.
4. Establece las opciones de inicio.

La imagen base utilizada es la versión Docker de Alpine [39] un sistema linux ultraligero adecuado para contenedores. Por simplicidad se decide usar la versión pre-compilada de los repositorios de la distribución. Sobre la imagen base se aplican dos cambios: uno para establecer la zona horaria del S.O. y otro para instalar Suricata:

```
FROM alpine:latest
ARG TZ
RUN apk add tzdata vim --no-cache && \
    cp /usr/share/zoneinfo/Europe/Madrid /etc/localtime && \
    echo "$TZ" >  /etc/timezone
RUN apk add suricata --no-cache
```

El directorio que contiene la configuración de Suricata se copia al contenedor con el comando *COPY*. Para gestionar las actualizaciones se han creado un script que descarga las nuevas actualizaciones y una tarea CRON que ejecuta el script en el periodo estipulado y guarda el output en un fichero de log personalizado. Las actualizaciones se aplican en caliente sin necesidad de reiniciar el contenedor. Ambos ficheros se instalan en el contenedor con el comando COPY en los directorios adecuados del dentro del contenedor y se activa el bit de ejecución.

```
# Programar cron
COPY suricata-update.cron /etc/crontabs/suricata-update-cron
RUN chmod +x /etc/crontabs/suricata-update-cron
COPY suricata-update.sh /suricata-update-script.sh
```

Adicionalmente, se copia un script para actualizar las reglas de Suricata y aplicarlas en caliente y un cron para ejecutar la tarea de forma periódica de tal manera que mantenga las reglas actualizadas. El script genera un log llamado *update.log* para mantener un registro del estado de las actualizaciones (Anexos A.3 y A.4).

La última sección del DockerFile habilita la señal SIGINT para detener el contenedor si se ejecuta en primer plano y configura el punto de entrada del contenedor, que es el comando inicial que se ejecutará cuando sea desplegado. El *entrypoint* es un script copiado también al contenedor desde el directorio fuente que se encarga de arrancar los servicios necesarios (cron) y actualizar Suricata si es el primer arranque y lanzar el propio Suricata con los parámetros utilizados por el *entrypoint* (Anexo A.5).

Antes de desplegar el contenedor es necesario situarse en el directorio del DockerFile y compilarlo con el comando:

```
TZ="Europe/Madrid" docker build .
```

Esto genera la imagen del contenedor lista para ser desplegada. Para comprobar que la imagen se ha desplegado correctamente debe aparecer listada en la salida del comando *docker image*:

```
suricata ~# docker image ls
REPOSITORY        TAG         IMAGE ID       CREATED          SIZE
suricata          latest      6fc1b81182a8   2 months ago     118MB
```

Como a la hora de hacer las pruebas es necesario desplegar varias veces el contenedor de Suricata para probar varias combinaciones, se hace uso de ficheros *Docker-compose* para mantener escritas en un fichero de configuración las opciones de arranque del contenedor. El anexo A.2 muestra el fichero de despliegue para el entorno de pruebas local, que se encarga de levantar el detector de intrusiones Suricata. Todo el entorno de despliegue docker completo se encuentra en mi propio enlace de Github para crear una persistencia del trabajo [40] .

El fichero indica la imagen que acabamos de compilar y el nombre del contenedor, así como los parámetros de inicio de Suricata, que serán recogidos por el script del punto de entrada. Lanzamos Suricata en la interfaz *ens4* con el modo de captura de paquetes AF_PACKET [1]. Con la directiva *cpuset* establecemos el número de CPU reservados al contenedor. Si esta directiva se omite por defecto se asignan todas las CPU disponibles. Es vital otorgar al contenedor los permisos adecuados para acceder al recurso de red y recoger el tráfico. El modo de red *host* crea un puente virtual entre la interfaz del contenedor y la interfaz del host, lo que permite tener acceso a todo el tráfico [42]. Los permisos *"NET_RAW"* y *"NET_ADMIN"* permiten escuchar el tráfico en la interfaz del host.

```
suricata:
    container_name: suricata
    image: suricata:latest
    command: "-i ens4 --af-packet"
    cpuset: "0,1,2,3,4,5,6,7"
    network_mode: "host"
    cap_add:
      - NET_ADMIN
      - NET_RAW
```

Para desplegar la instancia puede hacerse directamente con el comando *docker run* [40] o utilizando un fichero docker-compose. En esencia, el fichero docker-compose hace exactamente lo mismo que *docker run* pero permite persistir algunos parámetros. Para verificar

[1]Suricata dispone de varias tecnologías de capturas de paquetes, pero AF_PACKET es la más moderna y eficiente que implementa Suricata.Además, AF_PACKET es una tecnología que está implementada en el kernel de Linux [41], por lo que no depende de librerías externas y proporciona mecanismos de balanceo de carga de paquetes

que el contenedor está online, basta con ejecutar el comando *docker ps*. Como el contenedor de Suricata funciona en modo *host* no tiene puertos mapeados, pues comparte la conexión de red del host directamente. La salida del comando no es realmente así, pero se ha formateado para que pueda leerse correctamente:

```
suricata ~# docker-compose -f /data/suricata.yml up -d
CONTAINER ID
b9bf98718e9aaa339c147920e1327c9f4c25379352d1ffd34eee0b3f034d5388
IMAGE
suricata:latest
COMMAND
"/entrypoint.sh -i ens4 --af-packet"
CREATED
1 min ago
STATUS
Up
PORTS

NAMES
suricata
```

4. Evaluación

En este apartado se somete Suricata a un conjunto de pruebas para determinar su capacidad de detección y rendimiento. Primero se realizan unas pruebas de control para comprobar que el entorno funciona debidamente en situaciones normales. Segundo, se somete Suricata a pruebas de rendimiento para verificar que la configuración rinde bien en situaciones de alta tasas de tráfico.

Para analizar el rendimiento de Suricata durante las pruebas se han tenido en cuenta los siguientes puntos:

1. Output del fichero suricata.log. Si existe un problema grave, aparecerá en el registro de estado del propio programa.
2. Uso de recursos: consumo de CPU, RAM e I/O generado por el proceso. La forma que suricata utiliza los recursos del sistema puede ser indicativo de una mala distribución de carga entre sus hilos [3].
3. Output del fichero stats.log. la tabla A.1 enumera las estadísticas más interesantes utilizadas en los tests.
4. Output de alertas: Generar demasiados falsos positivos también puede afectar al rendimiento.
5. Output de logs del sistema operativo, top, journalctl, dmesg, ethtool.

Los resultados se muestran según la media del tiempo de uso de las CPU en forma de porcentaje, el tamaño de la RAM en MiB y las métricas de detección detalladas en la tabla 4.1. Con esta información podemos calcular el rendimiento y la eficiencia del sistema de detección de intrusiones.

Verdaderos positivos (*True Positives*) **TP** **Anomalía presente, detección exitosa**	Falsos positivos (*False Positives*) **FP** **Sin anomalías, detección errónea**
Falsos negativos (*False Negatives*) **FN** **Anomalía presente, fallo en detectarla**	Verdaderos negativos (*True Negatives*) **TN** **Sin anomalías, sin detección**

Table 4.1: Métricas de detección

4.1 Evaluación de detección

4.1.1 Prueba de carga y activación de reglas

Objetivos

- Verificar carga de reglas EMT.
- Verificar detección de reglas.
- Verificar log de alertas básico.

Test

La base de datos EMT de Suricata dispone de una regla de prueba para testear el funcionamiento correcto del IDS con el id: 2100498. La regla busca en el contenido del paquete la cadena '*uid=0(root)*'. Para activar la regla se hace uso del servicio web *http://testmyids.com*. La repuesta se compone de la cadena:

```
uid=0(root) gid=0(root) groups=0(root)
```

Para probarlo es necesario ejecutar temporalmente Suricata escuchando en la interfaz de acceso a internet y desde la misma máquina que el IDS (ya que el entorno está configurado para analizar la interfaz de red que conecta las dos maquinas y con un rango de IP especifico). Para esto basta con cambiar el apartado *command* del fichero dockercompose o la interfaz en la configuración. Finalmente se hace una petición web usando la herramienta *curl*:

```
atacante ~# curl http://testmyids.com
```

Resultado

Si la regla está correctamente cargada en memoria y la configuración de alertas habilitada, Suricata debe generar una alerta con la siguiente cadena:

```
[1:2100498:7] GPL ATTACK_RESPONSE id check returned root
```

El entorno devolvió el resultado esperado:

```
01/08/2021-10:28:47.464529  [**] [1:2100498:7] GPL ATTACK_RESPONSE id check
↪  returned root [**] [Classification: Potentially Bad Traffic] [Priority:
↪  2] {TCP} 31.3.245.133:80 -> 172.16.73.22:5212
```

El formato de una alerta del fichero fast.log cuenta con las siguientes columnas:

- Fecha y hora con precisión de microsegundos del momento en el que se registró la alerta, en hora local.
- Identificador y grupo de la alerta que la identifican (en azul).
- Nombre descriptivo de la alerta (rojo).
- Clasificación de la alerta, que tipo de anomalía o ataque es (violeta). Esta información es útil para entender el evento y extraer información adicional.
- Prioridad (oro). Un orden entre 1 y 3 siendo 1 la prioridad más alta y reservada para eventos que requieren atención inmediata y 3, eventos que se corresponden con anomalías de fondo y de baja prioridad. Aún así, estos eventos pueden ser decisivos para detectar una intrusión si se analizan bien.
- Protocolo, origen, destino y dirección del paquete que ha generado la alerta.

4.1.2 Prueba de carga y activación de reglas personalizadas

Objetivos

- Verificar carga de reglas personalizadas.

Test

Para el test se crea una regla sencilla muy específica, usando el formato de reglas de Suricata [38]. La regla chequea la existencia de un agente de usuario con el texto 'AGENT SUSPICIOUS' que esté incluida en una petición HTTP:

```
alert http $EXTERNAL_NET any -> $HOME_NET any (msg:"TEST SUSPICIOUS
↪  USER_AGENT";
content:"User-Agent: "; http_header;nocase; content:"TEST SUSPICIOUS";
↪  http_header;
distance:0; sid:1; rev:1; classtype:string-detect;)
```

- **Acción**: La primera directriz (en rojo) indica el tipo de acción a realizar cuando se dispara la regla. Puede ser alert, drop, reject, accept, etc. En este caso solo genera un mensaje de alerta, pero no realiza ninguna acción de respuesta.
- **Cabecera**: La segunda directriz (verde) indica el protocolo a decodificar, en este caso HTTP. Las directrices en morado determinan los extremos de la comunicación, así como el puerto. La directriz '->' indica la dirección de la comunicación, así que estamos indicando a Suricata que cualquier paquete cuyo origen sea EXTERNO (que en la configuración es cualquier red que no sea HOME, es decir, la IP del servidor en este caso) de cualquier puerto con dirección a la red HOME a cualquier puerto sea uno de los condicionantes para disparar esta regla.
- **Opciones**: La directriz 'msg' en color oro indica el mensaje descriptivo de la alerta. Las directrices en negro recogen las restricciones propias del protocolo que activarán la alerta. En este caso se busca en el agente de usuario de la cabecera HTTP una coincidencia exacta del texto 'TEST SUSPICIOUS' ignorando mayúsculas y minúsculas. Las directrices en azul al final establecen el id de la regla (sid) y su n.º de revisión. La opción *classtype* es opcional y permite clasificar la regla en un grupo.

Se guarda la regla en el fichero local.rules y ejecutamos Suricata escuchando en la interfaz de pruebas y en otra consola hacemos un seguimiento de las últimas entradas del log de alertas. Adicionalmente también escuchamos todos los paquetes en la interfaz ens4 que llegan a la instancia de Suricata con *ngrep* en otra consola. Esto lo hacemos para verificar si los paquetes llegan al sensor pero no está generando alertas:

```
suricata ~# ngrep -d ens4 -q '.'
```

Para probar la regla, el servidor simula un servidor web escuchando en el puerto 80 mediante la herramienta *netcat*. El atacante ejecuta una petición GET HTTP especialmente preparada hacia el servidor, con el agente de usuario que Suricata debe asociar con la regla:

```
suricata ~# nc -l 80
atacante ~# curl -A "TEST SUSPICIOUS" 172.16.68.8:80
```

Resultado

La petición generada con *curl* no recibe ninguna respuesta porque netcat no devuelve nada por defecto, pero el servidor muestra la cabecera HTTP generada por el atacante con el agente de usuario "malicioso". Esto demuestra que el servidor ha recibido el paquete:

```
suricata ~# nc -l 80
GET / HTTP/1.1
Host: 172.16.68.8
User-Agent: TEST SUSPICIOUS
Accept: */*
```

El reenvío por *iptables* redirige una copia del paquete del atacante al sensor de Suricata y *ngrep* muestra el paquete recibido donde se puede ver la misma cabecera HTTP que recibe el servidor:

```
suricata:~# ngrep -d ens4 -q '.'
interface: ens4 (172.16.68.0/255.255.255.0)
filter: ((ip || ip6) || (vlan && (ip || ip6)))
match: .

T 172.16.68.6:59346 -> 172.16.68.8:80 [AP] #5
  GET / HTTP/1.1..Host: 172.16.68.8..User-Agent: TEST SUSPICIOUS..Accept:
  ↪  */*....
```

Sin embargo, para que Suricata detecte la alerta el servidor debe mandar una respuesta. Con *netcat* basta con escribir cualquier cadena de texto para enviar la respuesta (en un servidor web normal sería una respuesta HTTP). Suricata analiza la petición y detecta el agente de usuario de prueba, notificando la alerta en el fichero *fast.log*:

```
01/08/2021-11:01:14.007543  [**] [1:1:1] TEST SUSPICIOUS USER_AGENT [**]
↪  [Classification: A suspicious string was detected] [Priority: 3] {TCP}
↪  172.16.68.6:59346 -> 172.16.68.8:80
```

En *ngrep* aparece el paquete con el contenido de vuelta al atacante.

4.2 Evaluación de rendimiento

Las pruebas de evaluación inicial se han repetido para cada configuración de Suricata, con diferentes configuraciones de *multi-threading* para 8 y 16 hilos el máximo posible que se puede utilizar en el entorno de pruebas, ya que está limitado por otras aplicaciones alojadas en la instancia que sirven para realizar las pruebas. Se asume que para redes de alta velocidad 4 hilos no es suficiente y se omiten en las pruebas.

La prueba se realiza con tráfico TCP generado de forma aleatoria con la herramienta **iperf**. El tráfico consume el 100% del ancho de banda disponible repartidos en 100 sesiones TCP

concurrentes de 100 Mbps cada una, con un tamaño de paquete de 1500 bytes. Las pruebas se ejecutan durante 10 minutos. El comando utilizado es el siguiente:

```
atacante ~# iperf3 -c suricata -M 1500 -P 100 -b 100M -t 600
suricata ~# iperf3 -s
```

Como suricata se ejecuta en la misma instancia que el servidor de iperf, establecemos la afinidad del proceso del servidor para asegurarnos que no interviene en las medidas de rendimiento de Suricata.

```
suricata ~# taskset -cp 21,22,23 $(ps -ax | grep "iperf3 -s" | grep -v
↪  grep | cut -d' ' -f1)
```

Para llevar un mejor seguimiento de las estadísticas de stats.log se ha reducido su intervalo de actualización de 8 a 3 segundos en la configuración de Suricata.

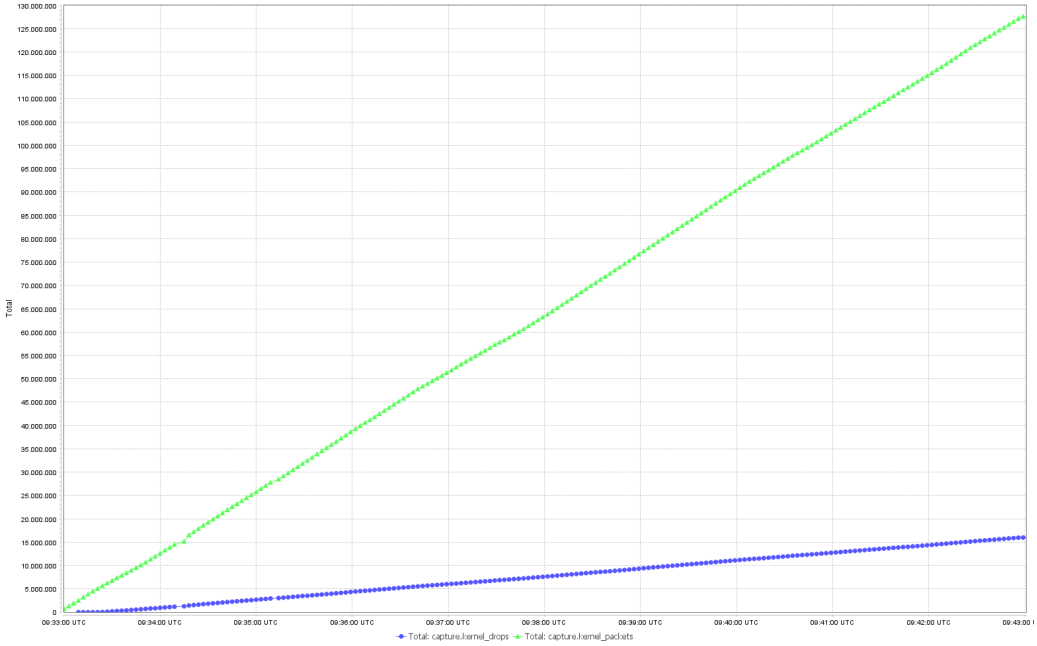
4.2.1 Resultado

La figura 4.1 muestra que el ratio de pérdida de paquetes es mayor con 16 que con 8 hilos. Un registro inusual teniendo en cuenta que es lo opuesto de lo esperado. Al final se ha determinado que ocurre por interferencia con otros procesos del sistema al mantener Suricata y el servidor de pruebas en la misma instancia, debido a la limitación del entorno de pruebas. La estadística registrada **app_layer.flow.failed_tcp** y **decoder.invalid** nos muestra que la gran mayoría del tráfico no ha sido analizado o no se ha podido analizar correctamente en profundidad por el motor de detección. Se generan alertas indicando de este hecho:

```
[**] [1:2260003:1] SURICATA Applayer Protocol detection skipped [**]
↪  [Classification: Generic Protocol Command Decode]
```

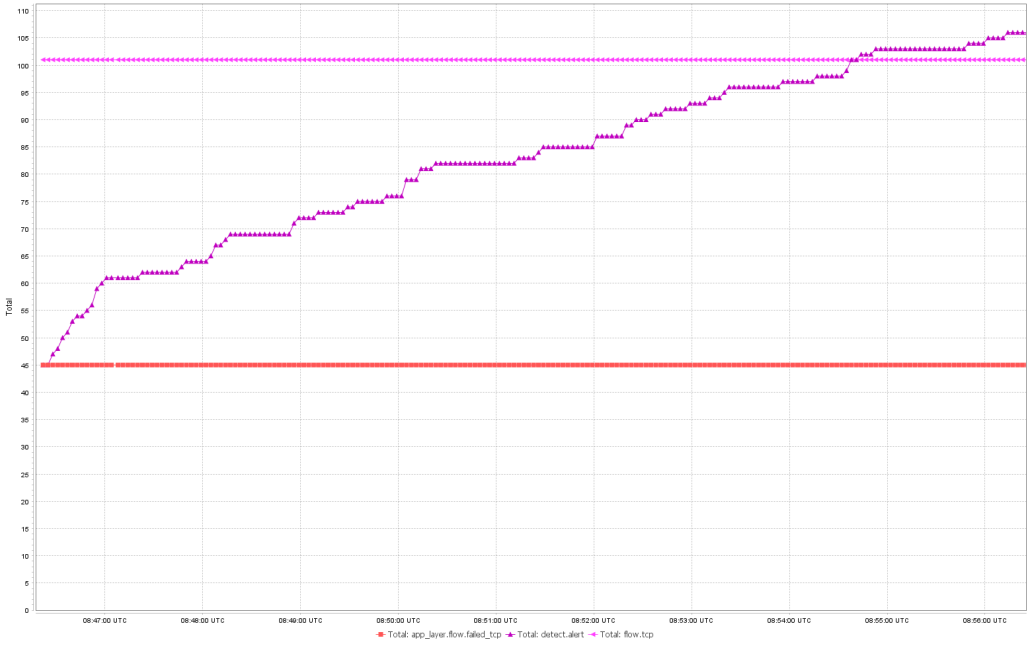
Lógicamente, al tratarse de tráfico TCP aleatorio limitado a la capa 4 Suricata no detecta protocolos de capas superiores que procesar. Llama la atención el número de alertas generadas. Se entiende que la alerta de aviso de análisis de protocolo evitado se genera por cada flujo TCP. Por lo tanto deberían existir 100 alertas de este tipo, pero el número es inferior en las pruebas. Esto sugiere que no se analizan el 100% de los flujos por falta de recursos y por lo tanto nos encontramos ante verdaderos positivos (en este caso, tráfico anómalo no identificable). En la figura 4.2 se muestran las estadísticas que se piensa confirman este hecho. Las estadísticas en concreto muestran el número total de flujos, representado por **flow.tcp** y cuales de esos flujos han sido procesados pero que Suricata no ha sido capaz de

(a) Suricata con 8 hilos.

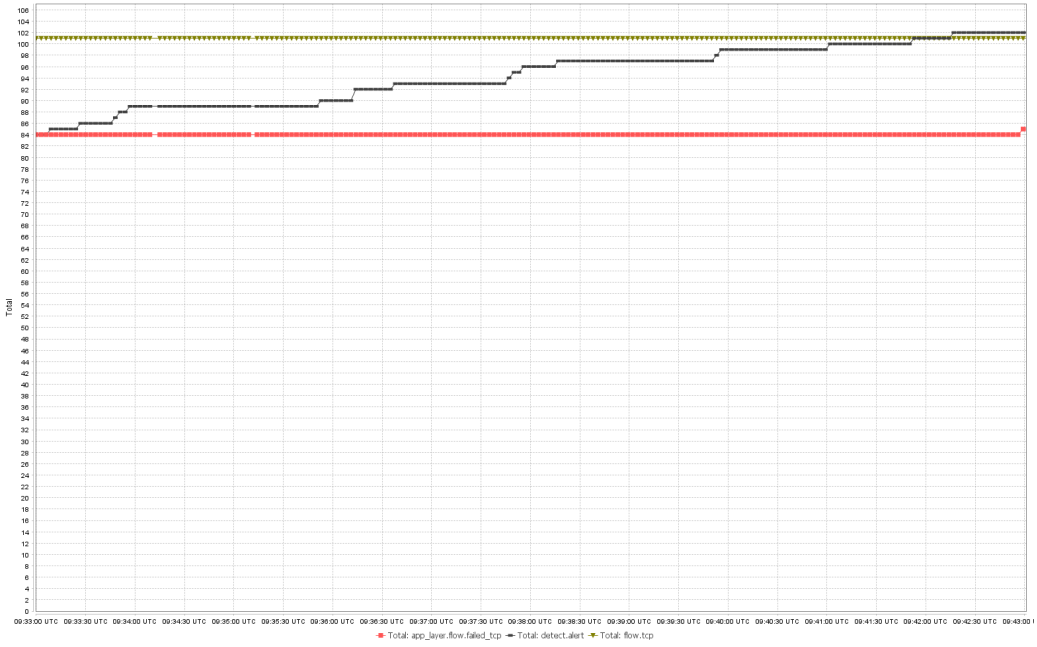


(b) Suricata con 16 hilos.

Figure 4.1: Comparación de paquetes capturados y paquetes perdidos con AF_PACKET por defecto. Los valores son acumulativos.



(a) Suricata con 8 hilos.



(b) Suricata con 16 hilos.

Figure 4.2: Comparación de flujos totales, flujos procesados por Suricata para la capa de aplicación (que en este caso determina que no es válido) y número total de alertas. Los valores son acumulativos.

detectar su protocolo, reflejado por **app_layer.flow.failed_tcp**. Como se ha comentado Suricata genera una alerta por cada uno de estos flujos, por lo tanto el número de alertas debería ser como máximo la cantidad de flujos capturados en tráfico, pero se generan otras alertas tal y como se ven en la tabla 4.2 para 8 hilos y 4.3 para 16. Estas alertas son falsos positivos generados por errores del decodificador de Suricata al no disponer de suficientes recursos de cálculo. El hecho de que la cantidad de estas alertas varíe con el número de hilos de proceso indica claramente que no es un problema del hardware subyacente. Un mayor número de hilos de procesamiento permite procesar más flujos, generará menos alertas y analizará una mayor superficie de trafico. La tabla 4.3, la configuración de 16 hilos presenta un ratio de falsos positivos mucho menor que con 8 hilos.

Alerta	Total
[1:2210044:2] SURICATA STREAM Packet with invalid timestamp	61
[1:2210044:2] SURICATA Applayer Protocol detection skipped	45

Table 4.2: Alertas generadas durante la prueba y número total de apariciones con 8 hilos.

Alerta	Total
[1:2210044:2] SURICATA Applayer Protocol detection skipped	84
[1:2210044:2] SURICATA STREAM Packet with invalid timestamp	18

Table 4.3: Alertas generadas durante la prueba y número total de apariciones con 16 hilos.

En cuanto a consumo de recursos, la carga media por CPU en 16 hilos es menor y está mejor repartida que en 8 hilos. El bajo uso de recursos puede explicarse porque realmente no hay análisis en la capa de aplicación.

4.2.2 Optimización de Suricata

Se repite la prueba con los siguientes ajustes para intentar reducir el ratio de pérdida de paquetes. Una menor pérdida de paquetes genera menos falsos positivos. Los resultados se comprueban solo con 16 hilos de procesamiento, ya que se ha demostrado que 8 no son suficientes para alcanzar un nivel óptimo. La nueva configuración consiste en habilitar el modo MMAP de AF_PACKET. Este modo permite al kernel copiar los paquetes recibidos directamente en el espacio de memoria del usuario en un *buffer* circular sin necesidad de esperar a eventos ni realizar llamadas de sistema para recoger los nuevos paquetes. Además de la característica MMAP también se activa tpacket-v3. Esto permitirá reducir el consumo de CPU en un 15-20% e incrementar el ratio de captura hasta un 20% aunque su uso no se recomienda con un IPS, pues causa latencia [43], afortunadamente no es el caso. El coste de utilizar estas mejoras supone un aumento del consumo de memoria RAM.

```
af-packet:
  - interface: ens4
    threads: auto
    cluster-type: cluster_flow
    defrag: yes
    use-mmap: yes
    tpacket-v3: yes
    ring-size: 32000
```

Con estas mejoras activas la pérdida de paquetes se reduce prácticamente a 0 durante toda la prueba. En la figura 4.3 la estadística **capture.kernel_drops** es 0. Se puede ver en la figura 4.4 como esta vez se han procesado correctamente todos los flujos y solo se han generado 100 alertas sin ningún falso positivo. Hay que tener en cuenta que aunque los resultados son buenos con iperf, no se asemeja a un tráfico real porque no se realiza procesamiento de protocolo más allá de TCP.

Figure 4.3: Suricata con 16 hilos y AF_PACKET con MMAP.

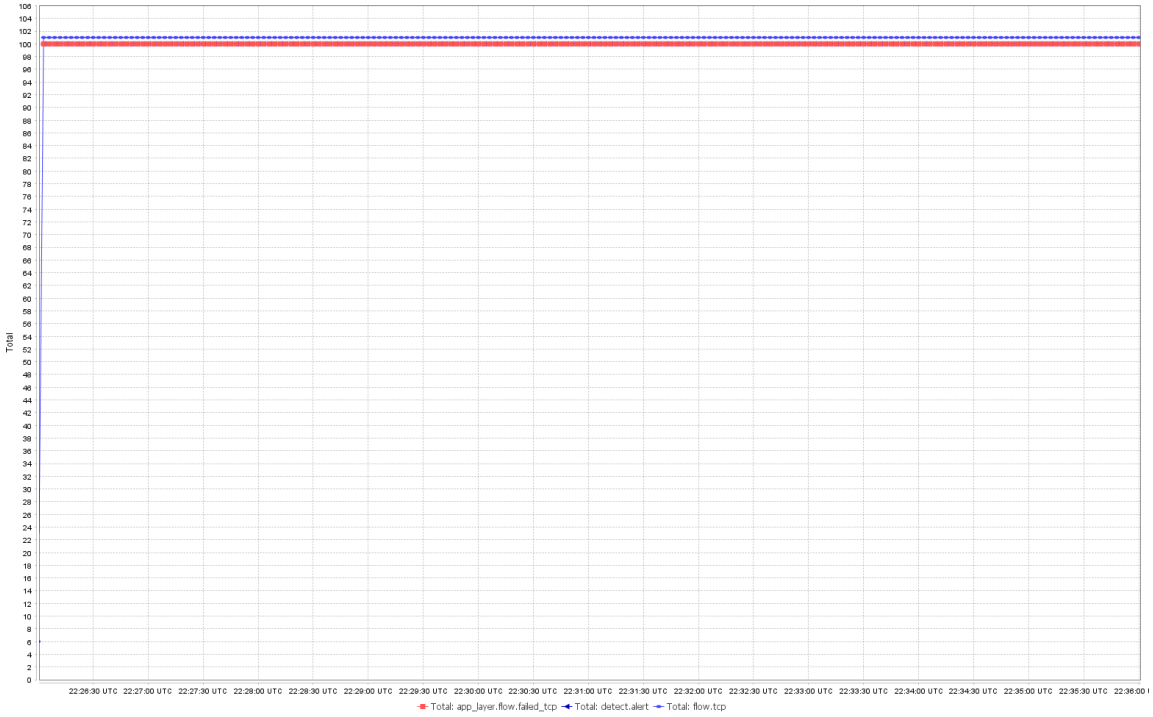


Figure 4.4: Suricata con 16 hilos y AF_PACKET con MMAP.

5. Conclusiones y trabajo futuro

En este trabajo se ha dado una visón global de los centros de procesamiento de datos, las amenazas y los riesgos de seguridad informática que les afectan. Se ha dejado claro las consecuencias de no detectar a tiempo una intrusión en un CPD y por ello se ha propuesto como una solución más a combinar con las existentes medidas de seguridad los sistemas de monitorización y detección de intrusiones. La tendencia actual de desplegar centros de procesamiento de datos en la nube agrava el problema de seguridad y hace más necesario que nunca diseñar un sistema de monitorización que permita controlar las actividades de grandes cantidades de usuarios de los que no se tiene ningún control sobre sus acciones.

El entorno de pruebas se ha visto limitado a la hora de desplegar la configuración deseada y la alternativa propuesta ha generado más limitaciones sobretodo de CPU, al tener que compartir el sensor y el servidor de pruebas en una misma instancia. Las herramientas utilizadas para automatizar la instalación y despliegue del entorno, aunque mínima, también añaden una ligera sobrecarga al sistema.

Las pruebas realizadas han dejado de manifiesto como la pérdida de paquetes puede afectar al output de alertas del sistema y la generación de falsos positivos, así como de falsos negativos, al menos con el caso de Suricata. El uso de múltiples hilos en en tornos de alto rendimiento se vuelve esencial y también el uso de las tecnologías adecuadas de captura y balanceo de carga de paquetes. Se han encontrado problemas para generar tráfico realista con los que poder poner a prueba el sistema de detección de intrusiones. En principio se había planeado realizar un último caso de prueba en el que se testeaba el entorno con tráfico real, abierto a internet. Pero por limitaciones de tiempo no pudo realizarse. Por esa razón se incluye como un objetivo futuro la realización de una prueba a mayor escala en un entorno con tráfico real, ya que el trafico simulado no es capaz de poner a prueba todas las capacidades del sistema de detección de intrusiones.

Los IDS que proporcionan recolección avanzada de información sobre los eventos aumentan las probabilidades de identificar un ataque asociado a un evento y el estado en el que se encuentra. Esto también significa que se genera un volumen de registros mucho mayor, por lo que se hace indispensable utilizar un sistema de recolección y gestión de registros.

Los IDS pueden resultar problemáticos si carecen de la precesión adecuada y generan alertas sobre eventos irrelevantes o no detectan anomalías relevantes, dando una falsa sensación de seguridad. Es importante optimizar y ajustar el sistema de detección de intrusiones en el

propio entorno. De la misma forma, es necesario revisar y analizar el funcionamiento del IDS periódicamente para reajustarlo a nuevas exigencias o cambios en el sistema. Por esta razón se contempla para trabajo futuro la inclusión de un sistema de auto-aprendizaje en el IDS que permita automatizar esta labor y aumente las capacidades de detección del mismo.

Entre los objetivos del trabajo futuro se propone mejorar la implementación actual añadiendo un IDS adicional con una metodología de detección basada en anomalías y combinarlo con Suricata con detección basada en reglas. Zeek es una buena alternativa que puede complementarse muy bien con Snort o Suricata. Este sistema aún podría mejorarse más añadiendo modelos de *Machine learning* para correlacionar las alertas de ambos IDS e identificar intrusiones emergentes.

También se propone como trabajo futuro integrar nuevas herramientas de recolección y post-analisis al entorno como elasticsearch o apache kafka, pues permitirá potenciar el desempeño de detección de anomalías, especialmente si se combina con las herramientas de post-análisis.

Bibliografía

[1] Statista, "U.s. data breaches and exposed records 2018," 2019. https://www.statista.com/statistics/273550/data-breaches-recorded-in-the-united-states-by-number-of-breaches-and-records-exposed; Online; Accedido el 12-11-2019.

[2] enisa, "Cloud security guide for smes." `https://www.enisa.europa.eu/publications/cloud-security-guide-for-smes`, 4 2015. Online; Accedido el 26-08-2019.

[3] Cisco, "What is a data center?," 2019. https://www.cisco.com/c/en/us/solutions/data-center-virtualization/what-is-a-data-center.html; Online; Accedido el 27-10-2019.

[4] D. Guardian, "The history of data breaches," 2019. https://digitalguardian.com/blog/history-data-breaches; Online; Accedido el 12-11-2019.

[5] A. K. Lab, "¿qué es el ransomware wannacry?." `https://www.kaspersky.es/resource-center/threats/ransomware-wannacry`, 2021. Online; Accedido el 30-06-2021.

[6] E. País, "Un ciberataque obliga a irlanda a cerrar el sistema informático de la sanidad pública," 05 2021. https://elpais.com/internacional/2021-05-14/un-ataque-cibernetico-en-irlanda-obliga-a-cerrar-el-sistema-informatico-de-la-sanidad-publica.html; Online; Accedido el 26-08-2019.

[7] N. I. of Standards and Technology, "Monitoring- glossary | csrc," 2021. https://csrc.nist.gov/glossary/term/monitoring; Online; Accedido el 12-03-2021.

[8] I. de Física de Cantabria, "Ifca | instituto de física de cantabria." `https://ifca.unican.es`, 2019. Online; Accedido el 26-12-2019.

[9] I. de Física de Cantabria, "Ifca cloud computing." `https://confluence.ifca.es/display/IC/Cloud+Computing`, 2019. Online; Accedido el 26-12-2019.

[10] P. A. Networks, "What is a datacenter?." `https://www.paloaltonetworks.com/cyberpedia/what-is-a-data-center`, 2021. Online; Accedido el 12-03-2021.

[11] A. Nordhoff, "What is a cluster? an overview of clustering in the cloud." `https://www.capitalone.com/tech/cloud/what-is-a-cluster/`, July 2020. Online; Accedido el 12-03-2021.

[12] Google, "¿qué es el cloud computing?." https://cloud.google.com/learn/what-is-cloud-computing, 2021. Online; Accedido el 27-06-2021.

[13] K. Rubin, "The advantages and disadvantages of traditional on-premises data centers vs. cloud computing," 2021. https://www.stratospherenetworks.com/blog/the-advantages-and-disadvantages-of-traditional-on-premises-data-centers-vs-cloud-computing/; Online; Accedido el 27-06-2021.

[14] N. I. of Standards and Technology, "The nist definition of cloud computing." https://csrc.nist.gov/publications/detail/sp/800-145/final, 2019. Online; Accedido el 15-09-2019.

[15] INCIBE, "Guía cloud computing." https://www.incibe.es/sites/default/files/contenidos/guias/doc/guia-cloud-computing_0.pdf, 2019. Online; Accedido el 26-08-2019.

[16] W. Stallings, L. Brown, M. D. Bauer, and A. K. Bhattacharjee, *Computer security: principles and practice. 3rd Edition.* Pearson Education Upper Saddle River (NJ, 2015.

[17] R. Banta, "Types of data center security." https://lifelinedatacenters.com/colocation/types-of-data-center-security, 12 2014. Online; Accedido el 26-08-2019.

[18] S. Axelsson, "Intrusion detection systems: A survey and taxonomy," tech. rep., Technical report, 2000.

[19] Amazon, "cyber-security-understanding-the-5-phases-of-intrusion." https://www.graylog.org/post/cyber-security-understanding-the-5-phases-of-intrusion, 2020. Online; Accedido el 22-02-2021.

[20] Rapid7, "Ms17-010 eternalblue smb remote windows kernel pool corruption." https://www.rapid7.com/db/modules/exploit/windows/smb/ms17_010_eternalblue/, 2021. Online; Accedido el 30-06-2021.

[21] Rapid7, "Denial-of-service attacks." https://www.rapid7.com/fundamentals/denial-of-service-attacks/, 2021. Online; Accedido el 27-06-2021.

[22] K. Scarfone and P. Mell, "Guide to intrusion detection and prevention systems (idps)," tech. rep., National Institute of Standards and Technology, 2012.

[23] M. Almgren, E. Lundin, and B. E. Jonsson, "Consolidation and evaluation of ids taxonomies," in *In Proceedings of the eighth Nordic Workshop on Secure IT systems (NordSec)*, 2003.

[24] H.-J. Liao, C.-H. R. Lin, Y.-C. Lin, and K.-Y. Tung, "Intrusion detection system: A comprehensive review," *Journal of Network and Computer Applications*, vol. 36, pp. 16–24, 2013.

[25] H. Debar, M. Dacier, and A. Wespi, "A revised taxonomy for intrusion-detection systems," in *Annales des télécommunications*, vol. 55, pp. 361–378, Springer, 2000.

[26] O. P. Team, "The world's most widely used host-based intrusion detection system." `https://www.ossec.net`, 2019. Online; Accedido el 27-06-2019.

[27] P. V. Rami Lehti, "Aide - advanced intrusion detection environment." `https://aide.github.io`, 02 2019. Online; Accedido el 27-06-2019.

[28] Cisco, "Página web oficial de snort." `https://www.snort.org`, 2021. Online; Accedido el 27-06-2021.

[29] OISF, "Suricata - open source ids / ips / nsm engine." `https://suricata-ids.org`, 2021. Online; Accedido el 27-06-2021.

[30] T. B. Project, "The zeek network security monitor." `https://www.zeek.org`, 2021. Online; Accedido el 27-06-2021.

[31] OpenStack, "What is openstack?." `https://www.openstack.org/software/`, 2020. Online; Accedido el 05-04-2021.

[32] Openstack, "Openstack compute (nova)." `https://docs.openstack.org/nova/latest/`, 2020. Online; Accedido el 05-04-2021.

[33] Openstack, "Neutron - openstack." `https://wiki.openstack.org/wiki/Neutron`, 2020. Online; Accedido el 05-04-2021.

[34] Docker, "Docker." `https://www.docker.com/`, 2021. Online; Accedido el 15-05-2021.

[35] Docker, "Docker-curriculum." `https://docker-curriculum.com/#introduction`, 2021. Online; Accedido el 15-05-2021.

[36] Docker, "Docker-compose." `https://docs.docker.com/compose/`, 2021. Online; Accedido el 15-05-2021.

[37] OpenStack, "Tap as a service (tapaas)." `https://docs.openstack.org/os-ken/latest/snort_integrate.html`, 2021. Online; Accedido el 27-06-2021.

[38] OISF, "Suricata user guide." `https://suricata.readthedocs.io/en/suricata-6.0.1`, 2021. Online; Accedido el 27-06-2021.

[39] A. L. D. Team, "Alpine linux." `https://www.alpinelinux.org/`, 2021. Online; Accedido el 15-06-2021.

[40] Danybr91, "Suricata docker image." `https://github.com/danybr91/suricata-docker`, 2021. Online; Accedido el 30-06-2021.

[41] A. Kleen, "packet(7) - linux manual page." `https://man7.org/linux/man-pages/man7/packet.7.html`, 2021. Online; Accedido el 30-06-2021.

[42] Docker, "Docker." https://docs.docker.com/network/, 2021. Online; Accedido el 15-05-2021.

[43] T. kernel development community, "Packet mmap - the linux kernel documentation." https://www.kernel.org/doc/html/latest/networking/packet_mmap.html, 2021. Online; Accedido el 30-06-2021.

A. Anexos

A.1 Configuración Docker

Fuente A.1: Dockerfile suricata

```
# Imagen base
FROM alpine:latest

ARG TZ

# Configurar el sistema
RUN apk add tzdata vim --no-cache && \
    cp /usr/share/zoneinfo/Europe/Madrid /etc/localtime && \
    echo "$TZ" >  /etc/timezone

# Dependencias del sistema
#https://pkgs.alpinelinux.org/package/edge/community/x86/suricata
RUN apk add suricata --no-cache

# Configurar suricata
COPY config/ /etc/suricata/

# Programar cron
COPY suricata-update.cron /etc/crontabs/suricata-update-cron
RUN chmod +x /etc/crontabs/suricata-update-cron
COPY suricata-update.sh /suricata-update-script.sh

# Activar el CTRL + C en el modo interactivo
STOPSIGNAL SIGINT

# Inicio del contenedor

COPY entrypoint.sh /
RUN chmod +x /entrypoint.sh
ENTRYPOINT [ "/entrypoint.sh" ]
CMD [ "--af-packet" ]
```

Dockerfile de suricata utilizado en las pruebas.

Fuente A.2: Docker-compose suricata

```
# https://docs.docker.com/compose/compose-file/compose-file-v2/
version: '2'

services:
# Suricata service
  suricata:
    container_name: suricata
    image: suricata:latest
    #Obtiene el valor desde el ENV de docker-compose
    command: "-i ens4 --af-packet"
    cpuset: "0,1,2,3,4,5,6,7"
    network_mode: "host"
    volumes:
      # Mount bind mode
      - '/data/suricata/etc:/etc/suricata'
      - '/data/suricata/rules:/var/lib/suricata'
      - '/data/logs/suricata:/var/log/suricata'
    cap_add:
      - NET_ADMIN
      - NET_RAW
    restart: unless-stopped
```

Docker-compose con la configuración de despliegue de Suricata. Está configurado para escuchar en la interfaz 'ens4' con la tecnología de captura AF_PACKET y 8 núcleos dedicados. El montaje de directorios es opcional y solo se usa para realizar con comodidad las pruebas. Sobreescribe la configuración base copiada cuando se compiló la imagen del contenedor. Para reiniciar el contenedor automáticamente junto con el host se utiliza la directiva *restart: unless-stopped*.

Fuente A.3: Script de actualización de Suricata

```
#!/bin/sh
/usr/bin/suricata-update && \
/usr/bin/suricatasc -c ruleset-reload-nonblocking
```

Script de actualización de Suricata.

Fuente A.4: Entrada cron de actualización de Suricata

```
0 0 * * * /bin/sh /suricata-update-script.sh >> /var/log/suricata/update.log 2>&1
```

Entrada cron para ejecutar el script de actualización (Ver: A.3) a diario y redirigir la salida normal y de errores aun fichero de log.

Fuente A.5: Entrada de ejecución del contenedor de Suricata

```
#!/bin/sh
# Update suricata rules (first time)
if [ ! -f /var/run/suricata.pid ]; then
    suricata-update
fi
# Start cron
crond
# Add cronjob
crontab /etc/crontabs/suricata-update-cron
# remove old pidfile if exists. Pidfile is required for logrotate
if [ -f /var/run/suricata.pid ]; then
    rm -f /var/run/suricata.pid
fi
# Start suricata (and pass extra arguments from CMD)
suricata -c /etc/suricata/suricata.yaml --pidfile /var/run/suricata.pid "$@"
```

Script de ejecución inicial del contenedor de suricata. Actualiza las reglas sólo si se ejecuta por primera vez y programa su ejecución periódica. Redirige los parámetros dados por CMD a suricata directamente. Para indicar que el proceso de suricata se está ejecutando crea un fichero pid file que dura durante la ejecución del contenedor. Debe borrarse antes de volver a lanzar suricata para evitar errores.

A.2 Suricata

Stat	Descripción
capture.kernel_packets	Cantidad de paquetes capturados por el motor de suricata en total. Sin decodificar aún.
capture.kernel_drops	Indica la cantidad de paquetes descartados por suricata por no tener suficiente velocidad de proceso. Este stqat debe ser analizado conjuntamente con el stat "capture.kernel_packets". Idealmente, el valor debe mantenerse en un 1% de "capture.kernel_packets". También hay que tener en cuenta las pérdidas ocasionadas por el propio hardware. Podemos verlo con ethtool -S.
decoder.pkts	Cantidad de paquetes decodificados por suricata.
decoder.invalid	Cantidad de paquetes que han tenido error al decodificar.
tcp.rst	Cantidad de paquetes TCP con el flag RST activado.
flow.wrk.spare_sync	Cantidad de hilos utilizados para analizar flujos.

Table A.1: Estadísticas de Suricata relevantes para las pruebas