Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

DISEÑO DE UN SISTEMA DE SEGURIDAD Y MONITORIZACIÓN PARA REDES DEFINIDAS POR SOFTWARE

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Antonio José Caballero Morcillo

Tutor: Mª Lourdes Peñalver Herrero

Co-Tutor: Jaime Lloret Mauri

2019-2020

Resumen

En este trabajo se aborda la creación de un sistema de seguridad y monitorización de red basado en las Redes Definidas por Software (SDN) o Redes Programables debido a su gran flexibilidad y a la configurabilidad que ofrece su elemento principal, el controlador. Para ello se ha realizado un estudio de la historia de estas redes y su funcionamiento, así como un análisis de los desafíos en materia de seguridad a los que se enfrenta este modelo.

El sistema que se presenta como solución es la combinación de dos elementos principales, el controlador Faucet junto a Snort, un sistema de detección de intrusiones (IDS). A estos se les suman las herramientas necesarias para la recogida de estadísticas y su visualización Gauge, Prometheus y Grafana, así como una base de datos MySQL donde almacenar las alertas de Snort. El resultando es un completo sistema de seguridad capaz de reaccionar ante aquellos eventos sospechoso sin afectar al funcionamiento mismo de la red preservando en todo momento la calidad de servicio y la integridad de este.

Palabras clave: SDN, Faucet, Snort, IDS, seguridad, monitorización.

Abstract

This project focuses on the creation of a security and network monitoring system based on Software Defined Networks (SDN) or Programmable Networks due to its great flexibility and the configurability offered by its main element, the controller. To this end, a study of the history of these networks and their performance has been carried out, as well as an analysis of the security challenges faced by this model.

The system presented as a solution is the combination of two main elements, the Faucet controller together with Snort, an intrusion detection system (IDS). In addition to these, there are the necessary tools for the collection of statistics and their visualization such as Gauge, Prometheus and Grafana, as well as a MySQL database where Snort's alerts are stored. The result is a complete security system capable of reacting to suspicious events without affecting the actual performance of the network, preserving always the quality of service and its integrity.

Keywords: SDN, Faucet, Snort, IDS, security, monitoring.

Tabla de contenidos

Índice de figuras y tablas

1. Introducción

Las Redes Definidas por Software es una tecnología en crecimiento exponencial y cuyo uso no se limita ya a grandes empresas y centros de datos si no que, gracias a sus múltiples posibilidades y a la gran cantidad de proyectos de código abierto creados por la comunidad, cada vez son más las pequeñas empresas o usuarios los que se están beneficiando de esta tecnología. Según la consultora Gartner, para 2022 se espera que el 75% de los datos generados por las empresas se creen y procesen fuera de un centro de datos centralizado tradicional. [1]

Una encuesta realizada por Juniper a 500 responsables de toma de decisiones en materia de Tecnologías de la Información reveló que el 98% dijera que ya estaban usando o considerando una implementación de SDN. Este interés por las Redes Definidas por Software está claramente justificado ya que el International Data Corporation (IDC) estima que el mercado mundial de SDN de centros de datos crecerá hasta valer más de 12.000 millones de dólares en 2022. [2]

Su característica principal, la separación del plano de control del plano de datos, hace más eficiente el transporte de información como el control de la red y de los distintos servicios que la componen. Gracias a esto, los arquitectos, operadores de redes y seguridad y también programadores, podrán ir más allá del enfoque rígido y a veces inseguro de “las redes definidas por hardware”. [3]

Separar los planos de control y datos es la forma más común de pensar en lo que es el SDN, pero es mucho más que eso, dijo Mike Capuano, director de marketing de Pluribus. [4] "En su corazón, SDN tiene una entidad inteligente centralizada o distribuida la cual posee una visión completa de la red, que puede tomar decisiones de enrutamiento y conmutación basadas en esa visión (…) Normalmente, los enrutadores y conmutadores de la red sólo conocen a su “vecino” en la red. Pero con un entorno SDN correctamente configurado, esa entidad central puede controlarlo todo, desde cambiar fácilmente las políticas hasta simplificar la configuración y la automatización en toda la empresa".

Esta virtualización de las redes convencionales está más cerca de todos nosotros, el gigante Vodafone anunció recientemente su transición hacia un modelo virtualizado debido a la flexibilidad y agilidad de estos; otro ejemplo mucho más claro, la empresa VMware la cual proporciona software de virtualización, desde Máquinas Virtuales a Centros de Datos virtualizados, es uno de los mayores exponentes del uso de las SDN en la actualidad. [4]

1.1. Motivación y contexto.

Introducidos ya algunos de los puntos más fuertes de las Redes Definidas por Software, cabe destacar uno de los aspectos de mayor importancia y que muchas veces se ve de alguna manera ignorado en detrimento de la búsqueda de la eficiencia y la funcionalidad. Este es la seguridad. En esta línea se percibe a modo personal, como uno de los campos más interesantes y cruciales en el cual, resulta requisito indispensable la formación de nuevos especialistas junto a la motivación de profesionales para desarrollar sus investigaciones relacionadas con este ámbito.

Estas inquietudes no nacen sin contexto, pues proceso similar sufrieron las redes tradicionales cuando se diseñaron los primeros modelos y protocolos, donde se buscó que fueran funcionales, cada vez más rápidos y que ofreciesen más servicios. Esta evolución basada en la necesidad presenta el problema de que, aunque resulte adaptativa, puede fallar en elementos tan significativos como el de la seguridad. [5]

La flexibilidad que caracteriza a las SDN ha sido uno de los grandes motivadores para la realización del presente trabajo. En este contexto, nos resultó interesante la posibilidad de que, mediante el análisis del tráfico entrante en una red y buscando preservar la seguridad del sistema sin afectar a la calidad del servicio o incidiendo en ella positivamente, se pueda modificar el funcionamiento de este de una forma estática y "manual" (con los datos otorgados por el sistema de detección de intrusiones) como se propone en este proyecto; o de manera más ambiciosa, de una forma automatizada mediante un motor de decisión.

Así pues, se presenta una tecnología nueva con un gran potencial pero que plantea nuevos retos a nivel de protección. Por esto, se ha decidido realizar este trabajo sobre el estudio, desde el punto de vista de la seguridad, de las Redes Definidas por Software para así poder posteriormente presentar el diseño de un sistema sencillo pero seguro basado en esta tecnología en combinación con otra muy popular en sistemas de seguridad como es un sistema de detección de intrusiones (IDS) mediante el software Snort.

1.2. Objetivos.

El objetivo principal de este trabajo es el análisis de la situación actual de las Redes Definidas por Software desde el prisma de la seguridad para el posterior diseño e implementación de una arquitectura que combine tanto seguridad y monitorización mediante el uso de esta tecnología en composición con un sistema de detección de intrusiones utilizando el software Snort. Para ello se tienen en cuenta los siguientes subobjetivos:

- Análisis de los distintos controladores SDN más populares.
- Repaso al protocolo OpenFlow, estándar en este tipo de redes.
- Definición de los distintos tipos de sistemas de detección de intrusiones.
- Diseño de la solución y exposición de los distintos elementos que la componen.
- Implementación de la solución y validación del funcionamiento.

1.3. Estructura de la memoria.

A continuación, se expone la forma en la que se ha estructurado la presente memoria:

- **Capítulo 1 Introducción**. Se trata del actual punto, en él se presenta el proyecto, introduciendo a modo de contexto algunos de los elementos de los que se hablará más adelante, así como exponiendo las motivaciones y objetivos del trabajo.
- **Capítulo 2**: **Estado del arte.** Este capítulo es el más extenso de todos, en él se explican detalladamente el estado actual de las dos tecnologías principales que se utilizan en el desarrollo del proyecto, las redes definidas por software y los sistemas de detección de intrusiones, para ello se presentan su evolución histórica y los detalles sobre el funcionamiento en cada una de ellas. Se concluye presentado la propuesta para el desarrollo de la solución final.
- **Capítulo 3**: **Análisis del problema.** En este capítulo se lleva a cabo un análisis del problema al que se considera que se puede acercar una solución mediante la implementación de una solución en función de los intereses y necesidades estudiadas.
- **Capítulo 4**: **Diseño de la solución.** Este punto del trabajo es donde se expone la arquitectura diseñada para su posterior implementación con el fin de dar solución a lo estudiado en apartados previos. Se exponen también los motivos por los cuales se han elegido los distintos elementos de la arquitectura diseñada como Faucet y Snort.
- **Capítulo 5**: **Implementación.** En este capítulo se realiza la implementación de la arquitectura diseñada, se muestran los problemas que han podido resultar con respecto a lo diseñado teóricamente y se termina presentado el sistema que será el utilizado para las posteriores pruebas de funcionamiento.
- **Capítulo 6**: **Pruebas realizadas.** Es el punto en el que se han llevado a cabo las pruebas de funcionamiento para demostrar las capacidades que ofrece el sistema diseñado, se han realizado tres pruebas asociadas a ataques de denegación de servicio por ser los más dañinos contra una red definida por software.
- **Capítulo 7**: **Conclusiones.** Tras la realización de las pruebas, en este capítulo se extraen las conclusiones obtenidas de las pruebas realizadas y además se presentan otras de las conclusiones a las que se han llegado después de terminar todo el proceso de estudio, análisis y pruebas del proyecto.
- **Capítulo 8**: **Trabajos futuros.** Este es uno de los puntos más interesantes del presente proyecto ya que sin duda quedan muchas puertas abiertas a trabajos futuros y caminos a seguir muy prometedores y ambiciosos.
- **Capítulo 9**: **Referencias.** Se trata de todas las fuentes consultadas durante la realización del proyecto.
- **Capítulo 10**. **Anexos.** Se muestran los procesos de instalación, configuración y desarrollo de algunas de las herramientas del trabajo que no han sido incluidos en el documento por cuestiones de extensión y síntesis.

2. Estado del arte

2.1. Redes Definidas por Software.

El trabajo previo de María Gómez Climent [6], antigua alumna de la Universidad Politécnica y de la ETSINF; quien ya habló sobre las Redes Definidas por Software bajo la misma tutorización que el presente en el curso 2015-2016, dejó explicado a la perfección el nacimiento de esta tecnología a través de una reseña histórica, así como sus principales características y arquitectura. A modo de recordatorio y para contextualizar el resto de este documento se exponen las principales etapas históricas por las que han pasado las SDN hasta ser como se conocen en la actualidad.

El comienzo de esta tecnología se debe situar en 2008 cuando, tras varios años de investigaciones en las Universidades de Standford y Berkeley, se publican en la ACM SIGCOMM, considerada la conferencia líder en comunicaciones de datos y redes en el mundo, dos artículos en la edición de ese año. Por un lado, se publica un artículo llamado "OpenFlow: enabling innovation in campus networks" [7] el cual sentaría las bases para el posterior nacimiento del protocolo OpenFlow del que se hablará más adelante. Por otro lado, el segundo de los artículos se trató de "NOX: Towards an Operating System for Networks" [8] que se basaría en el anterior sobre OpenFlow para introducir lo que sería el primer controlador SDN.

Las razones por las que surgió esta tecnología se pueden resumir en [9]:

- Las redes tradicionales son difíciles de manejar. A medida que estas crecían también se incrementaba la complejidad de su manejo. Cada vez se necesitan redes más flexibles y programables.
- La evolución de las redes tradicionales era costosa. Tanto a nivel económico como a nivel tecnológico debido a que su mejora depende en gran medida de la mejora del hardware de sus componentes y en este proceso entran otros factores como los materiales utilizados, etc.
- La seguridad y el control en redes requiere gran complejidad para implementarla en las redes tradicionales, listas de control de acceso, VLANs, Middleboxes… Su integración es posible, pero supone un coste considerado.

Los años posteriores servirían para crear lo que para cualquier tipo de tecnología es fundamental, unos cimientos teóricos sobre los cuales se podrá progresar cada vez más. Este proceso comienza con la publicación por parte de la Universidad de Stanford de la versión V1.0.0 del protocolo de enrutamiento OpenFlow en 2009 y que en 2010 se convertiría en el estándar para las redes definidas por software gracias a la creación de la Open Networking

Foundation[1] con 69 miembros entre los que se destacan grandes empresas como Google, Yahoo… y a los que más adelante se unirían Cisco, Juniper, HP, etc.

Esta estandarización haría que las SDN fueran mucho más accesibles para cualquiera y, su carácter dirigido a los usuarios y sin ánimo de lucro fueron los ingredientes perfectos para crear una comunidad de desarrolladores y proyectos de código abierto que popularizarían esta tecnología. Muestra de ello es la compra por parte de la compañía VMware de la empresa Nicira, creada por uno de los precursores de las SDN Martin Casado, por un valor de 1.26 billones de dólares en 2012.

Sin embargo, la línea de progreso en la que las SDN y OpenFlow avanzaban juntas se vería ramificada cuando, en 2014 la empresa Cisco anunciaría un nuevo protocolo de enrutamiento llamado OpFlex [10]. Así tras el anuncio de OpFlex, se pueden considerar dos enfoques principales para el plano de control [11] de un entorno virtualizado:

- Imperativo: basado en un controlador SDN centralizado que actuará como el cerebro del entorno. Este se comunica con las aplicaciones gracias a una interfaz de programa (API) y será el encargado de dictaminar el comportamiento al resto de componentes de la red (routers, switches, etc) a través del plano de datos para que estos puedan satisfacer las necesidades de la aplicación. Este enfoque es el utilizado por OpenFlow. Sin embargo, plantea un posible cuello de botella en el controlador ya que, si este falla, falla todo el sistema.
- Declarativo: describe un modelo en el que el controlador SDN declara lo que la aplicación necesita y envía ese mensaje al tejido de la red para que los routers y switches determinen cómo cumplir los requisitos de la aplicación. Un plano de control declarativo permite una inteligencia más distribuida; establece una política central, pero da poder a los nodos de la red para tomar más decisiones sobre cómo ejecutar dichas políticas. Este es el enfoque que plantea OpFlex.

Pese a este nuevo protocolo, OpenFlow no perdería popularidad; su uso y, por ende, el de las Redes Definidas por Software seguiría creciendo los siguientes años. Así se demostró en 2017 cuando Google anuncia que comenzarán a asociarse con los principales operadores de redes móviles, construyendo una plataforma basada en SDN para que los operadores ejecuten sus servicios de red [12].

El progreso ha seguido la misma tendencia hasta nuestros días cómo se habló en la introducción de este documento, y todo apunta a que seguirá incrementándose su popularidad y su uso en los siguientes años tras los anuncios de compañías como Vodafone [13], Movistar [14] que ya ofrecen algunos de estos servicios a sus clientes.

[1] https://www.opennetworking.org

Presentada ya la historia de las Redes Definidas por Software y los principales acontecimientos que han marcado su evolución, es pertinente explicar su arquitectura y cómo difiere esta de las redes tradicionales. Dado que el objeto de este trabajo no es una revisión teórica sobre las SDN, se tratará de ceñirse a los aspectos más relevantes.

2.1.1. Arquitectura.

Como es bien sabido, la principal característica de las Redes Definidas por Software es la separación entre el plano de control y el plano de datos pero, ¿qué significa esto a efectos prácticos? Antes de responder a esta pregunta, es pertinente explicar los tres planos o capas en los que se divide una Red Definida por Software:

-**Plano de control.** Es el plano intermedio y el cerebro del conjunto de la red. Su misión es la de gestionar la comunicación entre el plano de datos y el de aplicación dotando de lógica al sistema y haciendo posible el correcto enrutamiento de los paquetes a través de la red. Permite además, llevar a cabo todas estas acciones de manera centralizada y totalmente programable. El elemento principal de este plano es el Controlador, el cual como se indicará más adelante es un elemento de software.

-**Plano de datos.** Este plano está compuesto normalmente por dispositivos físicos (routers, switches, etc). Sin embargo, los elementos que forman este plano pueden ser elementos virtuales (Virtual Switches) basados únicamente en software. La función de este plano se limita a realizar tránsito de datos, dado que la lógica queda implementada en el plano de control. La forma de comunicarse con el controlador es mediante una interfaz sur o "*SouthBound"* que usa, de forma estándar, el protocolo OpenFlow que se explicará con más detalle en apartados posteriores.

-**Plano de Aplicación.** En este plano se crean aplicaciones para controlar la red, que mediante una interfaz norte o "*NorthBound*", se comunican con el controlador que realiza las pertinentes operaciones sobre los dispositivos del plano de datos. Es frecuente que se englobe la capa de aplicación y la de control en un mismo plano debido a que ambas se componen de elementos programables y también a causa de que, a diferencia de la interfaz sur o "Southbound" que hace posible la comunicación entre la capa de datos y la de control existe un estándar, el protocolo OpenFlow; para la correspondiente a la situada entre esta capa de aplicación con la de control, no existe algo parececido.

Se presentan para ilustrar la anterior explicación dos ilustraciones (Ilustración 1 y Ilustración 2), donde se pueden observar la distribución de los distintos planos y qué elementos comoponen cada uno de ellos.

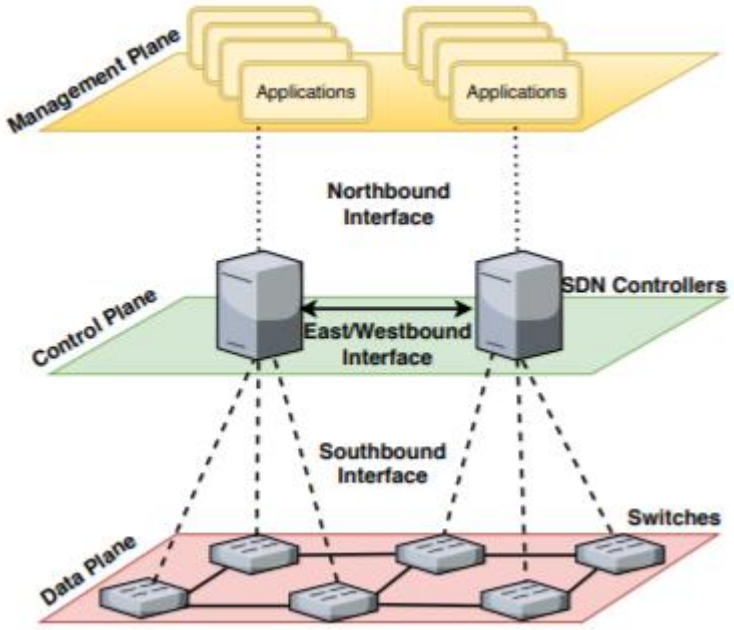


Ilustración 1. Arquitectura 1 SDN [15]

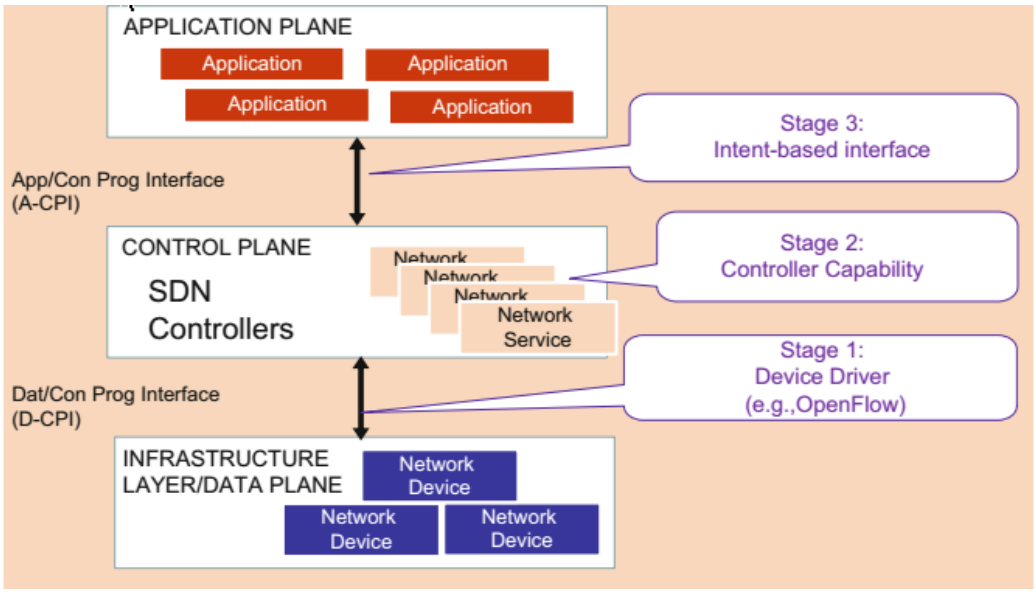


Ilustración 2. Arquitectura 2 SDN [16]

Expuestas las definiciones de las distintas capas de las SDN, se considera necesario explicar el motivo de la independencia entre ellas. Varias son las razones por las que la separación del plano de control del de datos es interesante:

La primera de ellas, el software de control de la red puede evolucionar independientemente del hardware de la misma. Esto quiere decir, que los elementos que componen el plano de datos (routers, switches, etc) pueden ser sustituidos en cualquier momento sin tener que preocuparnos porque la red quede en un estado inconsistente. Por otro lado, también permite que se puedan mejorar y gestionar de una forma más económica y menos compleja el funcionamiento de la red ya que, dado el crecimiento actual y las necesidades futuras comentadas en anteriores apartados, se presenta inviable un modelo donde sean necesarias las configuraciones manuales en sistemas hardware específicos y el mantenimiento de infraestructuras físicas [17].

Otra de las razones es que, gracias a su carácter centralizado donde se presenta al Controlador como el cerebro de toda la red, es posible tener una vista completa de la estructura de esta y también una mejor depuración y análisis del comportamiento del conjunto del sistema. Muchos controladores actuales incluyen un recolector de estadísticas de la red que permiten mediante interfaces muy visuales ver el estado general del sistema. Por otro lado, gracias a la programabilidad de estos controladores y a que muchos de ellos se tratan de proyectos de código abierto como veremos más adelante, las posibilidades de integración junto a otras aplicaciones son casi ilimitadas [18].

2.1.2. OpenFlow.

Como se ha comentado previamente, el protocolo OpenFlow se estableció como el estándar de comunicación entre el plano de control y el de datos en una red definida por software. Es el encargado de transmitir las instrucciones o directrices programadas en el controlador hasta los elementos de la capa de infraestructura en forma de paquetes para conseguir que estos se comporten de la forma deseada. Cabe destacar, que los dispositivos de la capa de datos deben ser compatibles con este protocolo. Ejemplo de ello son los OpenFlow switch (ilustración 3 e ilustración 4).

Las instrucciones de enrutamiento se basan en un flujo, que consiste en que todos los paquetes comparten un conjunto de características comunes [18]. Se pueden especificar una gran variedad de parámetros para definir un flujo. Entre los posibles criterios se incluyen: el puerto del switch donde llegó el paquete, el puerto Ethernet de origen, el puerto IP de origen, la etiqueta VLAN, el puerto Ethernet o IP de destino y otras características del paquete. El controlador especifica al switch el conjunto de parámetros que definen cada flujo y cómo deben procesarse los paquetes que coinciden con el flujo.

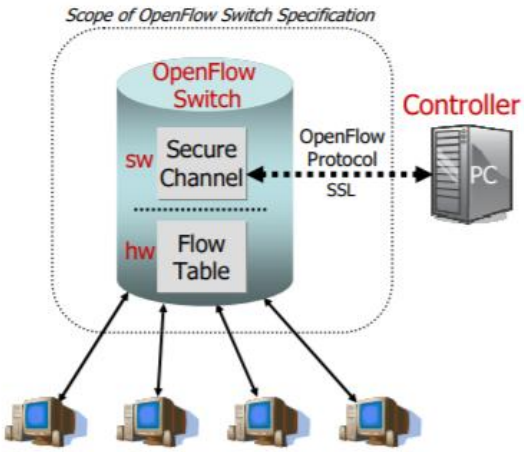


Ilustración 3. Switch OpenFlow[19]

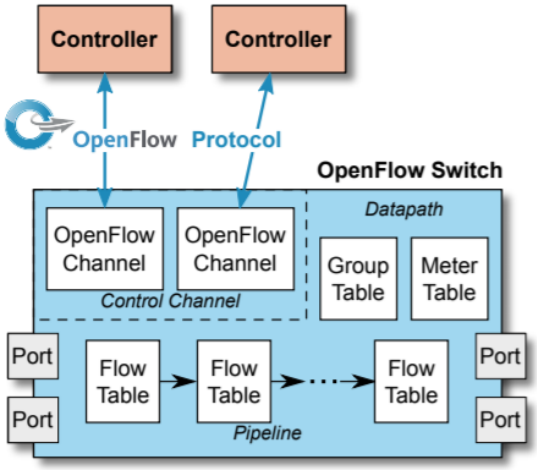


Ilustración 4. Vista interna switch OpenFlow [20]

Cada switch mantiene un número de tablas de flujo, y cada tabla contiene una lista de entradas de flujo. Cada entrada de flujo contiene un campo de coincidencia que define el flujo, un contador y un conjunto de instrucciones. Las entradas en el campo de coincidencia contienen un valor específico contra el cual se compara el parámetro correspondiente en el paquete entrante o un valor que indica que la entrada no está incluida en el conjunto de parámetros de este flujo. La Ilustración 5 muestra el proceso que sufre cada paquete desde que entra hasta que es redirigido o descartado.

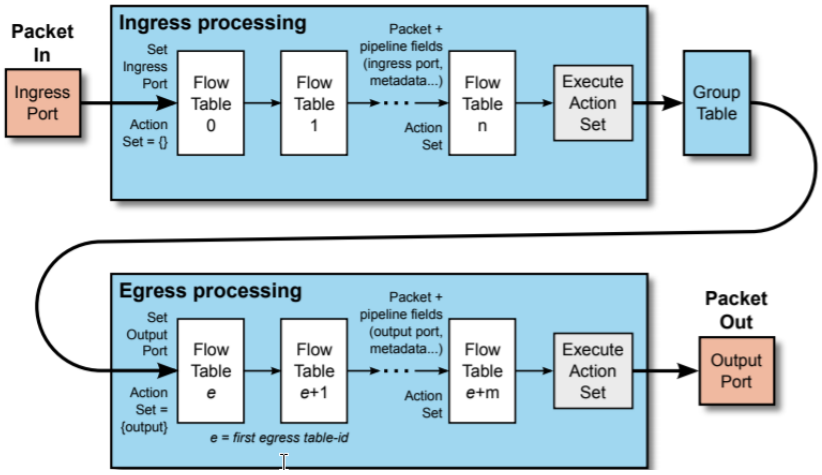


Ilustración 5. Transcurso del paquete

Sin embargo, pese a su extendido uso es necesario remarcar que OpenFlow no es sinónimo de Redes Definidas por Software [21]. Existen otros protocolos independientes a OpenFlow que surgen como alternativa para la comunicación en el Southbound de una SDN. Algunos de los más importantes son:

- ForCES [22]: FORwarding & Control Element Separation, estandarizado por el Grupo de Trabajo de Ingeniería de Internet, IETF[2] por sus siglas en inglés. Surge como alternativa a OpenFlow y plantea un modelo de separación lógica entre elemento de control y elemento de reenvío o de datos al igual que OpenFlow con la diferencia de que esta separación no afecta a la lógica y se mantiene en el mismo elemento físico por lo que es compatible con elementos de red tradicionales.
- OpFlex [23]: este protocolo impulsado por la empresa Cisco ha sido previamente comentado debido a que supuso un nuevo enfoque del plano de control hasta entonces Imperativo y que tras la salida de OpFlex se introdujo el enfoque Declarativo del mismo.
- NetConf [24]: desarrollado también por el IETF junto a Juniper Networks, este protocolo NETCONF proporciona mecanismos para instalar, manipular y eliminar la configuración de dispositivos de red. Sus operaciones se ejecutan sobre una capa sencilla de Llamada de Procedimiento Remota (RPC). Se puede separar en cuatro capas: contenido, operaciones, mensaje y transporte. Esta propuesta existía antes de SDN y al igual que OpenFlow también proporciona una API sencilla. Esta API puede ser utilizada por las aplicaciones para enviar y recibir conjuntos de datos de configuración completos o parciales.

[2] https://www.ietf.org

A continuación, se puede ver una tabla comparativa de estas tres alternativas a OpenFlow.

Alternativa	Objetivo	Solución	Beneficio
ForCES	Separación lógica entre elementos de control y de reenvío.	La lógica no cambia	Permite la separación del plano de control y el de datos utilizando elementos de red tradicionales
OpFlex	Reducción de la complejidad y mejora de la escalabilidad	Modelo de políticas declarativo	Mejora de la escalabilidad
NetConf	Reducir la complejidad y mejorar el rendimiento	Llamada de Procedimiento Remota (RPC)	Cercano al funcionamiento tradicional del switch, pero reduciendo costes.

Tabla 1. Alternativas a OpenFlow

2.1.3. Controladores.

El controlador es, sin duda, la parte más importante de una SDN y al igual que el resto de los elementos, ha sufrido una gran evolución desde sus inicios en 2008 con la presentación de NOX previamente mencionada.

El hecho de ser el "cerebro" de la red hace que las diferentes líneas de investigación se hayan centrado en gran medida en la mejora de este componente. Las distintas alternativas que se pueden encontrar en la actualidad difieren entre ellas principalmente en:

- El lenguaje de programación utilizado.
- Su arquitectura (centralizados o distribuidos).
- Las interfaces de programación de aplicaciones (API) que proporcionan (Northbound, Southbound y Eastbound).
- La compatibilidad con determinado sistema operativo.
- Si permiten paralelismo y modularidad.

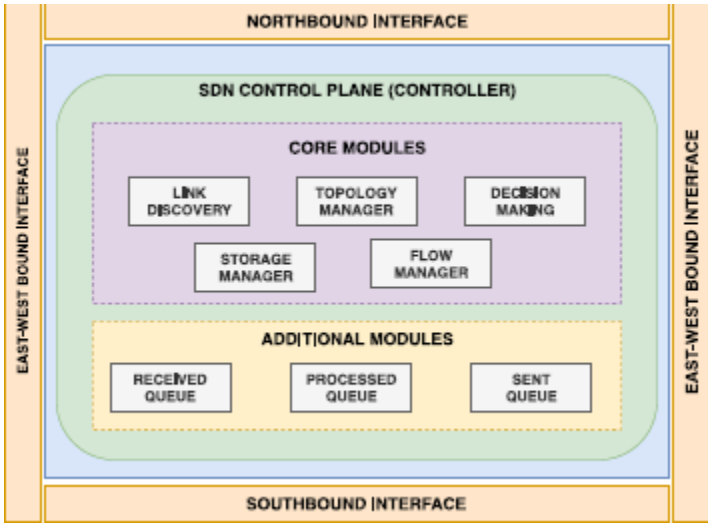


Ilustración 6. Arquitectura interna controlador SDN [25]

Se ha optado por comentar los principales controladores en base a su popularidad y la cantidad de documentación sobre cada uno de ellos disponible. Estos son:

- NOX [26]: es el primer controlador que surgió y su desarrollo se llevó a cabo paralelamente al de OpenFlow en los primeros años. Proporciona una API OpenFlow 1.0 basada en el lenguaje de programación C++. Es un controlador ligero y ágil caracterizado por ser asíncrono. Su uso se limita a algunas distribuciones de Linux.
- POX [27]: es un controlador desarrollado a partir de NOX para cubrir algunas de las necesidades de las SDN no cubiertas por este otro, usando para ello Python en lugar de C++. Su uso se extiende a sistemas Windows y Mac.
- Floodlight [28]: se trata de un controlador de código abierto compatible con OpenFlow, está escrito en Java. El desarrollo de Floodlight fue llevado a cabo por la compañía Big Switch Network y en principio formaría parte del proyecto OpenDaylight, sin embargo, las diferencias entre Big Switch y Cisco harían que tomarán distintos caminos y formaran proyectos independientes. Se diferencia de los anteriores, aparte de en el lenguaje de programación, en que es el único que cuenta con una API Rest para la interfaz Northbound haciendo posible la integración de aplicaciones externas.
- OpenDaylight [29]: este proyecto impulsado por la Linux Foundation[3] es, según su página oficial, el controlador SDN de código abierto más implementado en la actualidad. Al igual que Floodlight, está escrito en Java pero, a diferencia del resto, se trata de un controlador descentralizado. Cuenta con una interfaz Eastbound (Ilustración 1) para posibilitar la comunicación entre los distintos módulos de controlador, en conjunción con las otras dos interfaces Norte-Sur.
- Open Network Operating System (ONOS) [30]: a priori, este controlador es casi idéntico a OpenDaylight, ambos están escritos en Java y diseñados para un uso modular con una infraestructura personalizable, es importante señalar que cada socio del ONOS es también un miembro del ODL. Más allá de eso, hay algunas características clave que los diferencian. Por ejemplo, ONOS se centra más en aspectos de rendimiento y agrupación para aumentar la disponibilidad y la escalabilidad. Como resultado, ONOS se enfoca más en las redes "Carrier-Grade" [31] y los operadores de telecomunicaciones están involucrados en sus proyectos.
- RYU [32]: este controlador proporciona componentes de software con API bien definidas que facilitan a los desarrolladores la creación de nuevas aplicaciones de gestión y control de redes. Ryu soporta varios protocolos para la gestión de dispositivos de red, como OpenFlow, Netconf, etc. Todo el código está disponible bajo la licencia de Apache 2.0. Ryu está completamente escrito en

[3] https://www.linuxfoundation.org

Python y esto ha atraído a muchos desarrolladores debido a la curva de aprendizaje menos pronunciada que posee en comparación a, por ejemplo, ODL. Muchos son los proyectos nacidos a partir de RYU, uno de los más importantes es Faucet del cual hablaremos más adelante.

Nombre	NOX	POX	Floodlight	ODL	ONOS	RYU
Lenguaje	C++	Python	Java	Java	Java	Python
Arquitectura	Centralizado	Centralizado	Centralizado	Descentralizado	Descentralizado	Centralizado
NB. API	Ad-Hoc	REST	REST	REST, NETCONF, XMPP	REST, Neutron	REST
SB. API	OpenFlow	OpenFlow	OpenFlow	OpenFlow	OpenFlow	OpenFlow
EB. API	-	-	-	Akka, Raft	Raft	-
S.O	Linux	Linux. Windows, MacOs.	Linux, Windows, MacOs.	Linux, Windows, MacOs.	Linux, Windows, MacOs	Linux, MacOS
Licencia	GPL	Apache	Apache	EPL	Apache	Apache
Modularidad	Baja	Baja	Moderada	Alta	Alta	Moderada
Docu	Limitada	Limitada	Buena	Muy buena	Muy buena	Muy buena

Tabla 2. Resumen principales controladores SDN

2.1.4. La seguridad en las SDN.

Una vez introducidos los principales elementos que componen una red definida por software y explicadas las diferencias con las redes tradicionales, se pasa a exponer los desafíos a nivel de seguridad que presenta el uso de esta tecnología.

Como en cualquier otro nuevo paradigma las posibilidades que ofrece o puede llegar a ofrecer son muchas veces inciertas. Esto, en términos de seguridad presenta un aspecto para tener en cuenta ya que, esta incertidumbre se puede traducir en futuras debilidades que pueden terminar comprometiendo nuestro sistema.

Sin embargo, el estado actual de la investigación y la práctica en materia de seguridad es, en el mejor de los casos, fragmentario, local o específico para cada caso. Con infraestructuras modernas que soportan aplicaciones cada vez más complejas y omnipresentes, como las redes sociales, el Internet de las cosas, las aplicaciones móviles, los servicios en la nube, etc. Es preciso inventar tecnologías que se ajusten a la complejidad de las aplicaciones emergentes y a la sofisticación de sus atacantes.

Pero ¿qué se entiende por seguridad asociada a un sistema informático? Para asegurar una entidad/sistema, se acepta ampliamente que se requieren tres atributos de seguridad esenciales: confidencialidad, integridad y disponibilidad. Además de estas, se suelen incluir otras propiedades como son la autenticidad y no repudio.

La confidencialidad garantiza que la información privada y/o sensible sobre distintos datos o personas, no sea revelada a usuarios no autorizados. La integridad implica que la información y el funcionamiento previsto del sistema no sean manipulados de forma inadvertida o deliberada por usuarios no autorizados. La disponibilidad asegura que los sistemas y servicios sean accesibles cuando se necesiten. La autenticidad vela por que los usuarios puedan ser verificados como quienes dicen ser y que las entradas que llegan al

sistema provengan de una fuente confiable. No repudio genera el requisito de identificar de forma biunívoca la pertenencia de una información a un individuo.

Se considera necesario para entender los siguientes párrafos la definición previa de dos conceptos claves como son [33]:

- Vulnerabilidad: es una debilidad de un sistema (componente/producto/sistema/entorno) que podría permitir a un atacante comprometer la confidencialidad, la integridad, la disponibilidad, la autenticidad o el no repudio de ese sistema.
- Amenaza: es toda acción que aprovecha una vulnerabilidad para atentar contra la seguridad de un sistema de información. Es decir, que podría tener un potencial efecto negativo sobre algún elemento de nuestros sistemas. Las amenazas pueden proceder de ataques (fraude, robo, virus), sucesos físicos (incendios, inundaciones) o negligencia y decisiones institucionales (mal manejo de contraseñas, no usar cifrado).

Principalmente, la seguridad informática se ocupa de identificar las vulnerabilidades de un sistema, evaluar el riesgo asociado a las amenazas que explotan dichas vulnerabilidades y de proporcionar soluciones de seguridad.

SDN trae consigo nuevos retos de seguridad más allá de los que existían en las redes tradicionales. A medida que SDN desacopla el plano de control del plano de datos, la tecnología trae consigo nuevos conjuntos de componentes, interfaces, así como muchos nuevos problemas de seguridad. Los desafíos de seguridad en la SDN pueden dividirse en base a sus tres capas:

El plano de datos puede sufrir diversas amenazas a la seguridad, como los switches OpenFlow maliciosos, el descubrimiento de reglas de flujo, los ataques de inundación (por ejemplo, la inundación de la tabla de flujo de los switches), los flujos de tráfico falsos o falsificados, la manipulación de credenciales y la existencia de un host interno malicioso.

El plano de las aplicaciones hereda desafíos de seguridad como las aplicaciones no autorizadas o no autenticadas, la inserción fraudulenta de funciones, la falta de métodos de autenticación y la falta de aprovisionamiento seguro.

El plano de control por su parte se enfrenta a varios problemas de seguridad relacionados con la naturaleza centralizada del controlador SDN. Las interfaces de comunicación, la aplicación de políticas, la modificación de las reglas de flujo para modificar los paquetes, la inundación de comunicaciones entre el controlador y el switch, así como las debilidades heredadas del sistema operativo donde se implementa el controlador. Dado que el plano de control en la arquitectura SDN actúa como el corazón de esta infraestructura de red virtual, las vulnerabilidades de seguridad en esta capa pueden causar fallos en toda la arquitectura de la red virtual provocando una Denegación de Servicio (DoS).

Como principal vulnerabilidad en este tipo de sistemas destacan los problemas de configuración. Las políticas y protocolos de seguridad de la red se desarrollan continuamente

a medida que se detectan las vulnerabilidades de la red. Sin embargo, hay poca protección de tales políticas si no se implementan o se desactivan sin comprender las implicaciones de seguridad del escenario de despliegue. En una red basada en SDN, es crucial que los operadores de red impongan la aplicación de políticas como TLS. Las malas configuraciones o el uso incorrecto de las características de seguridad pueden afectar a todas las capas de la arquitectura. Las SDN nos proporcionan la capacidad de programar fácilmente la red y de permitir la creación de políticas de flujo dinámico. Es, de hecho, esta ventaja la que también puede conducir a vulnerabilidades de seguridad ya que con políticas de múltiples aplicaciones o aplicadas en múltiples dispositivos, se deben detectar inconsistencias para resolver los conflictos que se puedan ocasionar entre ellas.

Por otro lado, recordando la separación realizada entre los conceptos vulnerabilidad y amenaza. Las principales amenazas a las que se puede ver expuesto un sistema basado en redes definidas por software son [34]:

- **Acceso no autorizado**: El controlador proporciona una abstracción a las aplicaciones para que éstas puedan leer/escribir el estado de la red, lo cual es efectivamente, un nivel de control de la red. Si un atacante se hace pasar por un controlador/aplicación, podría acceder a los recursos de la red y manipular el funcionamiento de esta.
- **Fuga de datos**: es posible que un atacante determine la acción aplicada a tipos de paquetes específicos mediante el análisis del tiempo de procesamiento de estos paquetes. Por ejemplo, el tiempo de procesamiento de un paquete que pasa directamente del puerto de entrada al puerto de salida será más corto que el que tarda un paquete en ser redirigido al controlador para su procesamiento. Por lo tanto, el atacante puede descubrir la configuración del switch. Con un conjunto de paquetes creados expresamente, un atacante podría inferir información adicional sobre el dispositivo de red. Una vez descubierto el tipo de paquete que se redirige al controlador, el atacante puede entonces generar un conjunto de solicitudes de flujo falsas que conducen a un ataque de denegación de servicio (DoS).
- **Modificación de datos**: Si un atacante es capaz de “secuestrar” el controlador, entonces tendría el control de todo el sistema. Desde esta posición privilegiada, el atacante puede insertar o modificar las reglas de flujo en los dispositivos de la red, lo que permitiría dirigir los paquetes a través de la red en su beneficio. La falta, en ocasiones, de adopción de protocolos de intercambio seguro de paquetes, como TLS (Transport Layer Security) por parte de los principales proveedores (y en la mayoría de los controladores y switches de código abierto) puede permitir que un atacante de tipo "hombre en el medio" se haga pasar por el controlador y lance diversos ataques; por ejemplo, manipular los mensajes de control enviados por el controlador o inyectar mensajes para romper la conexión.
- **Aplicaciones maliciosas/comprometidas**: dado que el controlador actúa como una abstracción del plano de datos para las aplicaciones y que SDN permite que las aplicaciones de terceros se integren en la arquitectura; una aplicación

maliciosa podría tener un efecto tan perjudicial en la red como un controlador comprometido. Del mismo modo, una aplicación mal diseñada o con fallos podría introducir involuntariamente vulnerabilidades en el sistema. Por ejemplo, un error conocido podría ser explotado por un atacante para llevar la aplicación a un estado inseguro.

- **Denegación de servicio**: Una de las principales debilidades de seguridad de la SDN es introducida por su propia arquitectura: la existencia de un controlador central y la separación de los planos de control y datos. Debido a la necesidad de comunicación entre el controlador y el dispositivo de la red, un atacante podría inundar el controlador con paquetes que requieren una decisión de regla de flujo y provocar que este no esté disponible para los usuarios que lo necesiten.

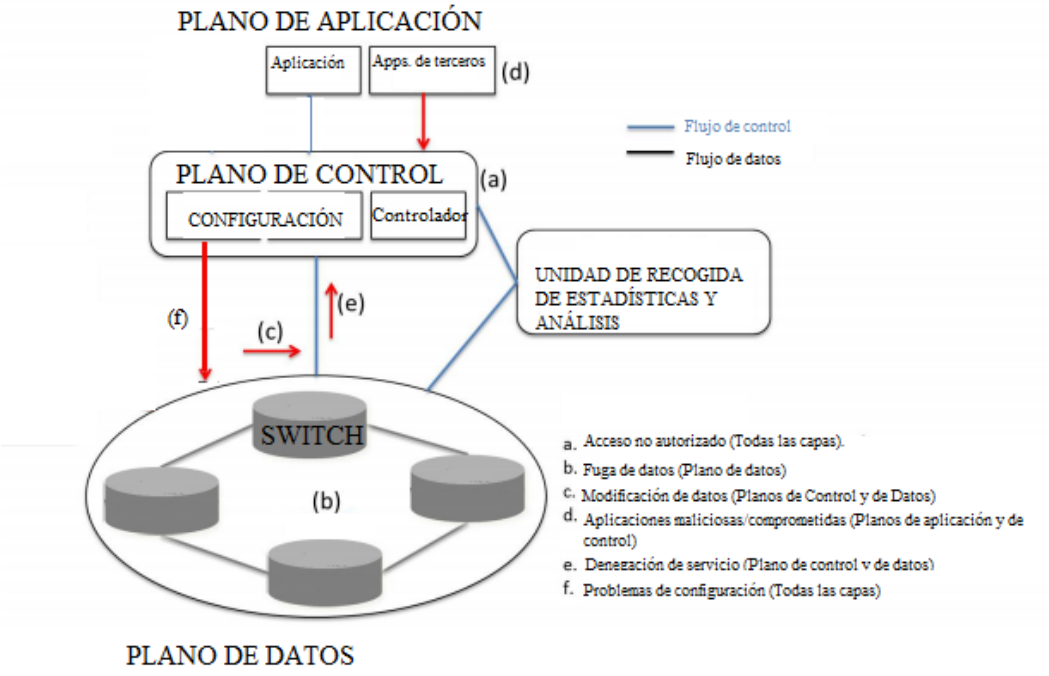


Ilustración 7. Potenciales amenazas y vulnerabilidades.

Seguridad SDN a nivel de sistema

Es vital, en términos de cumplimiento y operación de la red, poder proporcionar un inventario controlado de los dispositivos que la componen. Esto implica el conocimiento de qué dispositivos están funcionando, cómo están unidos a la red, etc. Por ejemplo, los switches OpenFlow pueden funcionar tanto en modo de seguridad como en modo autónomo. Cuando el switch se desconecta del controlador, la lógica interna del switch puede elegir operar en uno de estos modos. Al reconectarse, el controlador toma el control del switch y su estado de red interno. Desde una perspectiva operativa, es fundamental que un operador entienda el modo en que el switch funcionó durante la interrupción de la conexión, las entradas de flujo impactadas y el comportamiento del controlador después de restablecer la conexión. Para la responsabilidad y la auditoría, la capacidad de recuperar retrospectivamente tal información operacional es crítica y debe ser manejada en las SDN.

2.2. Sistema de Detección de Intrusos.

Ya se ha explicado la importancia de la información en la actualidad y de cómo el aumento exponencial de su volumen ha propiciado que surjan nuevas tecnologías como las redes definidas por software. Sin embargo, cuando se trata de seguridad muchas veces es preferible optar por sistemas fiables y cuya eficacia y rendimiento haya sido suficientemente puesto a prueba con éxito. Esto conlleva, que los sistemas de seguridad actuales no difieran a grandes rasgos con los utilizados hace una década, salvando todas las mejoras y perfeccionamiento que han ido experimentado.

Pese a que para la realización del presente trabajo se ha optado por la implementación de un sistema de detección de intrusos (IDS), es pertinente definir otros dos sistemas que junto a los IDS conforman los principales componentes de seguridad de cualquier entorno de red. Estos son los cortafuegos o firewalls y las redes privadas virtuales o VPN. Atendiendo a las definiciones que otorga el Instituto Nacional de Ciberseguridad (INCIBE)[4].

- Los firewalls o cortafuegos son un tipo de sistema de seguridad compuesto o bien de programas (software) o de dispositivos hardware situados en los puntos limítrofes de una red que tienen el objetivo de permitir y limitar, el flujo de tráfico entre los diferentes ámbitos que protege sobre la base de un conjunto de normas y otros criterios. La funcionalidad básica de un cortafuego es asegurar que todas las comunicaciones entre la red e Internet se realicen conforme a las políticas de seguridad de la organización o corporación.

[4] https://www.incibe.es

- Una red privada virtual, también conocida por sus siglas VPN (Virtual Private Network) es una tecnología de red que permite una extensión segura de una red local (LAN) sobre una red pública o no controlada como Internet. Al establecerlas, la integridad de los datos y la confidencialidad se protegen mediante la autentificación y el cifrado.

Por su parte, para la definición de un sistema de detección de intrusos/intrusiones se hará uso de la proporcionada por el manual de seguridad de Red Hat Enterprise Linux [35]:

Un sistema de detección de intrusos (IDS) es un proceso o dispositivo activo que analiza la actividad del sistema y de la red por entradas no autorizadas y/o actividades maliciosas. La forma en que un IDS detecta las anomalías pueden variar ampliamente; sin embargo, el objetivo final de cualquier IDS es el de atrapar a los perpetradores en el acto antes de que hagan algún daño a sus recursos.

Por lo que un IDS es un tipo de auditor de la red cuya función es alertar cuando se produzca algún evento no deseado dentro de ella o intentando acceder a esta, depende de la forma en la que se configure el sistema. El nacimiento de este tipo de mecanismo data de finales de los años ochenta [36] cuando se presentó un prototipo llamado IDES (Sistema Especialista en Detección de Intrusión). Este primer acercamiento se basaba en el estudio de patrones de comportamiento de usuarios legítimos para así mediante estadísticas de uso poder discernir un acceso por parte de un usuario autorizado, de un acceso llevado a cabo por un intruso o de un usuario no autorizado.

Sin embargo, debemos diferenciar IDS e IPS (Intrusion Prevention System). Mientras que el primero es un software que automatiza el proceso de detección de intrusos, el segundo es un software de prevención de intrusión, que tiene como objetivo impedir posibles ataques. Así que uno trabaja de manera reactiva e informativa, mientras que el IPS disminuye el riesgo de comprometimiento de un sistema.

2.2.1. Tipos de IDS

Dentro de los sistemas de detección de intrusos también existen distintos tipos en función de donde estén situados en la red, de la forma en la que detectan las intrusiones y según su comportamiento ante la detección de una intrusión. Así pues, se distinguen según su clasificación [37]:

Por situación:

- HIDS (Host IDS): Se enfoca en la detección basada en host de una única máquina, mirando sus registros de auditoria.
- NIDS (Network IDS): Se enfoca en la detección monitorizando el tráfico de la red a la que están conectados los hosts.

Por técnica de análisis usada:

- Detección de usos anómalos o basado en comportamiento: se basa en el estudio del comportamiento "normal" de la red para así detectar comportamientos que no se ajusten a esa normalidad y poder generar una alerta que lo notifique.
- Detección de usos indebidos o basado en conocimiento: esta otra técnica se apoya en el conocimiento de intrusiones previas o más comunes para así, mediante técnicas de *pattern matching*, generar alertas en el momento en el que detecte alguna de estas en el sistema. En este caso, es de suma importancia que se tenga una política de actualización continua de la base de datos de estas intrusiones previas para garantizar la efectividad del sistema, teniendo en cuenta que lo que no se conoce, literalmente, no será protegido.

Según su reacción ante una intrusión:

- Activa: el IDS tiene la capacidad de bloquear automáticamente actividades sospechosas o posibles intrusiones sin necesidad de intervención humana. Es un modelo que debe ser implementado con gran cuidado debido a los posibles falsos positivos en los que a un usuario legítimo se le puede bloquear el acceso a un recurso de la red, violando uno de los atributos principales en seguridad como es el de la disponibilidad.
- Pasiva: un comportamiento pasivo quiere decir que el IDS se limitará a monitorizar la red y de alertar en caso de que considere que se ha producido actividad sospechosa o un acceso ilegítimo. Estas alertas deberán ser tratadas por los administradores u otros equipos de seguridad, pero en ningún momento el sistema de detección de intrusos podrá interferir en la comunicación ni cambiar el comportamiento del sistema.

Para poner en funcionamiento un sistema de detección de intrusos se debe tener en cuenta que es posible optar por una solución hardware, software o incluso una combinación de estos dos. La posibilidad de introducir un elemento hardware es debido al alto requerimiento de procesador en redes con mucho tráfico. A su vez los registros de firmas y las bases de datos con los posibles ataques necesitan gran cantidad de memoria, aspecto a tener en cuenta.

En redes es necesario considerar el lugar de colocación del IDS. Si la red en la que se desea implementar este tipo de sistema de seguridad cuenta, además, con un firewall de red, la posición del IDS deberá ser considerada previamente, ya que:

- Si se posiciona antes del firewall: se analizará todo el tráfico de la red, tanto el entrante como el saliente. Esto podrá llevar a que el número de falsos positivos sea considerable.
- Si se posiciona después del firewall: únicamente se monitorizará el tráfico que no sea detectado y tratado por el propio cortafuegos. El porcentaje de que se produzcan falsos positivos desciende en gran medida.
- Si se posiciona junto al mismo firewall: de esta forma ambos sistemas trabajaran en paralelo, es decir, el firewall detecta los posibles paquetes maliciosos y el IDS los analiza.

2.2.2. Principales IDS

A la hora de comparar las distintas opciones de IDS disponibles, se ha optado por escoger aquellas herramientas más populares y de entre ellas, las alternativas de código abierto. Así, se han escogido tres de las muchas implementaciones de sistemas de detección de intrusos para comentar y diferenciar.

- Snort [5]: se trata de una de las herramientas NIDS más famosas actualmente. Snort es un software gratuito, bajo licencia GPL y puede ser instalado tanto en sistemas operativos Windows como en sistemas UNIX/Linux. Además, es un sistema probado y fiable y que cuenta con un gran soporte y actualizaciones conforme se van descubriendo nuevas vulnerabilidades a través de los distintos boletines de seguridad. Desde 2013, está respaldado por la compañía Cisco, lo cual hace que su mantenimiento y estado de actualización quede completamente asegurado. Dispone de varios modos de uso: logs, alertas por correo electrónico, integración con distintas plataformas, análisis en tiempo real… Como aspectos negativos a destacar, el uso de Snort exige muchas dependencias, por lo que la instalación puede resultar costosa, tiene una curva de aprendizaje muy pronunciada y su mala configuración puede llevar a que el número de falsos positivos sean muy altos.
- Bro/Zeek-IDS [6]: renombrada en 2018 como Zeek, esta herramienta combina los dos posibles funcionamientos de un NIDS según su técnica de análisis, por comportamiento y por conocimiento. Su implementación está limitada a sistemas Linux, FreeBSD y MacOs. Lo que diferencia a Zeek del resto de IDS es su gran capacidad de análisis de los paquetes; es capaz de tratar aspectos a más bajo nivel

[5] https://www.snort.org/
[6] https://docs.zeek.org/en/current/intro/

que cualquier otra herramienta. Esto trae consigo por su parte, que el usuario deba ser más especializado e incluso requiere del conocimiento de un lenguaje de programación propio basado en eventos (Bro-script). Sin embargo, Zeek cuenta con gran cantidad de usuarios y muchas universidades, centros de datos y demás corporaciones ya optan por su utilización a la hora de proteger sus infraestructuras de red.

- Suricata [7]: esta herramienta es, tanto un IDS como un IPS el cual puede funcionar en cualquier sistema operativo y promete un gran rendimiento gracias a la ejecución multi-hilo. Es de código abierto y propiedad de una fundación sin fines lucrativos dirigida por la comunidad, la Fundación Abierta de Seguridad de la Información (OISF)[8]. Desde su sitio oficial se asegura que cumple con todas las funcionalidades que posee Snort e incluso cuenta con las mismas firmas que esta otra; además incluye otros aspectos como son el almacenamiento de certificados TLS/SSL, peticiones HTTP… Su uso tiene mejores resultados en entornos de un elevado tráfico de red debido a su capacidad de trabajar de forma paralela. Sin embargo, se ha demostrado que en entornos donde el tráfico no es tan elevado su rendimiento se ve reducido.

2.3. Propuesta.

Se han visto hasta ahora las bondades y problemas que plantea el uso de un sistema basado en redes definidas por software, además se ha presentado con motivo de la necesidad de asegurar este sistema, las herramientas IDS de código abierto más importantes y los beneficios de su uso. Con todo esto, la búsqueda de la propuesta que se adecue a las necesidades vistas pasa por diseñar un sistema que junte los siguientes principios:

- Simplicidad: se tratará de presentar una solución lo más accesible para cualquier usuario que se interese por implementar un sistema similar a cualquier nivel.
- Eficiencia: la calidad de servicio del sistema deberá ser una prioridad y las decisiones que tomar serán en vistas de mejorar esta, en sintonía con los parámetros de seguridad que se consideren necesarios.
- Versatilidad: el sistema propuesto se podrá adaptar a cualquier necesidad tanto de seguridad como de análisis y monitorización futuras.
- Seguridad: se pretende diseñar un sistema lo más seguro posible que cubra con las necesidades que plantee el entorno en el que se implementará.
- Escalabilidad: la solución propuesta pretende ser ambiciosa con vistas a futuras mejoras y modificaciones las cuales serán comentadas más adelante.

[7] https://suricata.readthedocs.io/en/latest/
[8] https://oisf.net

Tras un análisis de las distintas alternativas actuales que se adapten a estos objetivos se ha optado por la implementación de un sistema el cual combine el uso de las redes definidas por software complementado con un sistema de detección de intrusos.

Algunos trabajos previos han abordado el uso de las redes definidas por software con objeto de mejorar la seguridad de un sistema. Sin embargo, en la mayoría de estos no se terminaba de dar con una solución completa o si se hacía requería de grandes conocimientos en programación debido a que muchos controladores requieren de conocer un lenguaje específico para la creación de reglas que lo hagan seguro. Por otro lado, la combinación de estos dos sistemas (SDN + IDS) tampoco ha sido un tema tratado en profundidad con anterioridad (por supuesto si acotamos al ámbito universitario en España).

3. Análisis del problema

Como se ha comentado en apartados anteriores, las redes definidas por software son una tecnología en constante crecimiento y que, en muchos casos, las líneas de investigación que se han seguido con respecto a ellas han decidido no enfocarse en el apartado de la seguridad. Sin embargo, muchos son también los que han visto esta tecnología como una gran oportunidad para mejorar la calidad de servicio de un entorno sin tener por ello que olvidar la seguridad de este e incluso, conseguir mejorarla en ocasiones.

Aun así, muchas de estas investigaciones en líneas de seguridad se encuentran en su etapa más primeriza o avanzan en direcciones muy dispersas, motivo de esto puede ser la importancia que están ganando los sistemas inteligentes y el aprendizaje automático. Esto puede hacer que un desarrollador sin mucha experiencia o incluso un estudiante con interés en seguridad en redes, se encuentre con grandes dificultades para poder introducirse en este mundo.

A la conclusión a la que se puede llegar es que es necesario acercar, de una forma amigable, esta tecnología que tantas oportunidades ofrece en materia de seguridad y de calidad de servicio, para que cada vez sean más los que se sumen y así conseguir obtener resultados mucho más rápido o nuevas ideas que den pie a nuevas alternativas. También, es necesario destacar que esta tecnología aún no se ha hecho con un hueco en las aulas universitarias en España. Sin embargo, las posibilidades didácticas que son capaces de ofrecer pueden llegar a ser de gran interés.

Mediante este proyecto se pretende aproximar una solución simple pero eficiente que demuestre cómo se puede mejorar la calidad de servicio de un sistema mediante la selección y el tratamiento de las tramas de red y de las rutas de encaminamiento en función de los parámetros de seguridad considerados y de las condiciones variantes del sistema.

3.1. Solución propuesta

La solución que se propone y que será mostrada en próximos apartados se trata de la elección de las herramientas idóneas a utilizar junto al diseño de una topología de red adecuada que permita cumplir con los objetivos e intereses marcados. Para ello ha sido preciso el análisis del problema y la comparación de las alternativas que puedan satisfacer los requisitos propuestos. Se ha tenido en consideración como es necesario, que esta pueda ser realizable y que además se pueda adaptar al propósito de este trabajo.

El siguiente paso será detallar el diseño de esta solución la cual, como se ha comentado, combinará el uso de las redes definidas por software junto a un sistema de detección de intrusos. Para el diseño se optará por explicar el porqué de la selección de cada uno de los elementos tratando de justificar estas decisiones. Una vez presentado el diseño, se expondrá como se ha llevado a cabo su implementación y finalmente se realizarán una serie de pruebas. En ellas se podrá comprobar si de verdad se ha logrado el propósito de este proyecto mediante comprobaciones en materia de seguridad, calidad de servicio y demostrando las utilidades para la monitorización y análisis.

4. Diseño de la solución

Debido a que la solución propone la coordinación de dos sistemas independientes se estima necesario definir cada uno de estos dos sistemas por separado para poder entender cómo terminarán funcionando en combinación. Esta solución puede ser una aproximación o un prototipo de otra futura más completa.

4.1. ARQUITECTURA

A continuación, se va a comentar la arquitectura que ha sido considerada para la posterior implementación de la solución. En la Ilustración 9, se puede observar el diseño de esta, el cual consta del controlador Faucet como elemento central de la topología. Este, será el encargado de la orquestación del resto de componentes, así como de la toma de decisiones en cuanto al enrutamiento y el flujo de datos. El siguiente elemento que destacar es, el sistema de detección de intrusiones (IDS), para ello se ha escogido la herramienta Snort presentada previamente. Dicha elección será justificada más adelante, así como los principios de su funcionamiento.

Para la conexión de estos dos elementos y como elemento fundamental de la capa de datos se ha optado por el uso de la herramienta Open vSwitch. De la misma manera, se podría haber escogido otra herramienta más conocida por haber sido implementada en trabajos previos como es Mininet [9]; aun así, debido a su popularidad y la multitud de funciones que ofrece OVS, se ha encontrado relevante su uso y exposición en el actual trabajo. Por último, debido a la necesidad de la presencia de otros elementos que simulen componentes de la red para futuras pruebas, se han utilizado los espacios de nombres de red de Linux para conseguir independencia entre ellos pese a realizarse todo desde la misma máquina.

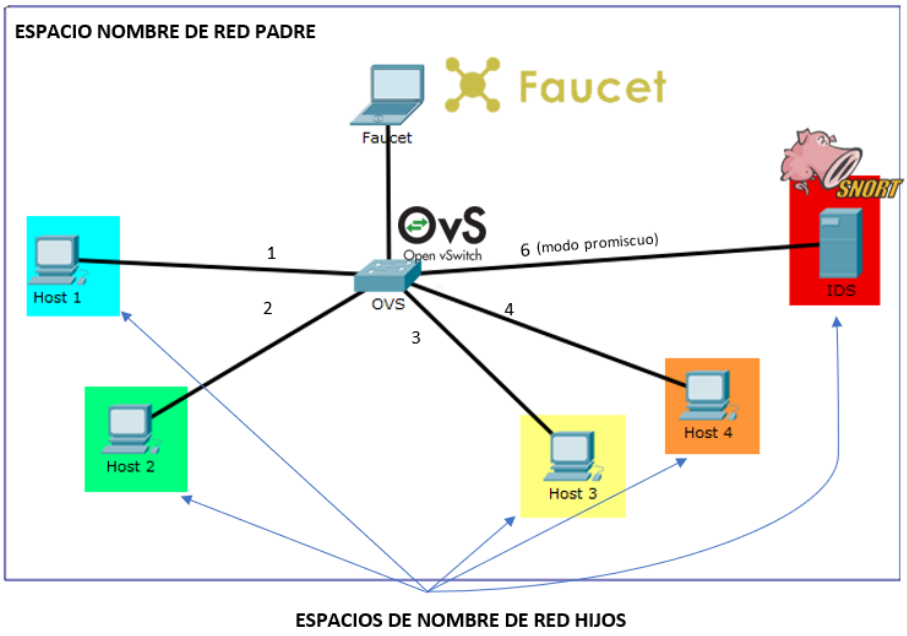


Ilustración 8. Arquitectura solución

[9] http://mininet.org/

4.1.1. Faucet.

No ha sido mencionado previamente y esto se debe a que Faucet es un controlador, compatible con el protocolo OpenFlow, que está basado principalmente en el controlador ya explicado, Ryu. Además, recientemente la Faucet Foundation ha pasado a ser la nueva encargada de mantener el proyecto Ryu.

Faucet[10] se trata de un proyecto de código abierto el cual nace en 2015 cuando REANNZ (Research & Education Advanced Network New Zealand) escribe la primera versión basada en un prototipo del grupo de investigación WAND de la Universidad de Waikato, en Nueva Zelanda.

Actualmente es uno de los proyectos de código abierto más importantes en el ámbito de las Redes Definidas por Software. Esto es gracias a la gran comunidad que lo respalda donde cada año se incrementa tanto el número de desarrolladores como también el número de usuarios y de organizaciones que están implementando esta solución.

Su comunidad es también una de las razones por la cual se ha elegido este controlador para el desarrollo del presente trabajo ya que se dispone de una gran cantidad de documentación y ayuda sobre como implementarlo en cualquier entorno; siendo posible incluso, el despliegue de este controlador en una sencilla Raspberry Pi, un pequeño ordenador no más grande que la palma de una mano.

FAUCET, aunque como se ha dicho está basado en Ryu, añade diversas funcionalidades que lo hacen de gran interés como son:

- La posibilidad de utilizar puertos etiquetados y así poder crear redes de área local virtualizadas o VLANs.
- Listas de control de acceso (ACLs) compatibles con los campos de la capa de enlace (capa 2) y la capa de red (capa 3) según el modelo OSI.
- Enrutamiento IPv4 y IPv6, estático y a través del protocolo de puerta de enlace de frontera o BGP (Border Gateway Protocol).
- Reenvío basado en políticas para la utilización de funciones de red virtualizadas o NFV por sus siglas en inglés.
- Estadísticas tanto para puertos como para el flujo de red a través de InfluxDB y Grafana. Estas herramientas serán explicadas más adelante.
- Estadísticas también para conocer el estado del controlador a través de Prometheus.
- Capacidad de testing gracias a la herramienta Travis, basada en Mininet y en Open Virtual Switch.

[10] https://faucet.nz/

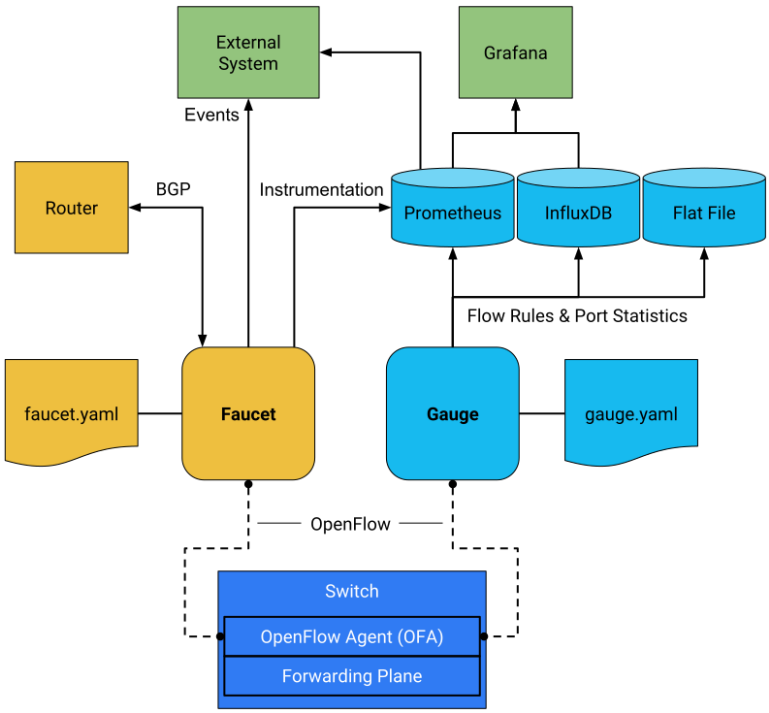
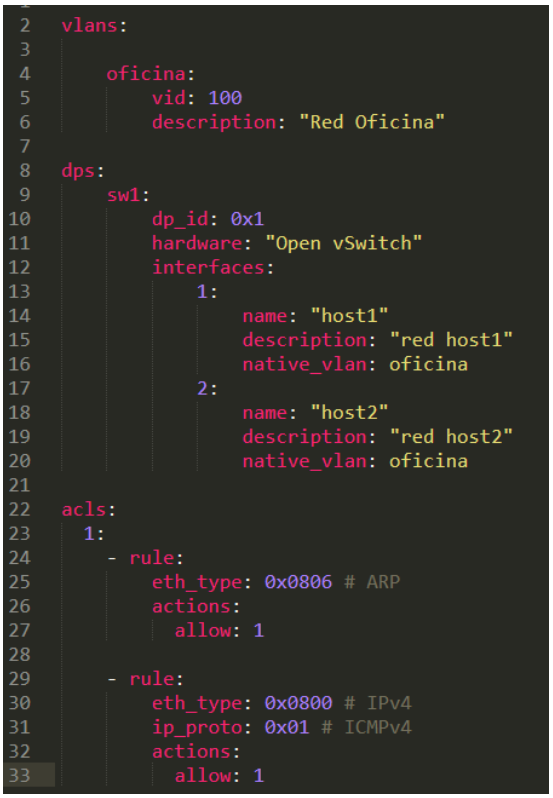


Ilustración 9. Arquitectura Faucet [38]

CONFIGURACIÓN FAUCET

Faucet utiliza un archivo de configuración YAML. YAML, o YAML Ain't Markup Language, es un lenguaje de notación de objetos muy parecido a JSON (JavaScript Object Notation), pero soporta un formato mucho más legible que admite comentarios como CSON (CoffeeScript Object Notation). La estructura de este archivo se puede ver en la ilustración 11.

Podemos distinguir en este archivo de configuración tres parámetros principales, todos ellos de tipo diccionario, esto significa que, sus posibles valores son una lista de pares clave-valor. Los diccionarios son un tipo de estructuras de datos típicas de algunos lenguajes de programación como puede ser Python. Estos parámetros, aun no siendo los únicos, debido a su importancia serán los explicados a continuación.



```yaml
vlans:

    oficina:
        vid: 100
        description: "Red Oficina"

dps:
    sw1:
        dp_id: 0x1
        hardware: "Open vSwitch"
        interfaces:
            1:
                name: "host1"
                description: "red host1"
                native_vlan: oficina
            2:
                name: "host2"
                description: "red host2"
                native_vlan: oficina

acls:
  1:
    - rule:
        eth_type: 0x0806 # ARP
        actions:
          allow: 1

    - rule:
        eth_type: 0x0800 # IPv4
        ip_proto: 0x01 # ICMPv4
        actions:
          allow: 1
```

Ilustración 10. Ejemplo archivo configuración YAML

VLANS. Es el apartado específico para la configuración de redes virtuales de área local. Sirven para simular entornos de red donde dos elementos de la misma, separados entre sí (por ejemplo, dos subredes distintas) puedan interconectarse. Algunos de los atributos a destacar que se pueden especificar a la hora de crear una VLAN son:

- Acl_in: indica que lista de control de acceso será aplicada a los paquetes que lleguen a la VLAN en cuestión. Su valor será un entero.
- Vid: indica el identificador de la VLAN. Su valor será un entero.
- Description: permite una descripción breve de la red virtual creada. Su valor es por tanto una cadena de texto.

DPS. Este parámetro hace referencia a las rutas de datos. Las claves son nombres o dp_ids de cada ruta de datos, y los valores son diccionarios de configuración que contienen la configuración de la ruta de datos. Así pues, este parámetro servirá para definir los distintos elementos que conformarán la capa de datos como pueden ser los switches. Son de especial interés los siguientes atributos en la configuración de una ruta de datos.

- Dp_id: se trata del identificador de la ruta de datos, por ejemplo, el switch 1.
- Hardware: por defecto el valor de este atributo es Open vSwitch, del cual se hablará en detalle más adelante.

- Interfaces: se trata de una estructura de tipo diccionario, sus distintas claves permiten la configuración de cada uno de los puertos OpenFlow que se requieran (hasta 6), desde a qué VLAN pertenecen hasta la aplicación de listas de control de acceso específicas.

ACLS. Mediante este parámetro es posible la definición de listas de control de acceso, estas son un conjunto de reglas por las cuales es posible determinar los permisos apropiados a un objeto, pudiendo cubrir algunos de las características de un firewall gracias a la multitud de opciones de configuración de las que se disponen. Algunas de ellas:

- Eth_type: tipo de paquete (ARP, IPv4…).
- Ip_proto: tipo de protocolo IP (IPv4, IPv6).
- Actions: en este campo se pueden definir las acciones que se consideran pertinentes para aquellos paquetes que cumplan con dicha regla. Lo principal es permitir o no su paso.

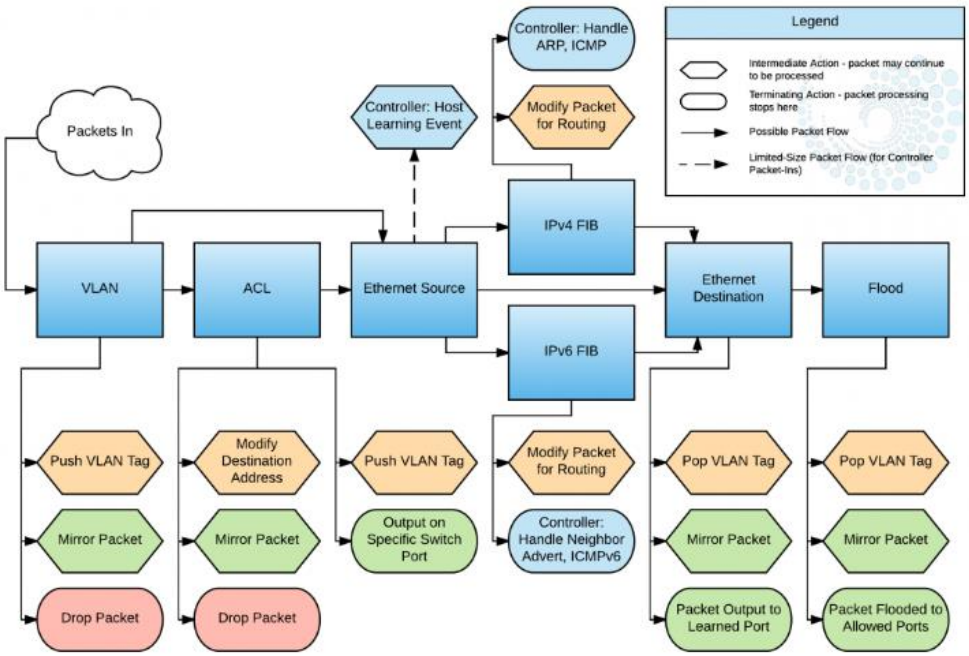


Ilustración 11. Flujo de paquetes en Faucet - Posibles acciones y caminos [39]

Como se ha explicado previamente, el funcionamiento del protocolo OpenFlow se basa en la validación, mediante las llamadas tablas de flujo, de los paquetes entrantes (Ilustración 5). Al tratarse de Faucet un controlador compatible con este protocolo, utiliza de la misma forma estas tablas para el tratamiento de los paquetes. Así mismo, en Faucet se estructuran las tablas de flujo de la siguiente manera:

- **Tabla 0**: ACLs aplicadas a puertos.
- **Tabla 1**: VLAN procedencia.
- **Tabla 2**: ACLs aplicadas a la VLAN.
- **Tabla 3:** procesamiento capa 2, aprendizaje MAC.
- **Tabla 4:** reenvío capa 3 para IPv4.
- **Tabla 5:** reenvío capa 3 para IPv6.
- **Tabla 6:** procesamiento de IPs virtuales.
- **Tabla 7:** procesamiento de salida en capa 2.
- **Table 8:** Difusión.

GAUGE

Faucet tiene dos componentes principales del controlador OpenFlow, el propio Faucet y Gauge. Faucet controla todos los estados de reenvío y conmutación, y expone su estado interno, por ejemplo, los hosts aprendidos, a través de Prometheus (para que, posteriormente, la herramienta de código abierto Grafana lo muestre gráficamente).

Gauge también tiene una conexión OpenFlow con el switch y monitoriza el estado de los puertos y del flujo (exportándolo a Prometheus o a InfluxDB, o incluso a archivos de registro de texto plano). Sin embargo, Gauge no modifica nunca el estado del switch, por lo que las funciones de monitorización del switch pueden ser actualizadas o reiniciadas, sin afectar al reenvío, es decir, sin interrumpir el correcto funcionamiento del sistema.

Estas dos herramientas, aunque independientes de Faucet, se incluyen dentro del paquete de instalación del controlador. Ambas están relacionadas con la gestión de estadísticas y el análisis mediante gráficas. Por un lado, Prometheus[11] se trata de un motor de estadísticas y alertas en tiempo real. Dentro de Faucet, recibe las métricas de la instrumentación por parte del controlador homónimo como se puede observar en la Ilustración 9; y las estadísticas de la red gracias al controlador Gauge explicado en el apartado anterior. Por otro lado, Grafana[12] es el software encargado de la visualización y el tratamiento de estas estadísticas y métricas una vez procesadas por Prometheus.

Estas dos herramientas, ambas de código libre, también permiten la integración con otro tipo de sistemas, por lo que hacen posible la integración de todas las estadísticas generadas en el sistema propuesto en la solución. Tanto las provenientes de la parte de la red, es decir, por Faucet y los diferentes elementos del plano de datos (switches, etc). Como de las alertas y avisos generados por el sistema de detección de intrusos.

4.1.2. SNORT

Como se ha explicado previamente, para el diseño de la solución se ha optado por la implementación de un sistema de detección de intrusos. De entre las posibles opciones que se han expuesto la que finalmente se ha escogido es, la herramienta Snort. Las razones de esta decisión son múltiples:

-En primer lugar, por su carácter de código abierto pero respaldado por una empresa como Cisco lo cual aporta confianza y seguridad.

-Seguidamente, esta herramienta posee una curva de aprendizaje menos pronunciada con respecto a las otras dos opciones previamente mencionadas (Suricata y Zeek-IDS) haciéndolo más accesible, atributo de especial interés para este trabajo.

-Finalmente, por su predominio como herramienta IDS más utilizada en la actualidad, esto convierte a Snort y al usuario experimentado en él en un objetivo potencial para cualquier empresa que tenga implementado un sistema de detección de intrusos.

FUNCIONAMIENTO

[11] https://prometheus.io/docs/introduction/overview/
[12] https://grafana.com/docs/grafana/latest/getting-started/what-is-grafana/

Snort, como se ha comentado, es un sistema de detección de intrusos basado en red o NIDS. Así mismo, implementa un motor de detección de ataques el cual permite registrar, alertar y responder ante cualquier anomalía previamente definida. Gracias a estos registros y a su almacenamiento en modo binario, es posible el posterior análisis en detalle además de su exportación e integración con otras herramientas de visualización de estadísticas en tiempo real. Snort brinda la posibilidad de tres modos de funcionamiento:

- Modo *Sniffer*: un *Sniffer* es un programa y/o dispositivo que monitoriza toda la información que pasa a través de una red. Su función es "olfatear" los datos que pasan por la y determinar a dónde van, de dónde vienen y qué son [40]. En este modo de funcionamiento se captura el tráfico de la red donde se ha configurado y simplemente imprime por pantalla (o consola) lo capturado.
- Modo registro de paquetes: los paquetes detectados por Snort en base a su configuración previa serán almacenados en ficheros para su análisis a posteriori e incluso, para volver a reproducir el tráfico almacenado en estos ficheros.
- Modo NIDS: este modo de funcionamiento se trata del más complejo y también el más configurable de todos. Se comparan los paquetes entrantes con el conjunto de reglas o patrones configurados, en función de las coincidencias se mostrará por pantalla (o consola) el detalle de estos o se almacenarán estas alertas en un sistema basado en registros.

```
alert tcp any any -> any 80 (msg:"Alerta"; content:"/secret/pass";
priority:10;)
```

Este sería un ejemplo de una regla de Snort muy básica. Cuando se define una regla se deben diferenciar dos partes. La primera sería la cabecera:

```
alert tcp any any -> any 80
```

Esta cabecera indica el evento que activará la regla, en este caso será cualquier conexión tcp desde cualquier dirección y puerto hacia cualquier dirección en el puerto 80.

La segunda parte de la regla sería la correspondiente a las opciones:

```
(msg:"Alerta"; content:"/secret/pass"; priority:10;)
```

En esta parte de la regla se declara lo que se desea que se realice una vez se detecte una coincidencia con la cabecera, siguiendo el ejemplo, ante una coincidencia con la cabecera, se notificará mediante un mensaje "Alerta" en el caso en que se intente acceder al recurso web "/secret/pass". Más adelante se explicará en profundidad qué más es posible controlar mediante estas reglas y cuan preciso se puede llegar a ser si así se desea.

4.1.3. OVS y Espacios de nombres de Red.

Se han presentado los dos pilares de la arquitectura, el controlador Faucet (junto a las distintas herramientas que lo componen) y el sistema de detección de intrusos Snort. Sin embargo, ambos elementos necesitan un nexo para poder funcionar en sintonía y que formen un único sistema compuesto. A continuación, se van a exponer dos herramientas las cuales harán posible este enlace y cuyo correcto funcionamiento y configuración será crítico para el correcto despliegue de la solución.

Por un lado, OVS se tratan de las siglas de Open vSwitch[13], un software de código abierto diseñado con el objetivo de funcionar como un switch multicapa en entornos virtualizados. Es compatible con las interfaces de gestión estándar y, además, incluye otras características relacionadas con las redes programables. Aunque sus funcionalidades y posibilidades son numerosas, para este trabajo serán de interés dos de ellas: configuración QoS (calidad de servicio) y compatibilidad con el protocolo OpenFlow.

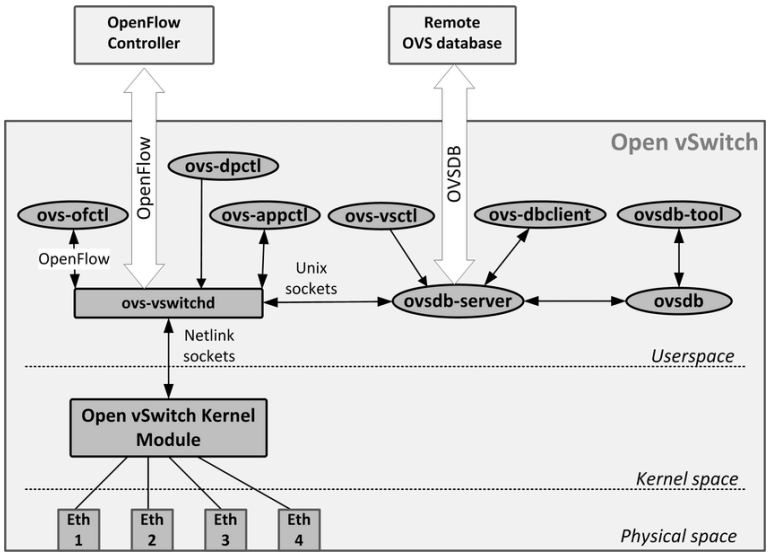


Ilustración 12. Arquitectura OVS [41]

Por otro lado, los network namespaces o espacios de nombres de red [42], son una característica del kernel de Linux mediante la cual es posible crear abstracciones dentro de un sistema operativo de modo que estas formen un sistema aislado que cuenta con sus propios recursos de red: interfaces de red, protocolo IPv4 e IPv6, pilas, tablas de enrutamiento IP, reglas del cortafuegos… Con esto se consigue que dentro de una misma máquina sea posible la simulación de independencia entre sistemas y así poder realizar pruebas de funcionamiento como si de una red real se tratara.

[13] http://docs.openvswitch.org/en/latest/intro/what-is-ovs/

5. Implementación

Partiendo de las herramientas explicadas en el apartado anterior y de la arquitectura diseñada (Ilustración 8), el siguiente paso será llevar a cabo la implementación de la solución con el objetivo de obtener el comportamiento deseado. Para ello se ha elegido el software de virtualización VMWare [14] el cual permite la creación de máquinas virtuales. Para la implementación que se ha llevado a cabo será necesaria tan solo una máquina virtual, en ella se instalará el sistema operativo Ubuntu 18.04 LTS. Todo el detalle de la instalación puede ser consultado en los anexos al final de este documento.

Con estos requisitos previos ya instalados, se explica a continuación cómo se ha llevado a cabo la implementación de la solución al problema propuesto. Para ello se abordará en primera instancia la instalación y configuración del paquete Faucet (Faucet, Gauge, Prometheus y Grafana); acto seguido se pasará a instalar el IDS Snort y se configurará para trabajar en modo NIDS, a modo de integrar todos los datos del sistema en la aplicación Grafana, se utilizará una base de datos MySQL[15] donde se guardarán los avisos de Snort y serán visualizados posteriormente.

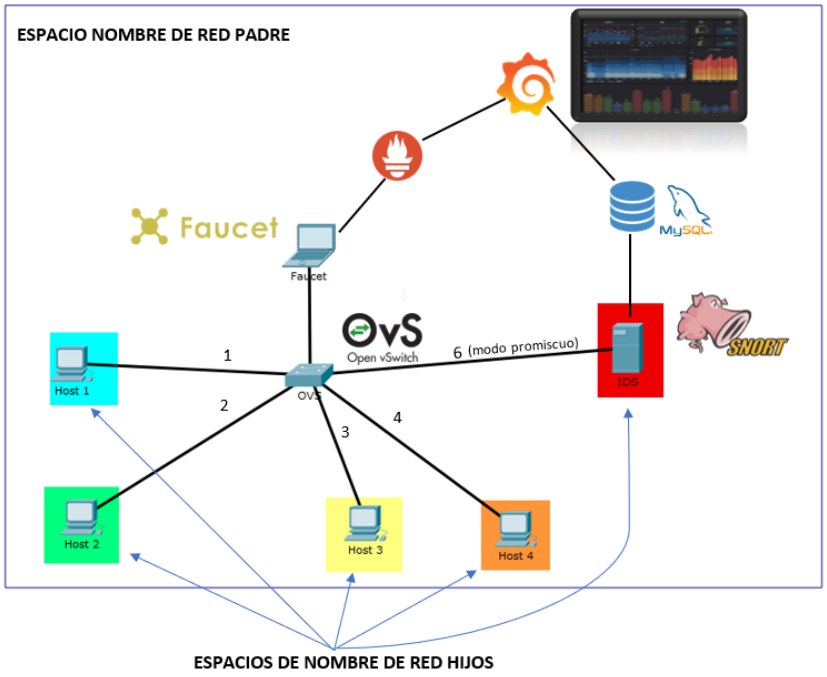


Ilustración 13. Implementación.

[14] https://www.vmware.com
[15] https://www.mysql.com

5.1. Instalación y configuración de Faucet.

A la hora de la instalación del paquete Faucet, se ha acudido a la documentación aportada desde el sitio oficial donde se presentan distintas guías de cómo realizar esta instalación según la plataforma deseada (Docker, Ubuntu, Raspberry Pi…). Debido a que el sistema operativo utilizado para la solución se trata de Ubuntu, se utilizará la Herramienta Avanzada de Empaquetado, APT por sus siglas en inglés, para la instalación del metapaquete "Faucet-all-in-one". A modo de simplificar la lectura del presente documento, el detalle de esta instalación puede ser consultado en el anexo correspondiente.

Siguiendo los pasos descritos para la instalación, se obtiene como resultado la correcta instalación e interconexión entre los elementos que componen el paquete Faucet como son Faucet, Gauge, Grafana y Prometheus.

5.2. Instalación y configuración de Snort.

La instalación de Snort resulta algo más compleja debido a la cantidad de dependencias que requiere. Por esto, se incluye el detalle de toda la instalación paso a paso en el anexo correspondiente al final de este documento. Si toda la instalación es correcta al escribir el comando:

```
$snort -V
```

Se obtendrá (Ilustración 14) una respuesta indicando la versión instalada de Snort.

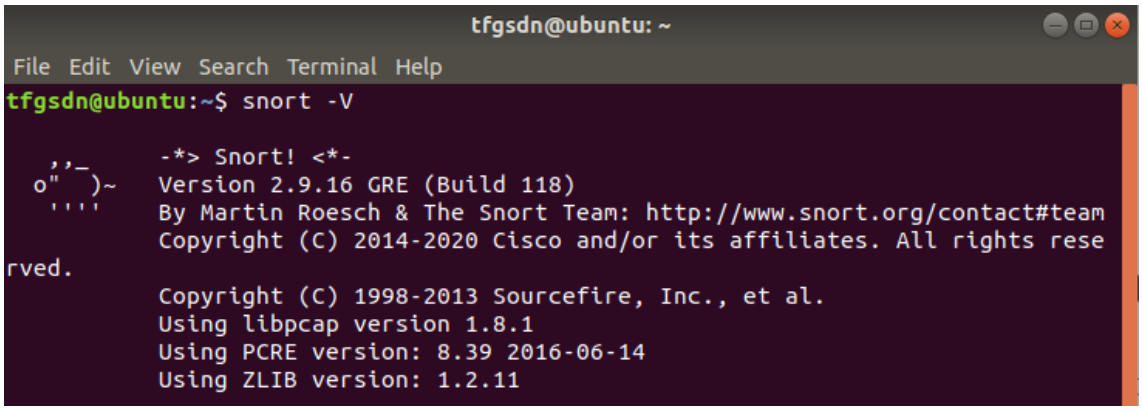


Ilustración 14. Instalación correcta Snort

El siguiente paso es configurar Snort para que funcione en modo NIDS de acuerdo con los objetivos que se han marcado. Para ello, el primer paso es la creación de las carpetas y ficheros requeridos por Snort. A su vez, se siguen buenas prácticas en seguridad como es la creación de un usuario sin privilegios y un grupo desde el que se ejecutará el proceso Snort, todos los permisos de las carpetas creadas serán correctamente ajustados para que sean accedidos por este usuario creado. Esto se detalla en el anexo al final del documento.

Llegados a este punto, lo único que resta es la creación de las reglas personalizadas que se consideren interesantes para el sistema en cuestión. La creación de estas reglas se verá más adelante en el apartado de pruebas realizadas, así como el funcionamiento al completo del conjunto de elementos. El comando para poner Snort en funcionamiento es el siguiente:

```
$sudo snort -i “interfaz” -u snort -g snort -c /etc/snort/snort.conf
```

5.3. Base de datos MySQL

Con el IDS Snort configurado, es necesario la integración de los avisos de este en Grafana con el objetivo de centralizar todas las alertas y estadísticas, tanto las de Faucet como las de Snort. Debido a que no es posible la integración de Snort y Prometheus, se ha considerado como mejor alternativa, la creación de una base de datos donde se almacenen las alertas de Snort y, por otro lado, la creación de un tablero en Grafana que muestre la información que contiene esta base de datos.

Para ello, el primer paso es instalar la extensión de Snort llamada Barnyard2[16]. Esta herramienta permite la exportación de los archivos de registro generados por el IDS para poder integrarlos en, por ejemplo, una base de datos ya que, de manera nativa, Snort no permite esta opción. Siguiendo el procedimiento anterior, el detalle de la instalación de esta herramienta, así como el de sus dependencias con MySQL se puede consultar en el anexo.

No se va a extender en la explicación detallada de esta base de datos pues no es el objetivo del trabajo. Esta base de datos no es más que una herramienta la cual permite la visualización de las alertas de Snort mediante su integración con la otra herramienta mencionada, Grafana. Así, se consigue un análisis del estado del sistema centralizado, visual y cómodo.

[16] https://github.com/firnsy/barnyard2

5.4. Interconexión de los elementos

El último de los pasos de esta implementación es el relacionado con la conexión de los distintos elementos que componen el sistema propuesto. Para ello, como ya se subrayó en el diseño, serán utilizados los espacios de nombres de red y el software libre de enrutamiento compatible con Openflow, OVS. Por defecto, OVS viene con el paquete Faucet-all-in-one por lo que no se va a explicar su instalación; de la misma forma, los espacios de nombres de red es una característica del kernel de Linux, luego ambas herramientas se encuentran ya disponibles en el sistema propuesto.

Se pasa a comentar la configuración de estas. Para facilitar la creación de espacios de nombres y el uso de estos se van a utilizar dos funciones Bash. La primera de ellas se trata de la función asociada a la ejecución de un comando dentro del espacio de red indicado, esta recibe el nombre de "*as_ns()"* y, continuando con el objetivo de facilitar la lectura del documento, tanto esta función como las siguientes serán detalladas en el Anexo correspondiente al final del presente trabajo. La segunda de las funciones está relacionada con la creación de un espacio de nombre de red, esta se ha definido con el nombre *"create_ns*()".

Ambas funciones serán añadidas al final del archivo .bashrc para que puedan ser utilizadas desde la línea de comandos sin tener que definirlas cada vez que se reinicie el sistema. Una vez se han definido, el siguiente paso será crear los distintos espacios de red para cada host.

```
tfgsdn@ubuntu: create_ns host1 192.168.2.2
tfgsdn@ubuntu: create_ns host2 192.168.2.3
tfgsdn@ubuntu: create_ns host3 192.168.2.4
tfgsdn@ubuntu: create_ns host4 192.168.2.5
tfgsdn@ubuntu: create_ns snort 192.168.3.1
```

Con los espacios de red creados para cada uno de los elementos del sistema, el siguiente paso será configurar el switch OVS y asignar a cada puerto de este la interfaz correspondiente a cada uno de los espacios virtuales.

```
tfgsdn@ubuntu: sudo ovs-vsctl add-br br0 \
-- set bridge br0 other-config:datapath-id=0000000000000001 \
-- set bridge br0 other-config:disable-in-band=true \
-- set bridge br0 fail_mode=secure \
-- add-port br0 veth-host1 -- set interface veth-host1 ofport_request=1 \
-- add-port br0 veth-host2 -- set interface veth-host2 ofport_request=2 \
-- add-port br0 veth-host3 -- set interface veth-host3 ofport_request=3 \
-- add-port br0 veth-host4 -- set interface veth-host4 ofport_request=4 \
-- add-port br0 veth-snort -- set interface veth-snort ofport_request=6 \
-- set-controller br0 tcp:127.0.0.1:6653 tcp:127.0.0.1:6654
```

Una vez hecho esto, si utilizamos el comando ifconfig en el espacio de red anfitrión se observa, como se ejemplifica en la Ilustración 15, una interfaz de red virtual por cada uno de los hosts creados:

```
tfgsdn@ubuntu:~$ ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.106.138  netmask 255.255.255.0  broadcast 192.168.106.255
        inet6 fe80::79e:a7d1:520d:523a  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:07:91:33  txqueuelen 1000  (Ethernet)
        RX packets 304720  bytes 23989836 (23.9 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 5389  bytes 520506 (520.5 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Bucle local)
        RX packets 100435  bytes 72785928 (72.7 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 100435  bytes 72785928 (72.7 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

veth-host1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::78cb:5fff:fe72:75be  prefixlen 64  scopeid 0x20<link>
        ether 7a:cb:5f:72:75:be  txqueuelen 1000  (Ethernet)
        RX packets 112215  bytes 6059906 (6.0 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 112240  bytes 6062939 (6.0 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

veth-host2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::343b:52ff:fe1a:727f  prefixlen 64  scopeid 0x20<link>
        ether 36:3b:52:1a:72:7f  txqueuelen 1000  (Ethernet)
        RX packets 112215  bytes 6059906 (6.0 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 112240  bytes 6062939 (6.0 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Ilustración 15. Algunas de las interfaces de red creadas

Completados todos estos pasos, se tiene el sistema tal y como se presentó en la Ilustración 13. En siguientes apartados se verán pruebas de funcionamiento de este, así como ejemplos donde su uso puede resultar ventajoso y se analizarán los resultados obtenidos para elaborar unas conclusiones; de la misma forma se analizarán formas de mejora del sistema y se propondrán líneas a seguir para trabajos futuros.

6. Validación del diseño

Tomando como punto de partida el esquema presentado en la Ilustración 13 y con la configuración realizada en el apartado anterior, se va a probar la efectividad de las herramientas de monitorización, así como la posibilidad de reconfigurar la red sin afectar al funcionamiento de esta ante un intento de ataque de denegación de servicio proveniente de dentro de la misma red. Como se ha comentado previamente, una de las mayores amenazas en materia de seguridad a las que se exponen las SDN es a este tipo de ataques debido a su carácter centralizado.

Un ataque de denegación de servicio (DoS) es una técnica que tiene por objetivo incidir sobre la calidad de servicio de un sistema o una red pudiendo llegar a dejarlo en un estado no operativo. Con este objetivo, existen diferentes tipos de ataques DoS y se pueden a la vez, clasificar de múltiples formas [43]. Una de ellas se basa en el tipo de dato o efecto que provocan en el sistema, atendiendo a esta clasificación se destacan:

1. Saturación: es posiblemente el más común de todos. Consiste en agotar o saturar alguno de los recursos del sistema objetivo, siendo de especial interés aquellos recursos claves como el tiempo de CPU, memoria, ancho de banda…
2. Modificación de la configuración: como su nombre indica, el objetivo pasa por alterar o incluso eliminar la configuración de alguno de los elementos del sistema, los más usuales son servidores o routers. Esto desemboca en su inutilización y muy posiblemente en un estado inconsistente del sistema completo.
3. Destrucción: tiene como objetivo la destrucción o alteración física de alguno de los componentes del sistema. Debido al aumento de los sistemas inteligentes y de elementos en sistemas industriales con acceso a internet, este tipo de ataques no requiere el acceso físico a los componentes objetivos y es posible su explotación de manera remota.
4. Disruptivo: es otro de los más comunes, consiste en la interrupción de la comunicación entre dos dispositivos al alterar el estado de la información lo que lleva a hacer inviable la transferencia de información.
5. Obstrucción: esta técnica no es únicamente utilizada de forma maliciosa si no que a menudo se utilizada por motivos legales/judiciales. Trata de impedir la comunicación entre un usuario de un servicio y la víctima de manera que ya no puedan comunicarse entre sí adecuadamente.

Por otro lado, se encuentra la clasificación por nivel de capa OSI [44], esta clasificación divide los ataques DoS en dos grandes grupos:

1. Nivel de infraestructura: se incluyen todos los ataques contra la capa de red y de transporte. Los protocolos más atacados son TCP, UDP o ICMP debido a que son los más presentes. Destacan en este nivel según el protocolo que atacan:
   - TCP → Inundación SYN (SYN flood).
   - UDP → Inundación UDP (UDP flood).
   - ICMP → Inundación ICMP (ICMP flood).

2. Capa de aplicación: el objetivo de los ataques dirigidos contra esta capa es identificar y utilizar alguna vulnerabilidad en una aplicación para poder así terminar llevando a esta a una degradación e incluso a la interrupción total del servicio del sistema. El protocolo más atacado en esta capa es el HTTP aunque también son objeto de ataques otros como SMTP o DNS.

Explicado en detalle qué es un ataque de denegación de servicio y vistos los distintos tipos que existen, se tiene que analizar a cuál o a cuáles de ellos hay que prestar más atención en el sistema presentado en el actual trabajo. Atendiendo a la separación por capas del modelo de redes definidas por software (Ilustración 1), el sistema se compone de seis elementos situados en la capa de datos, los cuatro hosts, el sistema de detección de intrusiones y el switch virtual OVS. Por otro lado, el plano de control se compone de dos elementos, Faucet y Gauge mientras que, el plano de aplicación se compone básicamente de las herramientas para la recolección de estadísticas Prometheus y Grafana.

Parece obvio que según la clasificación de los ataques DoS en función de la capa OSI, requiere especial atención aquellos dirigidos hacia el nivel de infraestructura debido a que afecta a la mayoría de los elementos del sistema y es donde se sitúan aquellos más cruciales. Por lo que, entendido el riesgo, el siguiente paso será trazar la estrategia que reduzca en mayor medida el nivel de riesgo del conjunto del sistema.

Para estas pruebas es de suposición previa el que uno de los nodos o elementos del plano de datos se ha visto expuesto y está siendo utilizado por un tercero para intentar provocar un fallo en el sistema completo y causar la denegación del servicio. Para simular este tipo de ataque se va a utilizar la herramienta hping3 la cual, permite el envío masivo de paquetes de distintas clases en función al tipo de prueba que se quiera realizar.

El primer paso será la creación de una o varias reglas en el IDS que alerten de este tipo de ataques [45]. Para ello, se deberá atender a los distintos tipos de ataques señalados previamente. Antes de esto, se muestran las tablas de flujo previas a la implantación de cualquier regla de control de acceso para así mostrar los cambios que se introducen con cada regla dentro del switch.

```
tfgsdn@ubuntu:~$ dump-flows br0
 priority=9099,in_port="veth-snort" actions=drop
 priority=4096,in_port="veth-host1",vlan_tci=0x0000/0x1fff actions=push_vlan:0x8100,set_field:4196->vlan_vid,goto_table:1
 priority=4096,in_port="veth-host2",vlan_tci=0x0000/0x1fff actions=push_vlan:0x8100,set_field:4196->vlan_vid,goto_table:1
 priority=4096,in_port="veth-host3",vlan_tci=0x0000/0x1fff actions=push_vlan:0x8100,set_field:4196->vlan_vid,goto_table:1
 priority=4096,in_port="veth-host4",vlan_tci=0x0000/0x1fff actions=push_vlan:0x8100,set_field:4196->vlan_vid,goto_table:1
 priority=4096,in_port="veth-snort",vlan_tci=0x0000/0x1fff actions=push_vlan:0x8100,set_field:4196->vlan_vid,goto_table:1
 priority=0 actions=drop
 table=1, priority=20480,dl_vlan=100 actions=output:"veth-snort",goto_table:2
 table=1, priority=20479,dl_vlan=100 actions=goto_table:2
 table=1, priority=0 actions=drop
 table=2, priority=20490,dl_type=0x9000 actions=drop
 table=2, priority=20480,dl_src=ff:ff:ff:ff:ff:ff actions=drop
 table=2, priority=20480,dl_src=0e:00:00:00:00:01 actions=drop
 table=2, hard_timeout=260, priority=8191,in_port="veth-host2",dl_vlan=100,dl_src=ae:6f:8f:d1:19:d6 actions=goto_table:3
 table=2, hard_timeout=291, priority=8191,in_port="veth-host3",dl_vlan=100,dl_src=f6:cb:fe:f1:41:d6 actions=goto_table:3
 table=2, hard_timeout=272, priority=8191,in_port="veth-host4",dl_vlan=100,dl_src=32:f6:eb:a3:ea:52 actions=goto_table:3
 table=2, hard_timeout=290, priority=8191,in_port="veth-host1",dl_vlan=100,dl_src=6a:da:8a:bc:b0:ec actions=goto_table:3
 table=2, priority=4096,dl_vlan=100 actions=CONTROLLER:96,goto_table:3
 table=2, priority=0 actions=goto_table:3
```

Ilustración 16. Tablas de flujo sin reglas ACLs

Como se ve, la única regla aplicada hasta el momento es la que migra todos los paquetes de la red a la interfaz de Snort la cual se aplica en la tabla 1 con prioridad 20480. A continuación, se mostrará la evolución de estas tablas en función de las reglas de control de acceso que se irán añadiendo.

6.1. Protocolo TCP.

Debido a que se trata del protocolo más utilizado en internet, es también el que más tipos de ataques de denegación de servicio presenta. Sin embargo, se va a limitar la detección del más común de ellos, la inundación SYN. Es un ataque de denegación de servicio en el que el atacante envía un gran número de paquetes TCP SYN desde una o muchas direcciones de origen falsas al servidor. El servidor trata de responder a estos paquetes SYN enviando paquetes ACK-SYN a las direcciones IP falsificadas sin obtener respuesta alguna. Todas estas conexiones en espera se almacenan en una cola durante algún tiempo y cuando el número de conexiones en espera exceden el límite de la cola, todas las subsiguientes solicitudes SYN se rechazan, lo que conduce a la denegación del servicio a usuarios autorizados.

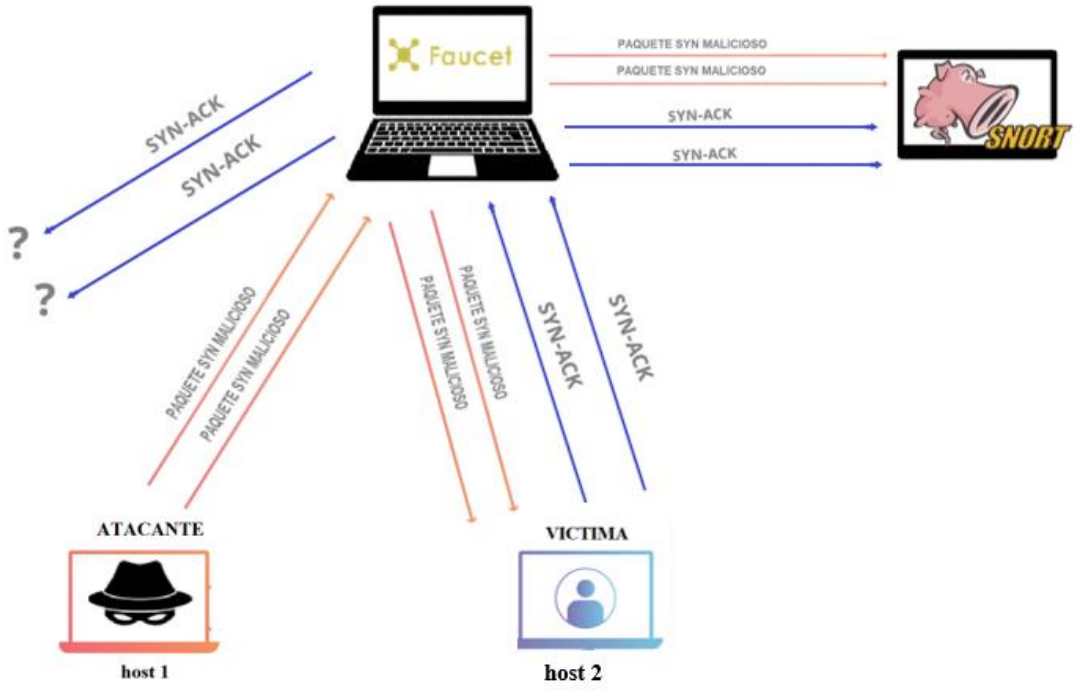


Ilustración 17. Gráfico ataque SYN-flooding

En la ilustración 17 se muestra un gráfico explicativo del transcurso de los paquetes durante el proceso de ataque. Como se puede ver, el atacante (host1) realiza el envío de paquetes TCP con el flag SYN activado de manera masiva hacia el host2. Esto provoca una sobrecarga en el controlador Faucet el cual es incapaz de conseguir responder a todas estas solicitudes. Al tratarse de un ataque de denegación de servicio no distribuido cuyo origen se sitúa dentro de la misma red, la dirección IP origen no será falsificada y muchas de las respuestas llegarán con éxito al nodo atacante. Sin embargo, debido a la cantidad ingente de estas peticiones, otras muchas no podrán ser respondidas correctamente provocando el rechazo de nuevas solicitudes.

Gracias a que la conexión con Snort se realiza en modo promiscuo, es decir, todo el tráfico que pasa por el controlador es migrado también a la interfaz asociada al IDS, es posible la detección temprana. A continuación, se explica el procedimiento seguido para detectar este tipo de ataque en Snort, para ello se ha creado la siguiente regla:

```
alert tcp any any -> $HOME_NET 80 (flags: S; msg:"POSIBLE ATAQUE DDOS
: SYN flood"; sid:10000002; rev:1; classtype: attempted-dos;
threshold: type both, track by_dst, count 1500, seconds 60;)
```

Donde por un lado la cabecera de la regla indica:

- Alert – la acción que se llevará a cabo cuando se active la regla. En este caso, snort generará una alerta cuando se cumpla la condición de la regla.
- Tcp – indica que el protocolo origen utilizado debe ser TCP.
- any any – indica que tanto la IP como el puerto origen puede ser cualquiera.
- $HOME_NET 80 – indica que la dirección destino pertenece a la red del sistema (HOME_NET es una variable de entorno indicada en el archivo de configuración de Snort) y que el puerto destino es el 80.

Por otro lado, las opciones de la regla son:

- Flags: S – se comprueba si el paquete tiene activado el flag TCP [46] SYN.
- Msg: “POSIBLE ATAQUE DDOS: SYN flood" – Es el mensaje que genera Snort junto a la alerta al activarse la regla.
- Sid: 10000002 – es el ID de la regla Snort. Para reglas locales se deben usar IDs superiores a 1000000 ya que las inferiores están reservadas.
- Rev: 1 – Es el número de revisión. Permite un mantenimiento más sencillo de la regla.
- Classtype: attepmted-dos – Categoriza la regla como un intento de DoS, el cual es un tipo predefinido en Snort lo cual ayudará a su posterior organización.
- Threshold: type both – Esta opción es utilizada debido a que, al estar trabajando con una base de datos, es necesario que Snort no genere una alerta por cada evento que cumpla con la regla establecida ya que, en un ataque DoS eso puede significar miles de alertas lo que terminaría colapsando la base de datos. Con esta opción se generará una única alerta cada vez que se cumpla la regla.

- Track by_dst – Con esta opción se indica que la cuenta del número de eventos registrados debe ser llevada únicamente por la dirección IP destino.
- Count 1500 – Es el número de veces que se debe cumplir la regla antes de que se genere la alerta.
- Seconds 60 – Se trata del intervalo en el cual se lleva a cabo la cuenta de eventos que cumplen la regla. Tras este intervalo el contador vuelve a 0.

Una vez añadida esta regla a Snort, puesto en marcha el sistema presentado en el apartado de implementación y partiendo del archivo de configuración de Faucet siguiente:

```
 acls:
       mirror-acl:
        - rule:
            actions:
                allow: True
                mirror: snort
vlans:
    general:
        vid: 100
        description: "Red General"
        acls_in: [mirror-acl]
dps:
    sw1:
        dp_id: 0x1
        hardware: "Open vSwitch"
        interfaces:
            1:
                name: "host1"
                description: "host1 network namespace"
                native_vlan: general
            2:
                name: "host2"
                description: "host2 network namespace"
                native_vlan: general

            3:
                name: "host3"
                description: "host3 network namespace"
                native_vlan: general

            4:
                name: "host4"
                description: "host4 network namespace"
                native_vlan: general

            6:
                name: "snort"
                description: "snort network namespace"
                native_vlan: general
```

Donde se ha definido una única regla ACL para migrar todo el tráfico de la red al puerto asociado al IDS Snort. El siguiente paso, será generar la simulación de ataque de inundación SYN. Para ello, se utilizará el host1 como nodo atacante y el host2 como nodo víctima. Para ello, haciendo uso de la herramienta hping3:

```
$as_ns host1 hping3 -p 80 -S --faster 192.168.2.3
```

La opción "-p 80" indica que el puerto destino donde realizar las solicitudes será el 80. A su vez, la opción "-S" activa el flag SYN y por último, con "--faster" se consigue que se envíen 100 paquetes por segundo.

Tras realizar este "ataque" contra el host2 y, debido a que todo el tráfico de la red pasa también por el IDS, la regla creada se activará. Una vez activada por Snort se generará un evento en el archivo de registro "/var/log/snortu2.*fecha*". A partir de este fichero, la herramienta barnyard2 toma el evento registrado y lo incluye en la base de datos Snort. A su vez, muestra por terminal el siguiente aviso:

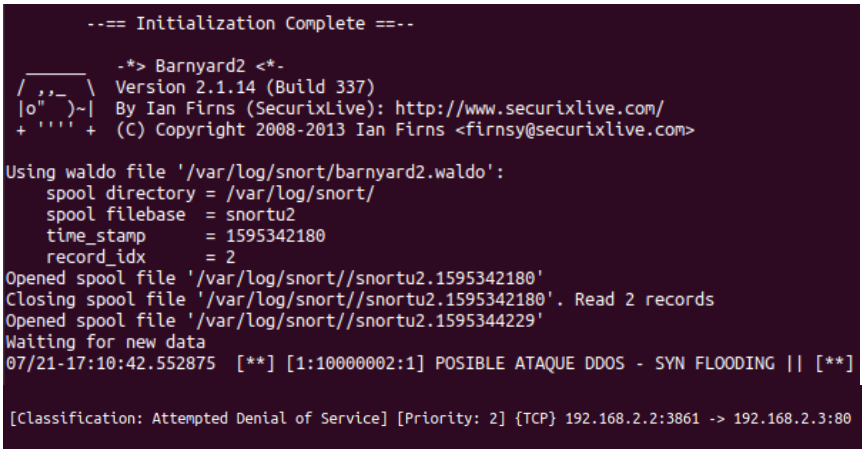


Ilustración 18. Aviso Barnyard2 Ataque SYN Flooding

Y desde Grafana, gracias al tablero creado (ver Anexo correspondiente) para la visualización general del estado del sistema, se observa lo siguiente:

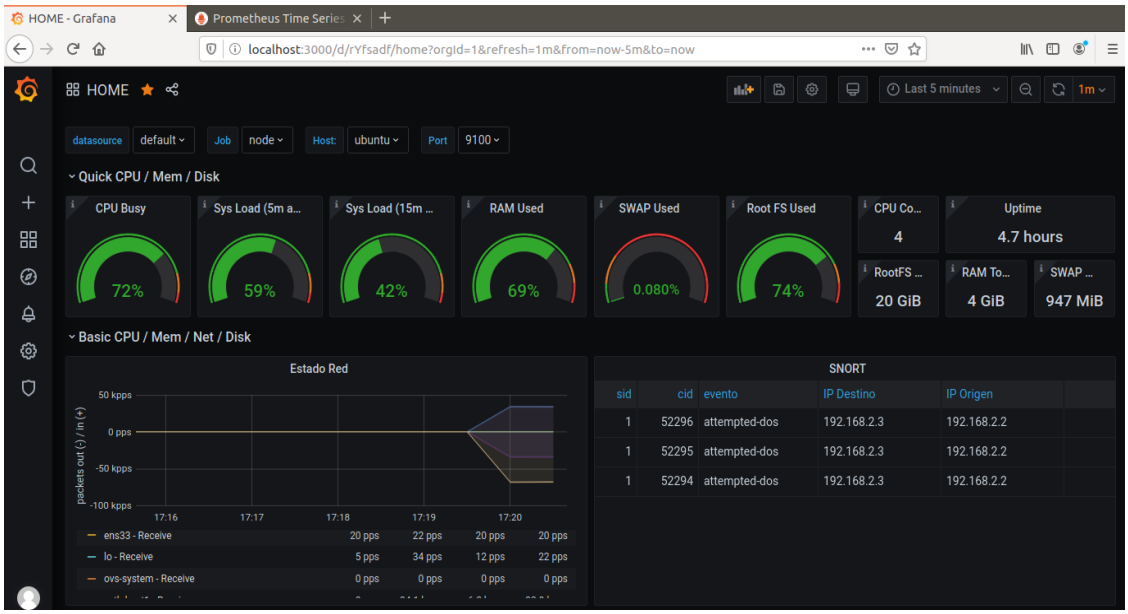


Ilustración 19. Vista desde Grafana del ataque SYN Flooding

Por un lado, se percibe como los niveles de uso de la CPU se encuentran en niveles elevados, además, en el gráfico de estado de la red, se puede contemplar un incremento significativo el cual es mayor en el número de paquetes enviados que en el de recibidos puesto que algunos de ellos, por saturación se han perdido no pudiendo llegar a su destino. Como último indicador clave, se tiene una abstracción de la propia base de datos Snort donde se pueden ver los eventos registrados. Esto último aporta más detalles concluyentes ya que indica que la regla creada se ha disparado puesto que el evento se trata de un intento de DoS y muestra las direcciones IPs tanto de destino como de origen.

Llegados a este punto, se tiene claro que se está produciendo un intento de ataque de denegación de servicio por lo que es necesario reaccionar antes esto antes de que el sistema falle por completo. Gracias al uso de las redes definidas por software y a su flexibilidad y programabilidad, esta tarea es relativamente sencilla en comparación a una red tradicional. Varias alternativas se plantean en estas circunstancias debido a que el nodo atacante se encuentra dentro de la misma red y ha sido, por tanto, un usuario legítimo en algún momento. Por lo que se decide por simplemente bloquear este tráfico proveniente del host1.

Para ello, se va a hacer uso de las reglas de control de acceso (ACLs) del controlador Faucet. Estas permiten, entre otras, bloquear el tráfico pudiendo diferenciar tanto por protocolos, puertos, dirección origen/destino, etc.

```
acls:
       mirror-acl:
        - rule:
            actions:
                allow: True
                mirror: snort

       block-SYN:
        - rule:
            dl_type: 0x800      # IPv4
            tcp_dst: 80         # Puerto destino
            ip_proto: 6         # TCP
            actions:
                allow: False
```

La nueva regla “block-SYN”, hará que el tráfico TCP (con número de protocolo 6 [47] generado y que tenga como destino el puerto 80 de cualquier dirección IPv4 de la red sea rechazado directamente. Esta es una regla con gran amplitud y que debe ser correctamente aplicada para evitar rechazar tráfico legítimo. Así pues, se aplicará esta ACL únicamente a la interfaz asociada al host1 ya que es sabido que es aquel el cual está generando este tráfico malicioso. Esto se consigue de la siguiente forma:

```
interfaces:
            1:
                name: "host1"
                description: "host1 network namespace"
                native_vlan: general
                acl_in: block-SYN
```

Una vez guardados los cambios en el archivo de configuración faucet.yaml, no es necesario detener el funcionamiento de la red para aplicarlos, mediante la orden:

```
tfgsdn@ubuntu: pkill -HUP -f faucet.faucet
```

Si se vuelven a observar las tablas de flujo, se puede observar cómo aparece un nuevo registro en la tabla 0, la cual es la asociada a las reglas aplicadas directamente a interfaces. En esta nueva entrada se muestra la regla que se ha creado ya que se indica que aquellos paquetes procedentes del puerto 1 (host1) y con destino el puerto tcp 80 sean automáticamente descartados.

```
tfgsdn@ubuntu:~$ dump-flows br0
 priority=20480,tcp,in_port="veth-host1",tp_dst=80 actions=drop
 priority=20480,in_port="veth-host2" actions=goto_table:1
 priority=20480,in_port="veth-host3" actions=goto_table:1
 priority=20480,in_port="veth-host4" actions=goto_table:1
 priority=20479,in_port="veth-host1" actions=goto_table:1
 priority=0 actions=drop
 table=1, priority=9099,in_port="veth-snort" actions=drop
 table=1, priority=4096,in_port="veth-host1",vlan_tci=0x0000/0x1fff actions=push_vlan:0x8100,set_field:4196->vlan_vid,goto_table:2
 table=1, priority=4096,in_port="veth-host2",vlan_tci=0x0000/0x1fff actions=push_vlan:0x8100,set_field:4196->vlan_vid,goto_table:2
 table=1, priority=4096,in_port="veth-host3",vlan_tci=0x0000/0x1fff actions=push_vlan:0x8100,set_field:4196->vlan_vid,goto_table:2
 table=1, priority=4096,in_port="veth-host4",vlan_tci=0x0000/0x1fff actions=push_vlan:0x8100,set_field:4196->vlan_vid,goto_table:2
 table=1, priority=4096,in_port="veth-snort",vlan_tci=0x0000/0x1fff actions=push_vlan:0x8100,set_field:4196->vlan_vid,goto_table:2
 table=1, priority=0 actions=drop
```

Ilustración 20. Tabla de flujo bloqueo SYN Flooding

Tras esto, si se intenta de nuevo realizar el ataque SYN flooding se comprobará, como se puede observar en la Ilustración 30, que ya no es posible el envío de estos paquetes por parte del host1. Sin embargo, si se intenta lo mismo desde cualquier otro host, sí que sería posible realizarlo.

```
tfgsdn@ubuntu:~$ sudo pkill -HUP -f faucet.faucet
tfgsdn@ubuntu:~$ as_ns host1 hping3 -p 80 --faster -S 192.168.2.3
HPING 192.168.2.3 (veth0 192.168.2.3): S set, 40 headers + 0 data bytes
^C
--- 192.168.2.3 hping statistic ---
1079410 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
tfgsdn@ubuntu:~$
```

Ilustración 21. SYN flooding bloqueado

6.2. Protocolo UDP.

Se trata de un ataque DoS en el que el atacante envía una avalancha de paquetes UDP falsificados a puertos aleatorios del servidor víctima. Como UDP es un protocolo sin conexión, no se requiere una negociación en tres pasos como el TCP. Al recibir un paquete UDP en un puerto determinado, se determina qué aplicación se está ejecutando en ese puerto y, si no hay ninguna aplicación disponible, se envía un paquete ICMP de "destino inalcanzable" a la dirección de origen falsificada. Si la víctima recibe un gran número de paquetes UDP, el rendimiento del sistema disminuye y puede llegar a no estar disponible.

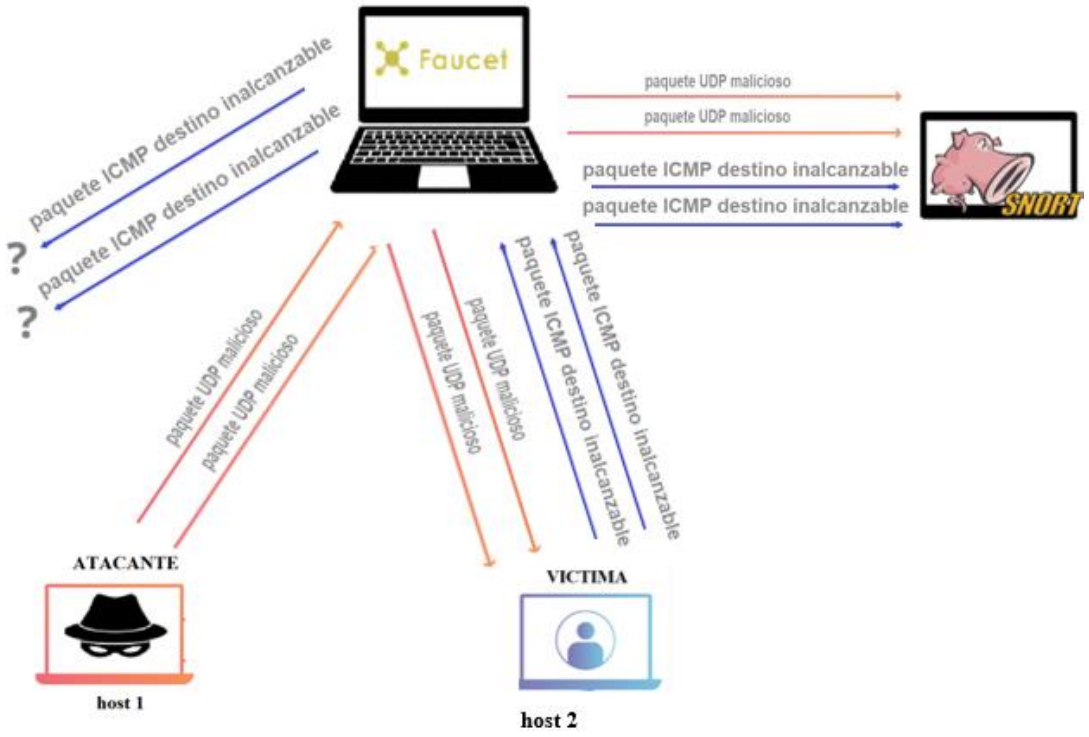


Ilustración 22. Gráfico ataque UDP-flooding

De nuevo, se incluye un gráfico explicativo del transcurso de los paquetes durante el proceso de ataque (ilustración 22). Al igual que en el ataque de inundación SYN, el atacante (host1) realiza el envío de paquetes UDP maliciosos de manera masiva hacia el host2. La reacción que provoca es la comentada en la explicación de este tipo de ataques. Si este no es detectado a tiempo, puede desembocar en que las solicitudes no puedan llegar a ser procesadas y los paquetes ICMP de respuesta no retornen al nodo que realizó esta solicitud. Si esto sucede, el atacante habría conseguido su cometido, provocar una denegación de servicio.

Para detectar este tipo de ataque en Snort se ha creado la siguiente regla:

```
alert udp any any -> $HOME_NET !53 (msg:"POSIBLE ATAQUE DDOS : UDP
flood"; sid:10000003; rev:1; classtype: attempted-dos; threshold: type
both, track by_dst, count 90000, seconds 60;)
```

Es similar a la regla para la inundación SYN, en este caso no hay que utilizar el campo flag debido a que no se trata de un paquete TCP y el número de eventos del count se incrementa debido a que se trata de un protocolo más simple que el TCP.

El siguiente paso, al igual que con el anterior ataque, será generar la simulación de ataque de inundación UDP. Para ello, se volverá a utilizar el host1 como nodo atacante y el host2 como nodo víctima. De la misma forma, haciendo uso de la herramienta hping3:

```
$as_ns host1 hping3 -2 --faster 192.168.2.3
```

La opción "-2" de la herramienta hping3 permite especificar UDP como el protocolo a utilizar para las solicitudes.

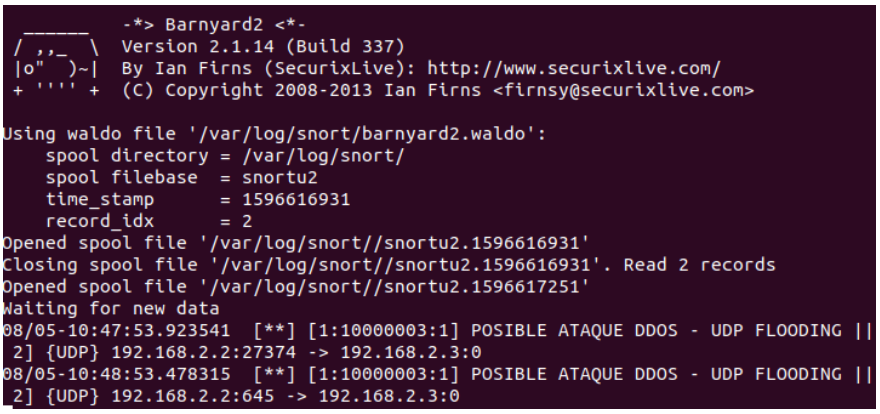


Ilustración 23. Aviso Barnyard2 ataque UDP flooding

De la misma forma que con el ataque SYN Flooding, la herramienta Barnyard2 genera un aviso por terminal al detectar la actividad sospechosa, al mismo tiempo se realiza el registro del aviso en la base de datos Snort. En el panel de Grafana para visualizar las estadísticas, se puede observar un comportamiento algo distinto al anterior. El porcentaje de uso de la CPU no es tan significativo como en el anterior ataque, esto se puede explicar ya que UDP no requiere una respuesta por parte del destinatario como sí pasa en TCP. Sin embargo, gracias al gráfico del estado de la red, así como a la vista de la base de datos, se percibe que se está produciendo un intento de ataque de denegación de servicio.

Ilustración 24. Vista en Grafana del ataque UDP flooding.

Llegados a este punto, se debe seguir el procedimiento anterior. Se debe crear una regla específica que bloquee este tipo de tráfico por parte del host1 pues es el nodo atacante. Cabe suponer que el anterior ataque y este no se han producido de manera consecutiva ya que llegados a este punto lo más lógico tras haber recibido dos ataques desde el mismo equipo sería sacar al mismo de la red.

```
acls:
       mirror-acl:
        - rule:
            actions:
                allow: True
                mirror: snort

       block-SYN:
        - rule:
            dl_type: 0x800       # IPv4
            tcp_dst: 80          # Puerto destino
            ip_proto: 6          # TCP
            actions:
                allow: False

       block-UDP:
        - rule:
            dl_type: 0x800
            ip_proto: 17         # UDP
            actions:
               allow: False
```

Se ha creado una nueva regla, esta vez para el protocolo UDP y de la misma forma que se hizo para el SYN Flooding, se aplicará a la interfaz asociada al host1.

```
interfaces:
            1:
                name: "host1"
                description: "host1 network namespace"
                native_vlan: general
                acls_in: [block-SYN, block-UDP]
```

Una vez se comprueba que la configuración es correcta y se aplican los cambios mediante el envío de la señal HUP, si se intenta realizar de nuevo el mismo ataque, se observará en este caso como no es posible y resulta en una pérdida de paquetes (Ilustración 25).

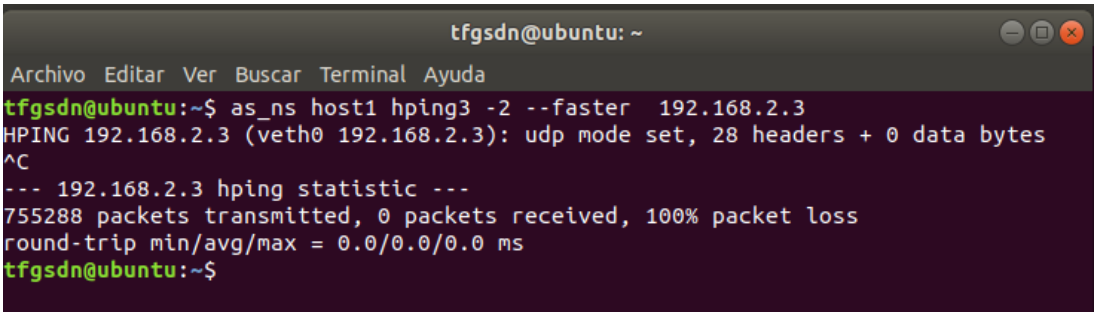


Ilustración 25. Intento fallido de ataque UDP Flooding

Además, si se observan de nuevo las tablas de flujo tras la aplicación de esta nueva regla, se puede ver como se ha creado un nuevo registro asociado a la tabla 0 para los paquetes UDP con procedencia el puerto asociado al host1.

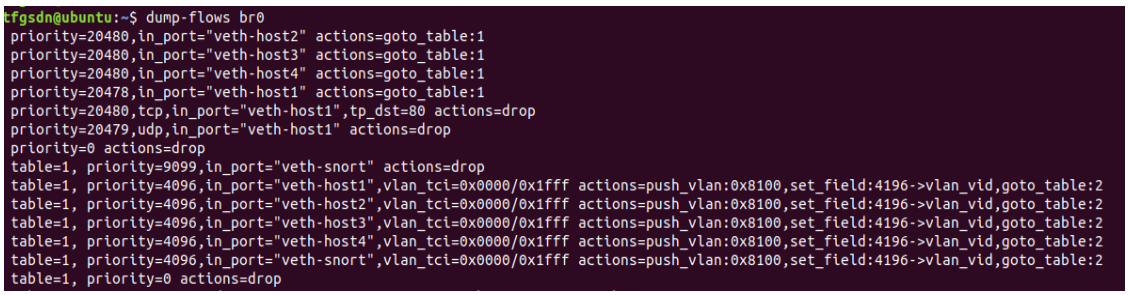


Ilustración 26. Tablas de flujo bloqueo UDP Flooding

6.3. Protocolo ICMP

Es otro de los más comunes debido a la sencillez de su procedimiento. Un ataque de inundación ICMP se ejecuta sobrecargando el servidor de la víctima con miles de peticiones de ping ICMP de IPs falsas. La víctima entonces usaría todos sus recursos para responder a estos mensajes ping hasta que ya no pueda procesar ninguna solicitud válida.

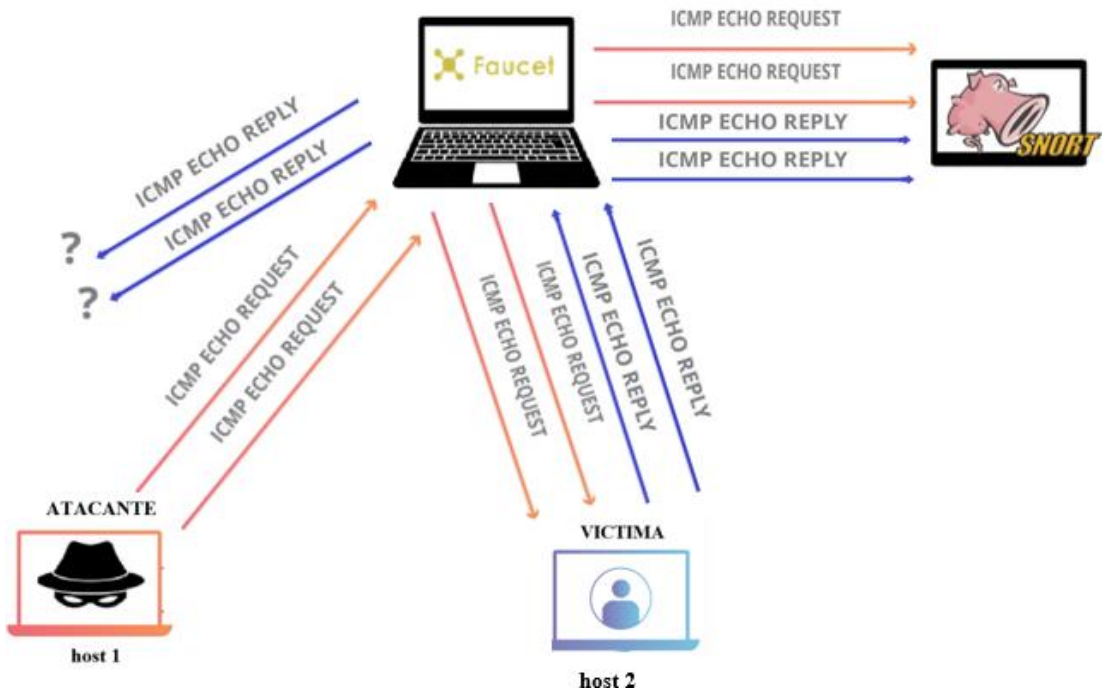


Ilustración 27. Gráfico ICMP flooding

En la Ilustración 27 se incluye de nuevo un gráfico con el fin de mostrar de manera más visual el procedimiento de este tipo de ataque. Al igual que con los dos ataques previos, la forma de actuar del atacante es similar salvo que en este caso utiliza solicitudes ICMP echo envíandolas de manera masiva hacia el host 2. Como en el resto de los ataques de denegación de servicio de los que se ha hablado el objetivo es conseguir saturar el controlador y provocar que no sea capaz de procesar tanto las solicitudes entrantes, así como las respuestas ICMP echo por parte de la víctima.

Para detectar este tipo de ataque en Snort y gracias a que todo el tráfico es migrado hacia el espacio de red en el que se encuentra en ejecución, se ha creado la siguiente regla:

```
alert icmp any any -> $HOME_NET any (msg:"POSIBLE ATAQUE DDOS : ICMP
flood"; itype:8; sid:10000001; rev:1; classtype: attempted-dos;
threshold: type both, track by_dst, count 90000, seconds 60;)
```

En este caso lo único que hay que tener en cuenta en especial con esta regla y para evitar falsos positivos, se indica, mediante el campo itype:8, que el paquete ICMP debe ser una solicitud eco.

Para el ataque, se sigue el procedimiento previo. Desde el host1 se realizará, utilizando la herramienta hping3, una simulación de ataque ICMP (opción -1) contra el host2 de la siguiente manera:

```
$as_ns host1 hping3 -1 --faster 192.168.2.3
```

Una vez realizado el ataque y teniendo el sistema de seguridad en funcionamiento, en el terminal donde se está ejecutando Barnyard2 se verá el siguiente aviso indicando que la regla asociada al ataque ICMP flooding previamente añadida, se ha visto activada. Al mismo tiempo, se crea una nueva entrada en la base de datos Snort.

Ilustración 28 Aviso Barnyard2 ataque ICMP flooding

Por otro lado, desde el panel de monitorización creado en Grafana, se detecta de manera muy visual este intento de ataque. Se puede apreciar tanto en la carga CPU ya que al tratarse de solicitudes ICMP echo, se produce una conversación pregunta-respuesta entre ambos hosts. De la misma forma, la gráfica del estado de la red muestra unos picos muy elevados, se pueden apreciar dos debido a que se ha lanzado dos veces el ataque para que el resultado sea más visual. Por último, se tiene como prueba de este intento de ataque la nueva entrada que ha aparecido en la base de datos Snort indicando un intento de denegación de servicio.

Ilustración 29. Vista en Grafana del ataque ICMP flooding

Como en los anteriores ataques, se debe gestionar este incidente antes de que provoque pérdidas de paquetes y el objetivo final, la caída del servicio. Para ello se crea una nueva regla de control de acceso y aplicarla a la interfaz asociada al host1:

```
acls:
        mirror-acl:
         - rule:
             actions:
                 allow: True
                 mirror: snort

        block-SYN:
         - rule:
             dl_type: 0x800      # IPv4
             tcp_dst: 80         # Puerto destino
             ip_proto: 6         # TCP
             actions:
                 allow: False

        block-UDP:
         - rule:
             dl_type: 0x800
             ip_proto: 17        # UDP
             actions:
                allow: False


        block-ICMP:
         - rule:
             dl_type: 0x800
             ip_proto: 1       #ICMP
             actions:
                allow: False

...
```

```
...

  interfaces:
            1:
                name: "host1"
                description: "host1 network namespace"
                native_vlan: general
                acls_in: [block-SYN, block-UDP, block-ICMP]
```

Tras esto, se debe comprobar que la configuración sea correcta y la sintaxis también y posteriormente enviar la señal HUB para que se aplique sin afectar al estado del sistema. Una vez se han realizado estos pasos, si se trata de nuevo de realizar el ataque ICMP flooding desde el host1 como anteriormente, se obtendrá el siguiente resultado:

```
tfgsdn@ubuntu:~$ as_ns host1 hping3 -1 --faster 192.168.2.3
HPING 192.168.2.3 (veth0 192.168.2.3): icmp mode set, 28 headers + 0 data bytes
^C
--- 192.168.2.3 hping statistic ---
745529 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
tfgsdn@ubuntu:~$
```

Ilustración 30. Intento fallido de ataque ICMP flooding

De nuevo, si acudimos a las tablas de flujo del switch, es posible observar al igual que en los ejemplos anteriores, la creación de un nuevo registro en las tablas 0 correspondiente a la regla creada para evitar este ataque ICMP flooding.

```
tfgsdn@ubuntu:~$ dump-flows br0
 priority=20480,in_port="veth-host2" actions=goto_table:1
 priority=20480,in_port="veth-host3" actions=goto_table:1
 priority=20480,in_port="veth-host4" actions=goto_table:1
 priority=20477,in_port="veth-host1" actions=goto_table:1
 priority=20480,tcp,in_port="veth-host1",tp_dst=80 actions=drop
 priority=20479,udp,in_port="veth-host1" actions=drop
 priority=20478,icmp,in_port="veth-host1" actions=drop
 priority=0 actions=drop
 table=1, priority=9099,in_port="veth-snort" actions=drop
 table=1, priority=4096,in_port="veth-host1",vlan_tci=0x0000/0x1fff actions=push_vlan:0x8100,set_field:4196->vlan_vid,goto_table:2
 table=1, priority=4096,in_port="veth-host2",vlan_tci=0x0000/0x1fff actions=push_vlan:0x8100,set_field:4196->vlan_vid,goto_table:2
 table=1, priority=4096,in_port="veth-host3",vlan_tci=0x0000/0x1fff actions=push_vlan:0x8100,set_field:4196->vlan_vid,goto_table:2
 table=1, priority=4096,in_port="veth-host4",vlan_tci=0x0000/0x1fff actions=push_vlan:0x8100,set_field:4196->vlan_vid,goto_table:2
 table=1, priority=4096,in_port="veth-snort",vlan_tci=0x0000/0x1fff actions=push_vlan:0x8100,set_field:4196->vlan_vid,goto_table:2
 table=1, priority=0 actions=drop
```

Ilustración 31. Tablas de flujo bloqueo ICMP Flooding

6.4. Análisis de resultados.

Tras la realización de los distintos tipos de ataque de denegación de servicio escogidos y demostrados los mecanismos de actuación frente a estos. El último de los pasos reside en analizar los resultados obtenidos a nivel de rendimiento y eficiencia de la arquitectura diseñada. Para este cometido se han escogido los siguientes parámetros con los que se analizará la capacidad de Snort de detectar los distintos ataques, así como la carga que supone al sistema sufrir cada uno de ellos.

1. **Eficiencia de Snort.** Se trata del número de paquetes analizados por segundo por parte del sistema de detección de intrusiones. Se mide en kilo paquetes por segundo (kpps) y la fórmula para su cálculo es la siguiente:

$$Eficiencia\ Snort = \frac{total\ paquetes\ analizados\ por\ Snort}{60 \times 1000}$$

(1) Fórmula cálculo eficiencia Snort

2. **Porcentaje de paquetes maliciosos no detectados.** Mediante este parámetro se evaluará la cantidad de paquetes maliciosos enviados que el IDS no ha sido capaz de analizar. Por lo que cuanto más alto es este porcentaje menor es el rendimiento de Snort.

$$\frac{total\ paquetes\ maliciosos\ enviados - \ paquetes\ analizados\ por\ Snort}{total\ paquetes\ maliciosos\ enviados}$$

(2) *Fórmula cálculo porcentaje de paquetes maliciosos no detectados*

3. **Uso CPU.** Corresponde al porcentaje de CPU utilizada bajo un ataque. Este valor será tomado directamente de Grafana gracias a los datos enviados con Prometheus.

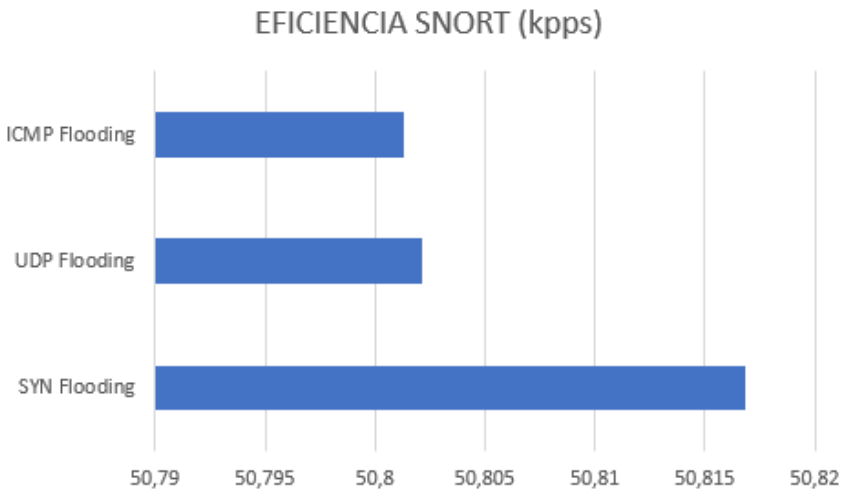


Ilustración 32. Gráfico eficiencia Snort

En este primer gráfico (Ilustración 32), se puede apreciar que en los tres casos el sistema de detección de intrusiones Snort presenta una eficiencia similar cuyos valores se comprenden entre 50,79 y 50,82 kilo paquetes por segundo. Este valor habría que compararlo con la eficiencia de Snort bajo condiciones de tráfico normal, donde se consiguen valores muy similares (entre 54 y 56 kpps). La conclusión que se obtiene es que pese a estar sometido a un ataque de denegación de servicio, el sistema de detección de intrusiones no ve afectada en gran medida su eficiencia, es decir, la cantidad de paquetes que procesa por segundo. Cabe destacar que para la recogida de estos datos se ha sometido al sistema a la misma cantidad de paquetes maliciosos enviados y que en la red local no existía ningún otro tipo de tráfico que no fuese el generado por el ataque.

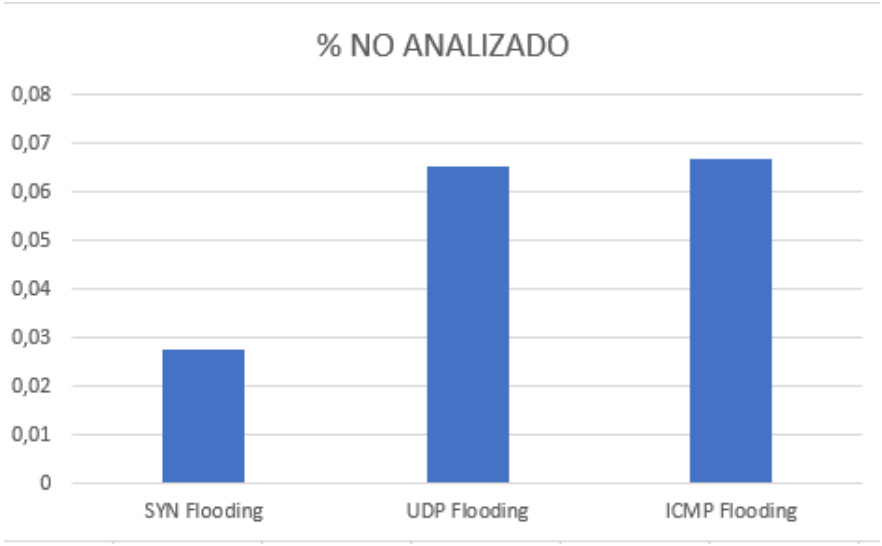


Ilustración 33. Gráfico % no analizado

En cuanto al porcentaje de paquetes maliciosos no analizados por Snort, como se puede apreciar en el gráfico de la ilustración 33, es considerablemente bajo. Esto, aunque un buen dato, no deja de ser relativo a las condiciones simuladas y se obtendrían resultados más interesantes bajo un entorno real. De nuevo se puede constatar mejores resultados para el tipo de ataque DoS SYN Flooding.

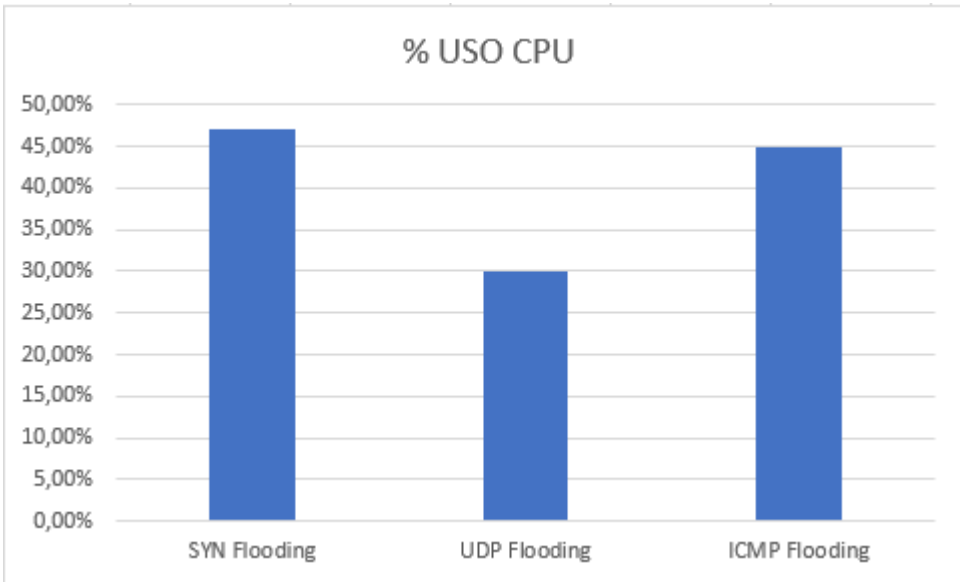


Ilustración 34. Gráfico % Uso CPU

Por último, el porcentaje de uso de CPU (ilustración 34) es uno de los datos más interesantes ya que no representa únicamente al sistema donde reside Snort si no que, debido a que todo se ha realizado bajo la misma máquina, representa el porcentaje de uso de la CPU del conjunto del sistema, lo que incluye tanto al controlador Faucet como al sistema de monitorización compuesto por Prometheus – Grafana – MySQL – Barnyard2, así como al switch virtual creado para la interconexión de los distintos hosts y, por supuesto tanto al nodo atacante como a la víctima. Pese a todo esto, se obtienen porcentajes de carga de la CPU dentro de la normalidad y en ningún momento alarmantes.

7. Conclusiones

La seguridad completa de un sistema es un objetivo inalcanzable y cuya obsesión puede desembocar en sistemas inestables e ineficientes. Teniendo esto en mente, mediante este trabajo se ha pretendido mostrar cómo, gracias a las características de las Redes Definidas por Software en combinación con elementos más tradicionales de seguridad como son los IDS, se pueden obtener resultados de gran interés a nivel de seguridad y de mantenimiento de la red.

Para ello durante la realización del presente proyecto se ha llevado a cabo un análisis en profundidad del estado de las SDN actuales, el cual se encuentra en continua evolución y aún existen muchas lagunas e información que no ha sido sometida a las pruebas necesarias para poder asegurar su confianza. Sin embargo, lo que sí se ha demostrado de manera clara es la necesidad de la investigación en la seguridad de estas.

También, se han estudiado los conceptos básicos en materia de seguridad y, viendo aquellos que afectan de primera mano al modelo SDN, se han presentado los sistemas IDS y en concreto se ha optado por el estudio de la herramienta Snort debido a su fama y confianza en este ámbito. Con esto, se ha creado un sistema que combina la seguridad propia del sistema, así como la capacidad de reacción ante un posible ataque gracias a los elementos de monitorización y análisis de la red utilizados como son Grafana, Prometheus y la base de datos MySQL.

Se ha diseñado e implementado una arquitectura que integra tecnologías diversas como son las SDN y un sistema IDS junto a las herramientas de monitorización previamente mencionadas, todo ello interconectado gracias al switch virtualizado Open vSwitch y al uso de los espacios de nombres de red, propios del entorno Linux. El resultado es un primer modelo el cual permite comprobar y profundizar en dos aspectos clave como son las redes programables y la seguridad en estas. Como muestra de ello, se han realizado una serie de pruebas sometiendo al sistema a diferentes ataques de denegación de servicio analizando y demostrando la capacidad de reacción frente a estos. Por su parte, la obtención de unos resultados menos completos de lo esperado (limitados a la red local) es, en parte debido a que las redes definidas por software se tratan de una tecnología ajena en muchos sentidos a los conocimientos cursados durante la carrera y a su vez de un campo con tantas posibilidades que es difícil acotarlo a la extensión óptima de este trabajo académico. Por lo que se podría considerar como la primera parte de un futuro proyecto más completo y ambicioso.

Pese a esto, en este trabajo se ha profundizado sobre otros ámbitos que sí han sido tratados durante el curso, la seguridad en Redes, el modelo de Red, los distintos protocolos, etc. Ha sido sin duda enriquecedor el haber podido indagar de una forma menos superficial en algunos de estos terrenos, así como de descubrir otros igual de apasionantes. De la misma forma, es interesante pensar que este trabajo puede servir para acercar el campo de las Redes Definidas por Software y el proyecto Faucet, a usuarios ajenos a estas y que puedan, si les interesa, investigar por su cuenta y adentrarse en la multitud de proyectos de código abierto que existen actualmente.

8. Trabajos futuros

Como se ha dicho, las SDN no son ya el futuro si no que son el presente mismo. El objetivo ahora pasa por que la expansión de los despliegues de esta tecnología, desde redes de campus a redes de área amplia, se vea acompañado por consideraciones de seguridad cada vez más complejas. Es importante que tanto los diseñadores de controladores SDN como los desarrolladores de protocolos para la interfaz norte (NBI) consideren firmemente la seguridad en sus diseños. Una de estas consideraciones podría pasar por el uso de transacciones seguras de aplicación-red lo cual complementaría una potencial transacción segura de plano de control-datos (con autenticación mutua), esto junto a transacciones seguras de control-control daría lugar a un modelo SDN seguro completo de varias capas.

Otro de los caminos futuros que se dibujan en materia de seguridad y SDN y que, pese a que su uso en el actual trabajo no ha sido posible por cuestiones de extensión, es uno de los más interesantes y con mayor proyección; se trata del uso del aprendizaje automático y el análisis de datos. Entrenar al sistema actual presentado en el trabajo utilizando los eventos recogidos y mediante técnicas de aprendizaje automático, desembocaría en que el propio sistema podría ser capaz de adaptarse a diferentes escenarios obteniendo mejoras tanto en la calidad de servicio como en seguridad. Ya existen algunas propuestas relacionadas con este tema que son muy prometedoras. Por ejemplo, el proyecto Poseidon presenta la combinación de Faucet como controlador SDN junto al aprendizaje automático para dar lugar a un sistema inteligente y reactivo.

9. Glosario

- **SDN:** software defined network.
- **IDS:** intrusión detection system.
- **VLAN:** virtual local area network.
- **DoS:** deny of service.
- **NBI:** northbound interface.
- **SBI:** southbound interface.
- **HUB:** concentrador.
- **ICMP:** internet control message protocol.
- **TCP:** transmission control protocol.
- **UDP:** user datagram protocol.
- **ACL:** access control list.
- **CPU:** central processing unit.

10. Referencias

1. ATOS SE. (2019, abril). Edge computing. Recuperado de https://atos.net/wp-content/uploads/2019/04/Atos_EdgeComputing_WP-web.pdf

2. Cooney, M. (2019, 6 junio). Juniper: Security could help drive interest in SDN. Recuperado de https://www.networkworld.com/article/3400739/juniper-sdn-snapshot-finds-security-legacy-network-tech-impacts-core-network-changes.html
3. de Nicola, M. (2019, 31 marzo). Redes Definidas por Software - SDN 2.0. Recuperado de https://www.ciberseguridadlatam.com/2019/03/31/redes-definidas-por-software-sdn-2-0/
4. VMWare. (s. f.). Software-Defined Networking (SDN). Recuperado de https://www.vmware.com/topics/glossary/content/software-defined-networking
5. Rambla, J. L. G., Alonso, C., Cebrián, J. M. A., & González, P. (2017). Ataques en redes de datos IPv4 e IPv6. Madrid, España: ZeroxWord Computing.

6. Gómez Climent, M. G. C. (2016, 14 septiembre). Seguridad y prioridad en redes diseñadas por software. Recuperado de http://hdl.handle.net/10251/72498

7. Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. OpenFlow: enabling innovation in campus networks. SIGCOMM Comput. Commun. Rev. 38, 2 (April 2008), 69–74. DOI: https://doi.org/10.1145/1355734.1355746

8. Gude, Natasha & Koponen, Teemu & Pettit, Justin & Pfaff, Ben & Casado, Martin & McKeown, Nick & Shenker, Scott. (2008). NOX: Towards an operating system for networks. Computer Communication Review. 38. 105-110.

9. Kesden, G. (2014, 28 marzo). Software Defined Networking (SDN) [Diapositivas]. Recuperado de https://cseweb.ucsd.edu/classes/fa16/cse291-g/applications/ln/SDN.pdf

10. Cisco. (2019, 14 enero). OpFlex: An Open Policy Protocol White Paper. Recuperado de https://www.cisco.com/c/en/us/solutions/collateral/data-center-virtualization/application-centric-infrastructure/white-paper-c11-731302.html

11. SDxCentral. (2014, 15 julio). What is Cisco OpenFlow? Recuperado de https://www.sdxcentral.com/networking/sdn/definitions/cisco-openflow/

12. Jain, A. (2017, 27 febrero). Partnering toward the next generation of mobile networks. Recuperado de https://blog.google/inside-google/infrastructure/partnering-toward-next-generation-mobile-networks/

13. Terry-Brown, P., Solis, B., May, A., Prats, J., Casalta, S., & Hassim, F. (2019). Exploring the vital importance of software-defined networking (SDN) as the foundation of digital transformation. Recuperado de

https://www.vodafone.com/business/news-and-insights/white-paper/SDN-research-report-2019

14. Telefónica. (s. f.). Virtual Private Networks (VPN) y Software Defined Networks (SDN). Recuperado de https://www.movistar.es/grandes-empresas/soluciones/fichas/vpn-y-sdn/

15. Latif, Z., Sharif, K., Li, F., Monjorul Karim, M., Biswas, S., & Wang, Y. (2020, 15 abril). A comprehensive survey of interface protocols for software defined networks. ScienceDirect, 156(2020). Recuperado de https://www.sciencedirect.com

16. Zhu, S. Y., Scott-Hayward, S., Jacquin, L., & Hill, R. (2017). Guide to Security in SDN and NFV: Challenges, Opportunities, and Applications. Nueva York, EEUU: Springer International Publishing.

17. IBM Services. (2019, 1 noviembre). SDN Versus Traditional Networking Explained. Recuperado de https://www.ibm.com/services/network/sdn-versus-traditional-networking

18. Jacobs, D. C. B. (2012, 20 febrero). OpenFlow protocol primer: Looking under the hood. Recuperado de https://searchnetworking.techtarget.com/feature/OpenFlow-protocol-primer-Looking-under-the-hood

19. McKeown, Nick & Anderson, Tom & Balakrishnan, Hari & Parulkar, Guru & Peterson, Larry & Rexford, Jennifer & Shenker, Scott & Turner, Jonathan. (2008). OpenFlow: Enabling innovation in campus networks. Computer Communication Review. 38. 69-74. 10.1145/1355734.1355746.

20. Open Networking Foundation. (2014, diciembre). Switch OpenFlow [Ilustración]. Recuperado de https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf

21. TechTarget. (2015, junio). Vendors take alternatives to OpenFlow SDN. Recuperado de https://searchnetworking.techtarget.com/essentialguide/Vendors-take-alternatives-to-OpenFlow-SDN

22. E. Haleplidis et al., "Network Programmability With ForCES," in IEEE Communications Surveys & Tutorials, vol. 17, no. 3, pp. 1423-1440, 2015, doi: 10.1109/COMST.2015.2439033.

23. Internet Engineering Task Force. (2016, 25 abril). OpFlex Control Protocol. Recuperado de https://tools.ietf.org/html/draft-smith-opflex-03

24. Internet Engineering Task Force (IETF). (2011, noviembre). “Network Configuration Protocol (NETCONF)”, RFC 6241. Recuperado de https://tools.ietf.org/html/rfc6241

25. Zhu, Liehuang & Karim, Md Monjurul & Sharif, Kashif & Li, Fan & Du, Xiaojiang & Guizani, Mohsen. (2019). SDN Controllers: Benchmarking & Performance Evaluation.

26. SDxCentral. (2008). NOX. Recuperado de https://www.sdxcentral.com/projects/nox/

27. POX Manual Current documentation. (2015). POX Manual. Recuperado de https://noxrepo.github.io/pox-doc/html/

28. SDxCentral. (2014, 15 septiembre). What is a Floodlight Controller? Recuperado de https://www.sdxcentral.com/networking/sdn/definitions/what-is-floodlight-controller/

29. OpenDaylight Project. (2019, 26 marzo). Platform Overview. OpenDaylight. Recuperado de https://www.opendaylight.org/what-we-do/odl-platform-overview

30. Open Networking Foundation. (2014, 29 octubre). ONOS - Wiki. Onos Project. Recuperado de https://wiki.onosproject.org/display/ONOS/ONOS

31. Davenport, P. (2019, 12 noviembre). What is "carrier-grade" exactly? Recuperado de https://www.appneta.com/blog/what-is-carrier-grade-exactly/

32. Ryu documentation. (s. f.). Ryu Docs. Recuperado de https://ryu.readthedocs.io/en/latest/getting_started.html

33. INCIBE. (2017, 20 marzo). Amenaza vs Vulnerabilidad, ¿sabes en qué se diferencian? Recuperado de https://www.incibe.es/protege-tu-empresa/blog/amenaza-vs-vulnerabilidad-sabes-se-diferencian

34. Murillo, Andrés & Rueda, Sandra & Morales, Laura & Cardenas, Alvaro. (2017). SDN and NFV Security: Challenges for Integrated Solutions. 10.1007/978-3-319-64653-4_3.

35. Red Hat, Inc. (2005). Detección de intrusos. Recuperado de http://web.mit.edu/rhel-doc/4/RH-DOCS/rhel-sg-es-4/ch-detection.html

36. J. Yost. (2016). The March of IDES: Early History of Intrusion-Detection Expert Systems. En IEEE Annals of the History of Computing (Vol. 38, pp. 42-54). https://doi.org/10.1109/MAHC.2015.41

37. Pandini, W. (2018, 2 enero). IDS: Historia, concepto y terminología. Recuperado de https://ostec.blog/es/seguridad-perimetral/ids-conceptos

38. Faucet. (s. f.). Arquitectura Faucet. Recuperado de https://docs.faucet.nz/en/latest/architecture.html

39. Scott, L. (2016, 14 octubre). Dissecting the Faucet v1.1 Pipeline. Recuperado de https://inside-openflow.com/2016/09/16/dissecting-faucet-pipeline/

40. Jelkovich, Y. (2014, agosto). What is a Sniffer and How to Protect your Data Against Sniffing. Recuperado de http://www.e-articles.info/e/a/title/What-is-a-Sniffer-and-How-to-Protect-your-Data-Against-Sniffing/

41. A Survey on the Contributions of Software-Defined Networking to Traffic Engineering - Scientific Figure en ResearchGate. Disponible desde: https://www.researchgate.net/figure/Main-components-of-the-Open-vSwitch-OVS-architecture-placed-in-the-physical-kernel-or_fig7_311338103

42. man7. (s. f.). network_namespaces(7) - Linux manual page. Recuperado de https://www.man7.org/linux/man-pages/man7/network_namespaces.7.html

43. INCIBE. (2019, 8 octubre). Clasificación de ataques DoS. INCIBE-CERT. https://www.incibe-cert.es/blog/clasificacion-ataques-dos

44. H. Zimmermann, "OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection," in IEEE Transactions on Communications, vol. 28, no. 4, pp. 425-432, April 1980.

45. Gupta, Alka & Sharma, Lalit. (2019). Mitigation of DoS and Port Scan Attacks Using Snort. International Journal of Computer Sciences and Engineering. 7. 248-258. 10.26438/ijcse/v7i4.248258.

46. Administración avanzada de redes TCP/IP. Javier Carmona Murillo, David Cortés Polo, M. Domínguez-Dorado, Alfonso Gazo-Cervero, José-Luis González-Sánchez, Francisco Javier Rodríguez Pérez. ISBN 978-84-695-2037-6. January, 2012.

47. Internet Assigned Numbers Authority. (2020, 31 enero). Protocol Numbers. Protocol Numbers. https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml

11. ANEXOS

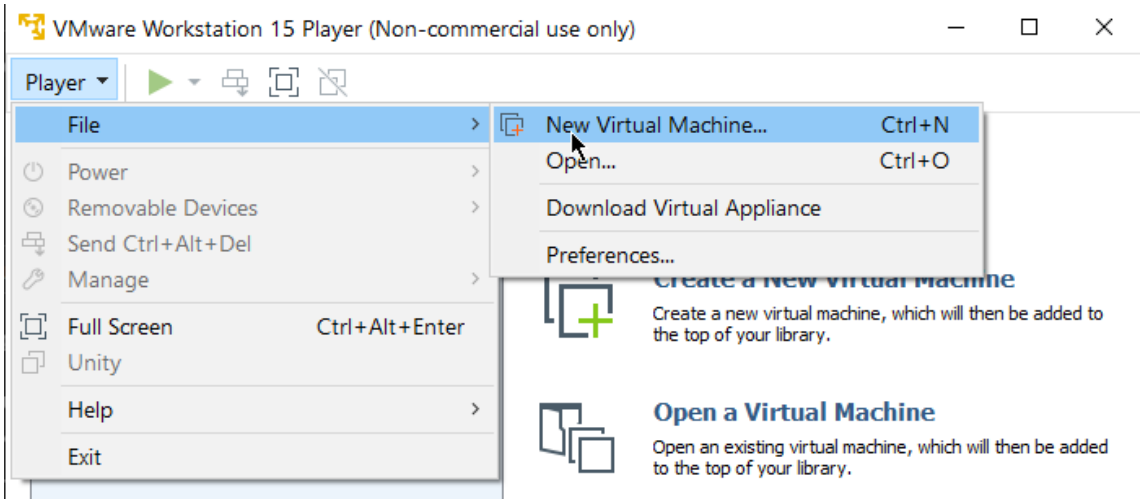
1. INSTALACIÓN Y CONFIGURACIÓN DE VMWARE Y UBUNTU.

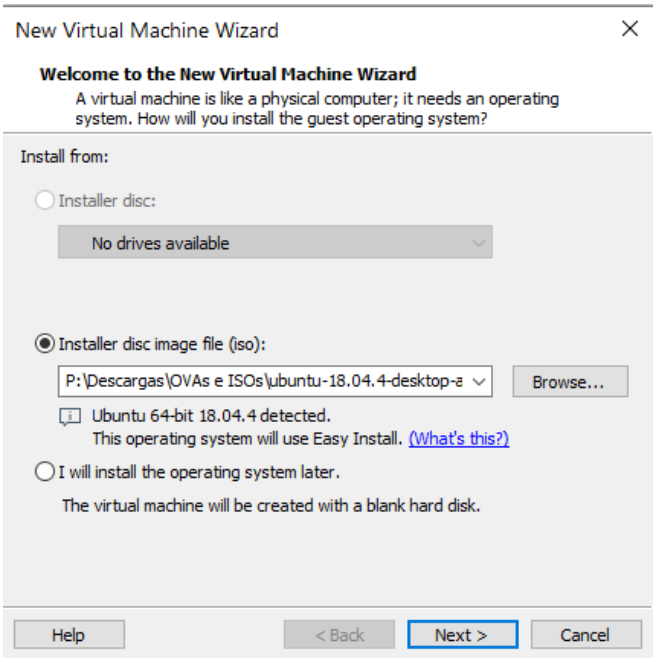
Bajaremos el fichero VMware-player-15.5.6-16341506.exe desde la web oficial https://www.vmware.com/products/workstation-player/workstation-player-evaluation.html

Clicamos y ejecutamos el archivo. Se abrirá el asistente de instalación, completaremos el proceso y al acabar VMware se abrirá automáticamente.

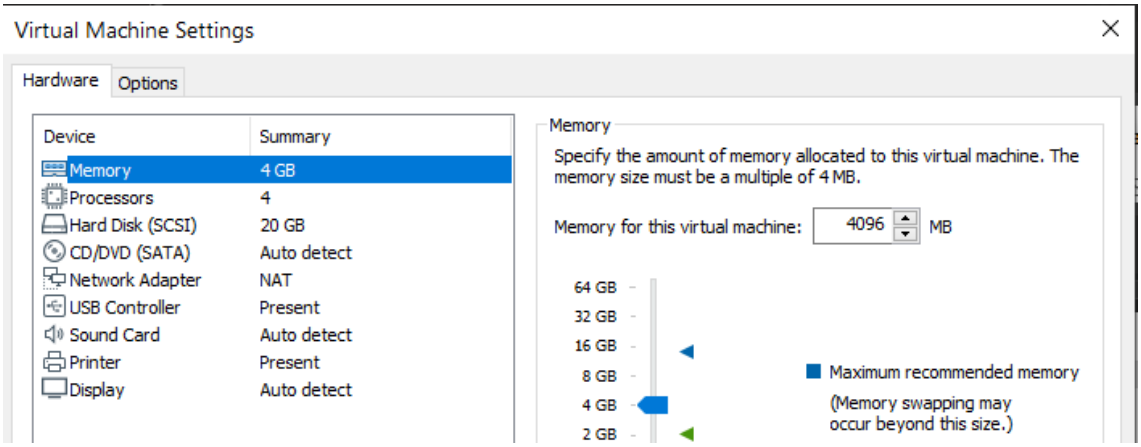
Una vez instalado VMware, bajamos la imagen .iso de Ubuntu 18.04 LTS desde https://releases.ubuntu.com/18.04/

Con la imagen descargada, vamos a VMware y creamos la máquina virtual de la siguiente manera:





Se nos solicitará introducir el nombre de la máquina, nombre de usuario y contraseña. Una vez hecho, debemos gestionar los recursos propios de la máquina ya que debido a la multitud de herramientas a utilizar es necesario que esta tenga un mínimo para gestionarlas sin problemas.

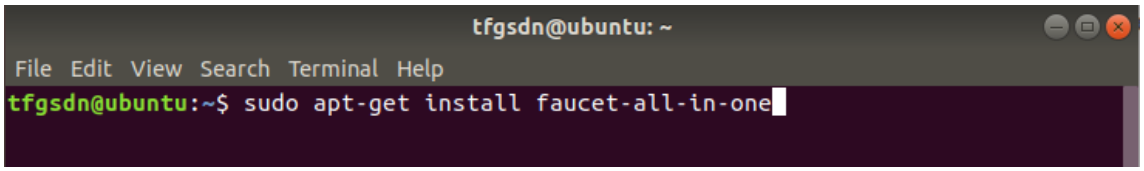


Arrancamos la máquina y una vez iniciada, actualizamos el sistema ante posibles actualizaciones de seguridad no instaladas. Para ello:

```
tfgsdn@ubuntu sudo apt-get update && sudo apt-get upgrade -y
```

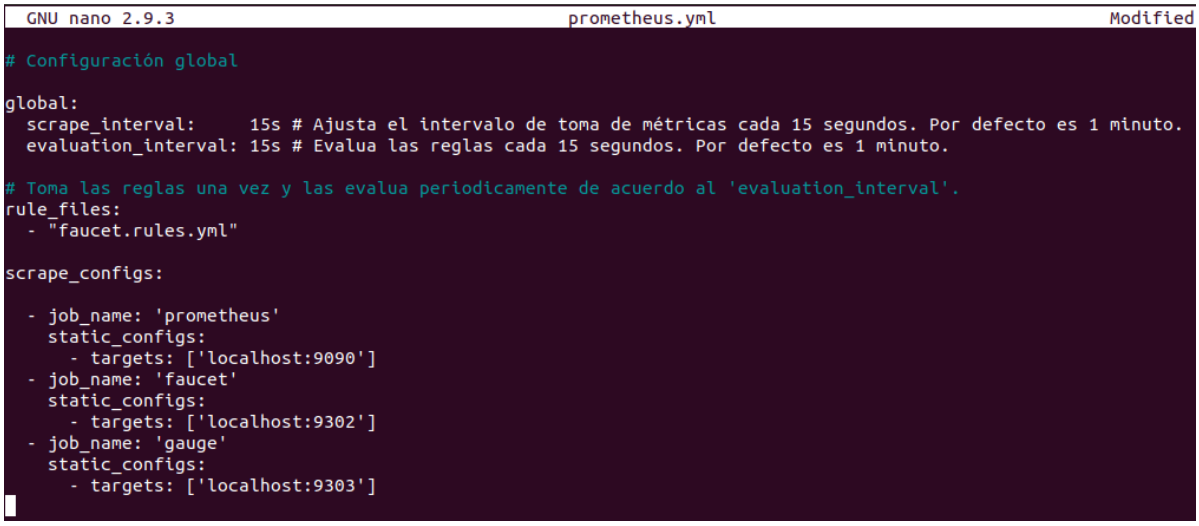
2. INSTALACIÓN Y CONFIGURACIÓN FAUCET.

El primer paso será descargar el meta paquete “Faucet-all-in-one”.

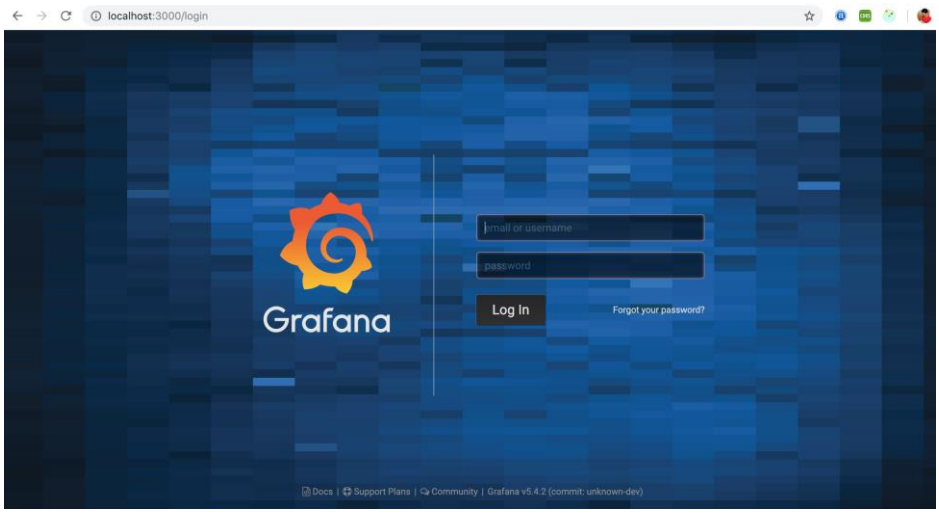


Tras la instalación se puede comprobar cómo, a parte de los controladores Faucet y Gauge, se han instalado las herramientas Prometheus y Grafana en el sistema; así como todas las dependencias necesarias para el correcto funcionamiento de estas. El siguiente paso será el de configurar correctamente cada una de las tecnologías.

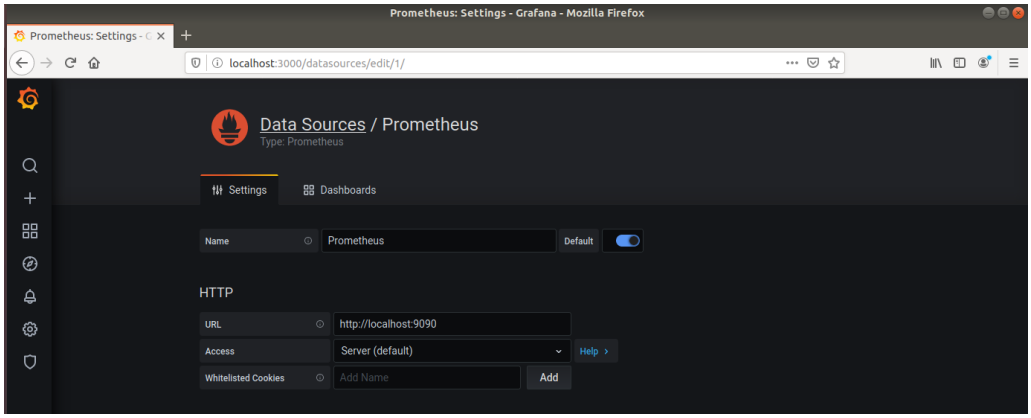
En primer lugar, es necesario configurar Prometheus para que interprete y procese correctamente los datos recibidos tanto por Faucet como por Gauge. Para ello, se requiere modificar el fichero de configuración *prometheus.yml* de forma que quede de la siguiente manera:



Configurado Prometheus, es necesario hacer lo propio con Grafana. A diferencia de la anterior herramienta, Grafana es configurada mediante su aplicación web. Una vez se accede a ella mediante la dirección “http://localhost:3000” con las credenciales admin/admin, por defecto.



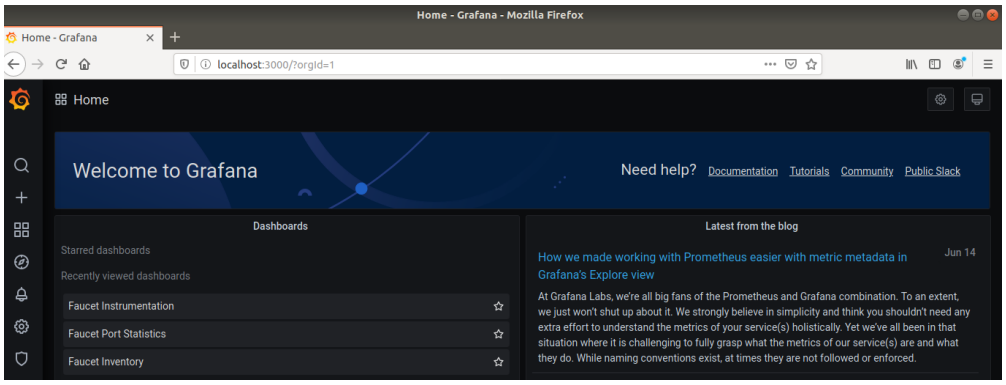
El siguiente paso es añadir Prometheus como fuente de datos y así poder visualizar las métricas del sistema procesadas por esta. Para ello, se tiene que añadir una nueva fuente de datos e indicar que se trata de Prometheus; es necesario indicar la dirección utilizada por Prometheus para exponer las métricas que procesa, esta ha sido configurada previamente "http://localhost:9090".



Teniendo la conexión Faucet – Prometheus – Grafana resuelta, lo siguiente será añadir lo que en Grafana se llaman Tableros de Control o *Dashboards* que son los que permiten la visualización de los diferentes datos provenientes de Faucet. En concreto se tratan de tres tableros:

- El primero de ellos muestra toda la información relacionada con el uso de CPU, uso de Memoria y estado de la configuración.
- El segundo indica los elementos que se encuentran activos en el sistema tanto los del plano de datos como los controladores.
- El último tablero estará relacionado con las estadísticas de los puertos del plano de datos, bits entrantes/salientes, paquetes entrantes/salientes, paquetes perdidos…

Para añadir estos tres tableros, será necesario descargarlos de la página oficial de Faucet e importarlos en la web de Grafana.



Es el turno de configurar los controladores, en primer lugar, se tiene que modificar el archivo de configuración de Faucet del que se ha hablado anteriormente: "*faucet.yml*".

```
tfgsdn@ubuntu: /etc/faucet
File  Edit  View  Search  Terminal  Help
tfgsdn@ubuntu:/etc/faucet$ cat faucet.yaml
acls:
    mirror-acl:
        - rule:
            actions:
                allow: true
                mirror: snort
vlans:
    general:
        vid: 100
        description: "Red General"

dps:
    sw1:
        dp_id: 0x1
        hardware: "Open vSwitch"
        interfaces:
            1:
                name: "host1"
                description: "host1 network namespace"
                native_vlan: general
            2:
                name: "host2"
                description: "host2 network namespace"
                native_vlan: general

            3:
                name: "host3"
                description: "host3 network namespace"
                native_vlan: general

            4:
                name: "host4"
                description: "host4 network namespace"
                native_vlan: general

            6:
                name: "snort"
                description: "snort network namespace"
                native_vlan: general
```

La imagen anterior, muestra la configuración implementada, esta cuenta con una sola regla ACL para que todos los paquetes de la red sean enviados a Snort independientemente de su destino original. También se ha configurado una única VLAN para crear una red lo más sencilla posible. Por último, se ha creado una ruta de datos la cual está formada por un switch

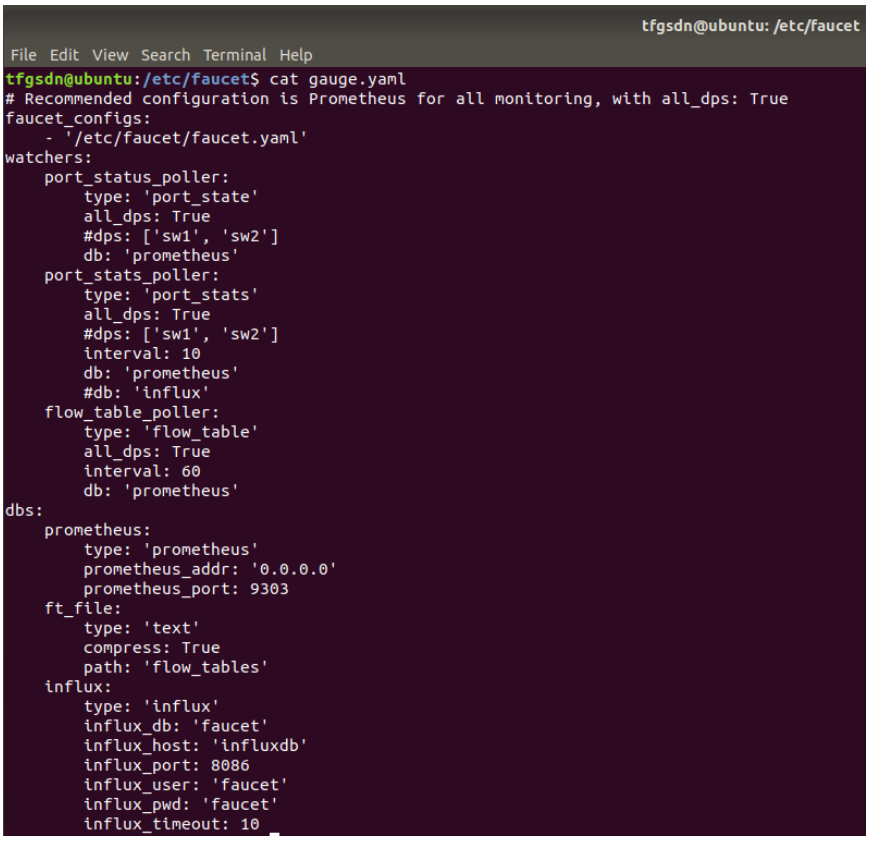
(sw1), este a su vez cuenta con 5 interfaces, 4 de ellas para hosts y la última será la utilizada por el IDS Snort.

Para comprobar si esta configuración es correcta se utiliza el comando:

```
$check_faucet_config /etc/faucet/faucet.yaml
```

Si todo es correcto, este comando devolverá un objeto JSON con toda la configuración de Faucet, tanto la modificada como la que permanece por defecto.

Por último, quedaría por configurar el controlador Gauge, sin embargo, su configuración por defecto será la utilizada en las distintas pruebas a realizar. A modo de explicación se muestra en la siguiente imagen, el archivo de configuración *gauge.yml*.



Como se puede observar, permite la exportación de las estadísticas y del estado de funcionamiento del sistema en tres formatos: Prometheus, archivo de texto plano y InfluxDB. Para este ejemplo se ha optado por tan solo utilizar la exportación a Prometheus.

3. INSTALACIÓN Y CONFIGURACIÓN SNORT.

Antes de pasar con la instalación de Snort, es necesario instalar los prerrequisitos de Snort, estos son 4 principales: PCAP, PCRE, Libdnet, DAQ.

Para su instalación, primero es necesario obtener el paquete *build-essentials* el cual instalará las herramientas necesarias para construir nuestro software:

```
tfgsdn@ubuntu sudo apt-get install -y build-essential
```

A continuación, se instalarán tanto PCAP, PCRE como Libdnet desde el. repositorio de Ubuntu

```
tfgsdn@ubuntu sudo apt-get install -y libpcap-dev libpcre3-dev
libdumbnet-dev
```

Para acabar, se instalará el último de los prerrequisitos, DAQ, para ello:

```
tfgsdn@ubuntu sudo apt-get install -y bison flex
tfgsdn@ubuntu wget https://www.snort.org/downloads/snort/daq-
2.0.6.tar.gz
tfgsdn@ubuntu tar -xvzf daq-2.0.7.tar.gz
tfgsdn@ubuntu cd daq-2.0.7
tfgsdn@ubuntu./configure
tfgsdn@ubuntu make
tfgsdn@ubuntu sudo make install
```

De manera adicional, es necesario instalar el paquete zlibg ya que se trata de una biblioteca de compresión que nos permitirá configurar Snort correctamente.

```
tfgsdn@ubuntu sudo apt-get install -y zlib1g-dev
```

Cumplidos estos pasos previos, pasamos a instalar Snort. Se ha escogido la versión Snort 2.9.16 ya que es la última versión estable, lanzada el 15 de marzo de 2020:

```
tfgsdn@ubuntu wget https://snort.org/downloads/snort/snort-
2.9.16.tar.gz
tfgsdn@ubuntu tar -xvzf snort-2.9.16.tar.gz
tfgsdn@ubuntu cd snort-2.9.16
tfgsdn@ubuntu ./configure --enable-sourcefire
tfgsdn@ubuntu make
tfgsdn@ubuntu sudo make install
```

A continuación, para acabar con la instalación de Snort, es necesario actualizar las librerías compartidas y crear un enlace simbólico al binario de Snort en /usr/sbin:

```
tfgsdn@ubuntu sudo ldconfig
tfgsdn@ubuntu sudo ln -s /usr/local/bin/snort /usr/sbin/snort
```

Lo siguiente será configurar tanto Snort como los archivos de configuración de los que depende. Para ello, el primer paso será crear una cuenta sin privilegios y un grupo con el que será lanzado el programa.

```
tfgsdn@ubuntu sudo groupadd snort
tfgsdn@ubuntu sudo useradd snort -r -s /sbin/nologin -c SNORT_IDS -
g snort
```

Además, se crearán las carpetas y ficheros para satisfacer las dependencias de Snort, todo ello con los permisos necesarios para que funcione correctamente:

```
tfgsdn@ubuntu sudo mkdir /etc/snort
tfgsdn@ubuntu sudo mkdir /etc/snort/rules
tfgsdn@ubuntu sudo mkdir /etc/snort/rules/iplists
tfgsdn@ubuntu sudo mkdir /etc/snort/preproc_rules
tfgsdn@ubuntu sudo mkdir /usr/local/lib/snort_dynamicrules
tfgsdn@ubuntu sudo mkdir /etc/snort/so_rules

# Crear archivos que contendrán reglas y listas de IPs :
tfgsdn@ubuntu sudo touch /etc/snort/rules/iplists/black_list.rules
tfgsdn@ubuntu sudo touch /etc/snort/rules/iplists/white_list.rules
tfgsdn@ubuntu sudo touch /etc/snort/rules/local.rules
tfgsdn@ubuntu sudo touch /etc/snort/sid-msg.map

# Crear los directorios donde se almacenarán nuestros registros:
tfgsdn@ubuntu sudo mkdir /var/log/snort
tfgsdn@ubuntu sudo mkdir /var/log/snort/archived_logs

# Modificar los permisos de las carpetas:
tfgsdn@ubuntu sudo chmod -R 5775 /etc/snort
tfgsdn@ubuntu sudo chmod -R 5775 /var/log/snort
tfgsdn@ubuntu sudo chmod -R 5775 /var/log/snort/archived_logs
tfgsdn@ubuntu sudo chmod -R 5775 /etc/snort/so_rules
tfgsdn@ubuntu sudo chmod -R 5775 /usr/local/lib/snort_dynamicrules

#Cambiamos la propiedad de las carpetas para que no haya problemas
de acceso
tfgsdn@ubuntu sudo chown -R snort:snort /etc/snort
tfgsdn@ubuntu sudo chown -R snort:snort /var/log/snort
tfgsdn@ubuntu sudo chown -R snort:snort
/usr/local/lib/snort_dynamicrules
```

Por último, copiaremos los archivos de configuración y sus preprocesadores dinámicos:

```
tfgsdn@ubuntu cd snort-2.9.16/etc/
tfgsdn@ubuntu sudo cp *.conf* /etc/snort
tfgsdn@ubuntu sudo cp *.map /etc/snort
tfgsdn@ubuntu sudo cp *.dtd /etc/snort

tfgsdn@ubuntu cd snort-2.9.16/src/dynamic-
preprocessors/build/usr/local/lib/snort_dynamicpreprocessor/
tfgsdn@ubuntu sudo cp * /usr/local/lib/snort_dynamicpreprocessor/
```

Obtenemos, tras estos pasos, la siguiente distribución de carpetas:

- Archivo binario Snort: /usr/local/bin/snort
- Archivo de configuración Snort: /etc/snort/snort.conf
- Directorio de registros Snort: /var/log/snort
- Directorios de reglas Snort: /etc/snort/rules
  /etc/snort/so rules
  /etc/snort/preproc rules
  /usr/local/lib/snort dynamicrules
- Directorio de listas IP Snort: /etc/snort/rules/iplists
- Preprocesadores dinámicos Snort: /usr/local/lib/snort dynamicpreprocessor/

Con la estructura de archivos y carpetas creada, se debe modificar el archivo de configuración de Snort, *"etc/snort/snort.conf"*. Lo primero que se ha realizado, ha sido comentar todos los archivos de reglas que incluye Snort por defecto ya que el interés reside en la creación de reglas propias hechas a medida del sistema. Se ha utilizado el siguiente comando para este cometido:

```
$sudo sed -i "s/include \$RULE\_PATH/#include \$RULE\_PATH/"
                    /etc/snort/snort.conf
```

Hecho esto, lo siguiente será editar aquellas líneas del fichero de configuración en relación con los detalles de la red utilizada y las relacionadas con las carpetas a utilizar por la aplicación donde se indicarán aquellas creadas previamente. También, se habilitará el uso de reglas locales.

```
44  # Setup the network addresses you are protecting
45  ipvar HOME_NET any

101 # Path to your rules files (this can be a relative path)
102 # Note for Windows users:  You are advised to make this an absolute path,
103 # such as:  c:\snort\rules
104 var RULE_PATH /etc/snort/rules
105 var SO_RULE_PATH /etc/snort/so_rules
106 var PREPROC_RULE_PATH /etc/snort/preproc_rules
107
108 # If you are using reputation preprocessor set these
109 # Currently there is a bug with relative paths, they are relative to where snort is
110 # not relative to snort.conf like the above variables
111 # This is completely inconsistent with how other vars work, BUG 89986
112 # Set the absolute path appropriately
113 var WHITE_LIST_PATH /etc/snort/rules/iplists
114 var BLACK_LIST_PATH /etc/snort/rules/iplists

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

Una vez realizados los cambios en el archivo de configuración, estos deben ser comprobados puesto que, si existe algún error, la configuración no será cargada y Snort no podrá funcionar correctamente. El comando para comprobar si la configuración es correcta es:

```
$sudo snort -T -c /etc/snort/snort.conf -i ens33
```

Donde ens33 se trata de la interfaz de red de la máquina en la que se ejecuta Snort. Si no existe ningún problema de configuración, la respuesta a este comando será la siguiente:

```
Snort successfully validated the configuration!
Snort exiting
tfgsdn@ubuntu:~$
```

4. CONFIGURACIÓN BARNYARD2 Y MYSQL

Para la instalación de Barnyard2 y MySQL, lo primero es instalar las dependencias de Barnyard2:

```
tfgsdn@ubuntu sudo apt-get install -y mysql-server libmysqlclient-
dev mysql-client autoconf libtool
```

Tras esto, se nos pide una contraseña para MySQL. Y una vez introducida pasaremos a descargar e instalar Barnyard2:

```
tfgsdn@ubuntu cd usr/src/snort_src
tfgsdn@ubuntu wget
https://github.com/firnsy/barnyard2/archive/master.tar.gz -O
barnyard2-Master.tar.gz
tfgsdn@ubuntu tar zxvf barnyard2-Master.tar.gz
tfgsdn@ubuntu cd barnyard2-master
tfgsdn@ubuntu autoreconf -fvi -I ./m4

#Creamos un enlace simbólico hacia la libería dnet.h
tfgsdn@ubuntu sudo ln -s /usr/include/dumbnet.h /usr/include/dnet.h
tfgsdn@ubuntu sudo ldconfig

#Terminamos la configuración con MySQL
tfgsdn@ubuntu ./configure --with-mysql --with-mysql-
libraries=/usr/lib/x86_64-linux-gnu
```

Por último, resta terminar la instalación de Barnyard2 y crear los archivos y carpetas necesarios para su correcto funcionamiento:

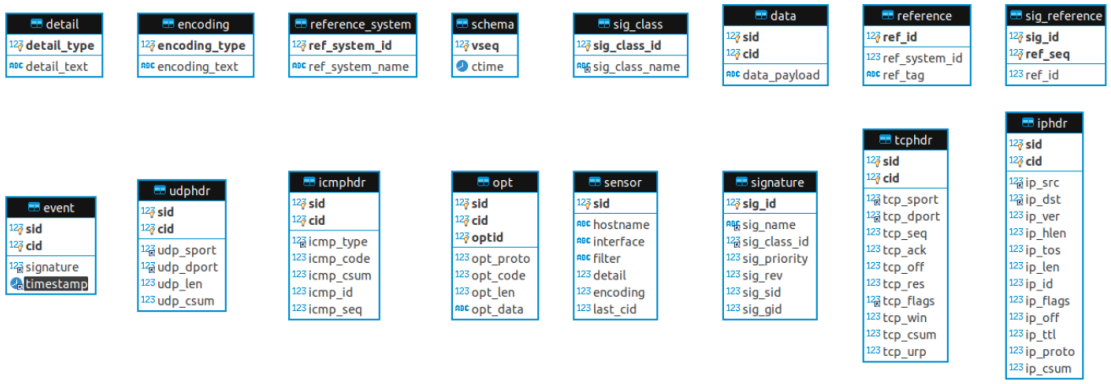
```
tfgsdn@ubuntu cd barnyard2-master
tfgsdn@ubuntu make
tfgsdn@ubuntu sudo make install
tfgsdn@ubuntu sudo cp /usr/src/snort_src/barnyard2-
master/etc/barnyard2.conf /etc/snort/
tfgsdn@ubuntu sudo mkdir /var/log/barnyard2
tfgsdn@ubuntu sudo chown snort.snort /var/log/barnyard2
tfgsdn@ubuntu sudo touch /var/log/snort/barnyard2.waldo
tfgsdn@ubuntu sudo chown snort.snort /var/log/snort/barnyard2.waldo

#Para prevenir que otros usuarios puedan leer la contraseña:
tfgsdn@ubuntu sudo chmod o-r /etc/snort/barnyard2.conf
```

Es necesaria una modificación del fichero de configuración de Snort para que genere los archivos de registro en formato unified2 el cual es el procesado por la herramienta Barnyard2.

```
519 # unified2
520 # Recommended for most installs
521 output unified2: filename /var/log/snort/snortu2, limit 128
```

Una vez se ha configurado tanto Snort como Barnyard2, tendremos que crear la base de datos para las alertas. Se iniciará sesión en MySQL desde un usuario con privilegios (root), una vez iniciada, se realizarán los siguientes pasos:

```
# Se crea la base de datos llamada snort
create database snort;

# Se selecciona la base de datos
use snort;

# Importamos el esquema de la base de datos proporcionado por la herramienta barnyard

source /usr/src/snort_src/barnyard2-master/schemas/create_mysql

#Se crea el usuario snort con contraseña root

create user 'snort'@'localhost' identified by 'root';

#Se le otorgan todos los permisos a este usuario

grant create, insert, select, delete, update on snort.* to 'snort'@'localhost';
```



Realizados estos pasos, se obtiene una base de datos que sigue el esquema de la imagen anterior.

Por su parte, el comando para poner en funcionamiento Barnyard2 es el siguiente:

```
$sudo barnyard2 -c /etc/snort/barnyard2.conf -d /var/log/snort/  -w
   /var/log/snort/barnyard2.waldo -g snort -u snort -f snortu2;
```

5. FUNCIONES DE LOS ESPACIOS DE NOMBRES DE RED

```
as_ns () {
  NAME=$1                          #nombre del espacio de red
  NETNS=faucet-${NAME}
  Shift                            #desprecia el primer argumento
  sudo ip netns exec ${NETNS} $@   #ejecuta como NETNS el comando indicado en\
                                    el segundo argumento.
}
```

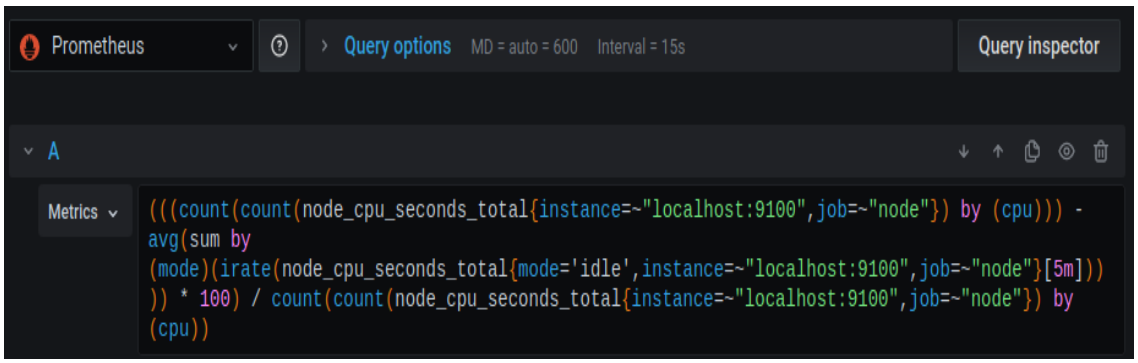
```
create_ns () {
  NAME=$1     #se toma el primer argumento para identificar el espacio de red
  IP=$2                            #la dirección será el segundo argumento
  NETNS=faucet-${NAME}             #nombre del espacio de red
  sudo ip netns add ${NETNS}       #crea un espacio de red con el nombre NETNS
  sudo ip link add dev veth-${NAME} \
  type veth peer name veth0 netns ${NETNS}   #se añade una interfaz ethernet
                                              virtual llamada veth-Nombre y
                                              se asocia al extremo virtual
                                              llamado veth0 que forma parte
                                              del espacio de red NETNS
  sudo ip link set dev veth-${NAME} up       #se habilita esta interfaz
  as_ns ${NAME} ip link set dev lo up        #dentro del espacio de red, se
                                              habilita la interfaz loopback
  [ -n "${IP}" ] && as_ns ${NAME}\
  ip addr add dev veth0 ${IP}              #se añade la ip determinada a la
                                            interfaz veth0
  as_ns ${NAME} ip link set dev veth0 up   #habilita la interfaz veth0
}
```

6. CONFIGURACIÓN GRAFANA Y CREACIÓN TABLERO

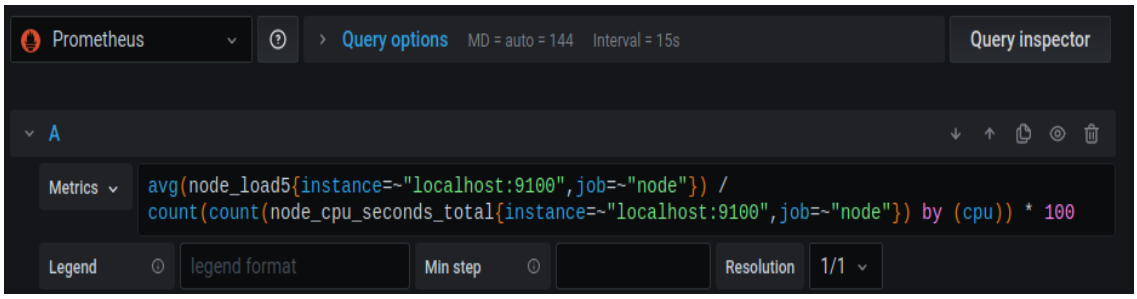
Para la visualización de las estadísticas de una forma estructurada y agrupando las que son de mayor interés se ha creado un tablero en Grafana que muestra, como se puede ver en la Ilustración 29, el detalle de tanto el estado del sistema (carga cpu, ram utilizada, etc) como el estado de la red, así como una vista de la base de datos de Snort.

Por extensión y ya que no resulta interesante en sí mismo todo el archivo JSON correspondiente al tablero, se muestran las Querys, o consultas para mostrar los datos como se ven en la ilustración.

- Para la carga de la CPU:

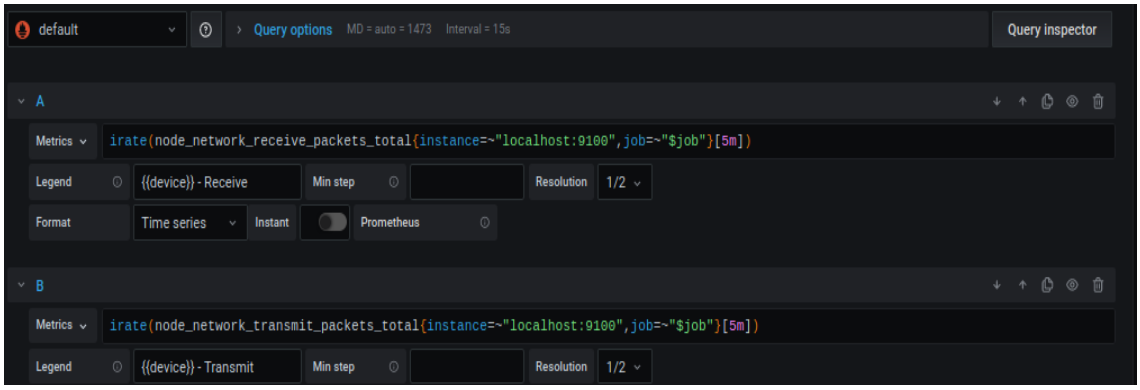


- La carga del sistema



- RAM usada

- Estado de la red



- Base de Datos