Universidad Complutense de Madrid

Facultad de Informática

SIMULACIÓN DEL DESPLIEGUE DE IPV6 EN LA RED DE LA UCM. RETOS Y SEGURIDAD

SIMULATION OF IPV6 DEPLOYMENT ON THE UCM NETWORK. CHALLENGES AND SECURITY

Trabajo Fin de Máster
Máster en Informática

Autor
Gonzalo Isla Llave

Tutor
Juan Carlos Fabero

Convocatoria: Enero 2026

Calificación: 10/10

Agradecimientos

Quiero expresar mi más profundo agradecimiento a mis profesores de la universidad por todos estos años de enseñanza, especialmente al tutor de este trabajo, Juan Carlos Fabero. Gracias por el apoyo brindado desde el primer día, por la confianza aportada y por la orientación ofrecida durante estos meses, incluso en época de festividad.

Gracias a mis padres, por su amor y paciencia durante todos estos años de estudios, y a mis hermanas, Mónica y Cristina, por ser un pilar fundamental en mi vida.

Gracias al resto de mi familia, por su comprensión, motivación y cariño constante.

Finalmente, quiero agradecer a todos mis amigos por darme el respiro necesario para seguir adelante.

Resumen

La adopción de IPv6 se ha convertido en un elemento clave para la modernización de las infraestructuras de red. Esta importancia viene de un progresivo agotamiento de direcciones IPv4 junto al incremento exponencial de número de dispositivos. Para su implementación de forma segura se necesitan mecanismos de seguridad y detección de ataques.

Con esta misma premisa lo que se persigue conseguir en este trabajo es realizar una prueba de concepto en la cual se simula el despliegue de IPv6 en la infraestructura de red de la UCM a partir de la herramienta GNS3. A través de la asignación de bloques /64 a las distintas facultades y con ayuda de protocolos de encaminamiento como OSPF[1] y BGP[2], se ha conseguido realizar la conexión entre los distintos puntos de la red. Además, se ha realizado un sistema de monitorización de la red que controla de forma pasiva el tráfico mediante el uso de IDS[3], concretamente Suricata, y se ha puesto a prueba mediante ataques IPv6 con la herramienta thc-ipv6.

El resultado obtenido del trabajo es el correcto funcionamiento del sistema de monitorización creado para la mitigación de ataques IPv6. Para lograrlo, se ha demostrado la eficiencia del IDS en la mayoría de los ataques probados, mostrando en el dashboard creado las alertas generadas en tiempo real. Además, se han visualizado los ataques realizados mediante *Wireshark* para definir donde existen vulnerabilidades en el protocolo IPv6. Por último, mediante el sistema de monitorización, se ha habilitado una vía a través de un formulario web para añadir reglas *nftables*, lo cual permite realizar una mitigación de forma rápida y eficaz.

Palabras clave: GNS3, IPv6, monitorización, seguridad, dashboard, *nftables*, IDS, Suricata y despliegue. El repositorio con el sistema de monitorización, scripts y la configuración realizada se encuentra en este enlace.

[1] Open Shortest Path First
[2] Border Gateway Protocol
[3] *Intrusion Detection System* (Sistema de detección de intrusos)

Abstract

The adoption of IPv6 has become a key element for the modernisation of network infrastructures. This importance stems from the progressive exhaustion of IPv4 addresses coupled with the exponential increase in the number of devices. For its secure implementation, security and attack detection mechanisms are required.

Under this premise, the objective of this work is to carry out a proof of concept simulating the deployment of IPv6 within the UCM (Complutense University of Madrid) network infrastructure using the GNS3 tool. By assigning /64 blocks to the different faculties and utilising routing protocols such as OSPF and BGP, connectivity between the various network points has been achieved. Additionally, a network monitoring system has been implemented to passively control traffic using an IDS—specifically Suricata—and has been tested against IPv6 attacks using the thc-ipv6 tool.

The outcome of this work is the correct operation of the monitoring system designed for IPv6 attack mitigation. The efficiency of the IDS has been demonstrated for most of the attacks tested, with the custom dashboard displaying the alerts generated in real time. In addition, the attacks carried out have been inspected using *Wireshark* in order to identify where vulnerabilities exist in the IPv6 protocol. Finally, through the monitoring system, a web form has been provided that allows *nftables* rules to be added, enabling rapid and effective mitigation.

Keywords: GNS3, IPv6, monitoring, security, dashboard, nftables, IDS, Suricata, and deployment. The repository containing the monitoring system, scripts, and the implemented configuration is available at this link

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Motivación

El protocolo de Internet (IP) constituye uno de los pilares fundamentales en la arquitectura de Internet. Desde sus orígenes en ARPANET, su espacio de direccionamiento de 32 bits nunca fue un problema durante el inicio debido al número reducido de dispositivos conectados. El aumento exponencial de dispositivos conectados a Internet, impulsado por la popularización de la *World Wide Web*, ha llevado al agotamiento de las direcciones IPv4. Esto pone en manifiesto la necesidad de un nuevo protocolo que garantice la continuidad y escalabilidad de las redes. Para paliar esta escasez de forma temporal, se han desarrollado soluciones intermedias, como la eliminación de la jerarquía de clases creada desde el principio con *Classless Inter-Domain Routing* (CIDR), lo que permitió una asignación más flexible de direcciones, o la traducción de direcciones de red (NAT). Ambos mecanismos han conseguido prolongar el uso de IPv4 a costa de incrementar la complejidad y limitar ciertas funcionalidades, como la conectividad extremo a extremo al reutilizar direcciones privadas.

No fue hasta mediados de los 90 que la *Internet Engineering Task Force* (IETF) inició el desarrollo de un nuevo protocolo conocido como IPng (*Internet Protocol next generation*). Fruto de esto, Steve Deering y Robert Hinden [1] publicaron, a través de un RFC[1], las especificaciones de un nuevo protocolo que elimina el problema del agotamiento de direcciones, llamado IPv6. Además de solucionar este problema, pasando de un espacio de direcciones de 32 bits en IPv4 a 128, IPv6 introduce ciertas mejoras como la autoconfiguración sin estado (SLAAC), protocolos como *Neighbor Discovery Protocol* (NDP), un encabezado más simple o una integración con el mecanismo de seguridad IPsec, entre otras.

A pesar de que el estándar de IPv6 se realizó hace casi tres décadas, su despliegue no ha sido inmediato. Realizar la migración implica grandes costes de actualización de hardware y la formación necesaria para realizarla, así como parar el tráfico de Internet para los cambios necesarios. Esto ha obligado a crear mecanismos de coexistencia que permitan realizar la transición de forma paulatina, como *dual-stack*, la creación de túneles o la traducción de direcciones. A su vez, estos mecanismos introducen nuevos retos en cuanto a la gestión, el rendimiento y la seguridad.

La complejidad del proceso de transición y el uso de IPv6 es un tema vigente en los estudios actuales. Según las estadísticas aportadas por Google [2], el 45 % del tráfico ya funciona a través de IPv6. Este estudio coloca a países como Francia o India pioneros en el uso de IPv6, siendo España uno de los países de Europa más atrasados en esta transición, con apenas un 10 % del tráfico en IPv6. Por otro lado, estudios como *'Towards a Non-Binary View of IPv6 Adoption'* [3] no se centran solo en la cantidad de uso de IPv6, sino en evaluar para qué tipo de tráfico se está utilizando. Entre sus conclusiones, se determina que muchos servicios de acceso frecuente son exclusivos de IPv4 y

[1] *Request for Comment*

que la transmisión de recursos embebidos, como imágenes, es poco frecuente. Además, se subraya la importancia de que no se trata de una cuestión de adoptar IPv6 o no, sino que es necesaria la implementación completa en recursos de terceros, empresas y proveedores de nube para garantizar un uso completo del protocolo.

Sabiendo el uso que se está realizando, en este trabajo se quiere destacar la necesidad de contar con un sistema de detección, mitigación y soporte para el conjunto de comunicaciones IPv4 e IPv6. Entornos como GNS3 permiten reproducir redes que soportan IPv4/IPv6 y evaluar los efectos a la hora de realizar el encaminamiento, QoS, el efecto sobre los firewalls y la detección de amenazas, realizando estas investigaciones en entornos controlados.

La motivación de este trabajo proviene de mi interés por el ámbito de las redes de computadores y por profundizar en el aprendizaje sobre el funcionamiento de los distintos protocolos utilizados. Más allá de la teoría, considero fundamental poder utilizar entornos de prueba para estudiar los problemas que pueda ocasionar la implantación de IPv6, así como los mecanismos necesarios para su coexistencia con IPv4.

Asimismo, el uso que se prevé que tenga IPv6, junto con la transición entre protocolos, lo convierte en una vía muy interesante para realizar pruebas, ya que permite evaluar de manera realista el impacto que puede tener en una infraestructura de red y profundizar sobre su gestión.

1.2. Objetivos

Este trabajo tiene como objetivo principal la creación de un sistema de evitación, detección y mitigación de los ataques IPv6 que puede sufrir una infraestructura de grandes dimensiones, como puede ser la red de la UCM. También se quiere proporcionar un entorno de desarrollo donde simular el despliegue para la formación del personal y mejorar la planificación del despliegue. Estos objetivos se conseguirán mediante:

- El diseño de un plan de despliegue que permita la escalabilidad de la red de la UCM.
- La simulación en GNS3 de una topología representativa de la red, identificando y evaluando el impacto de posibles ataques específicos contra IPv6.
- La creación de un dashboard que permita recopilar la detección de las alertas generadas y una vía para realizar la mitigación de los ataques.
- La puesta a prueba tanto de la infraestructura como del sistema de mitigación creado mediante ataques IPv6.

El presente documento se estructura como se describe a continuación:

- En el capítulo 2 se detallan las herramientas utilizadas en el trabajo, así como los protocolos utilizados y los mecanismos de coexistencia entre IPv4 e IPv6.
- En el capítulo 3 se describe el entorno actual de la UCM y el entorno creado para este proyecto en GNS3. Se realiza la división en subredes para cada facultad y, posteriormente, se especifica la configuración utilizada.
- En el capítulo 4 se explica la arquitectura de la propuesta, explicando cómo se realizan las comunicaciones entre las distintas herramientas.
- En el capítulo 5 se pone a prueba el sistema de monitorización realizando una serie de ataques IPv6 sobre distintos puntos de la infraestructura.
- En el capítulo 6 se realiza una comparación del sistema creado con el firewall de Fortinet.
- Por último, en el capítulo 7 se explican las conclusiones generales y las posibles vías para trabajos futuros.

Capítulo 2

Introduction

2.1. Motivation

The Internet Protocol (IP) is one of the fundamental pillars of the Internet architecture. Since its origins in ARPANET, its 32-bit addressing space was never an issue at the start due to the small number of connected devices. The exponential growth of Internet-connected devices, driven by the popularization of the World Wide Web, has led to the exhaustion of IPv4 addresses. This highlights the need for a new protocol that ensures the continuity and scalability of networks.

To temporarily alleviate this scarcity, intermediate solutions were developed, such as eliminating the class-based hierarchy originally established with Classless Inter-Domain Routing (CIDR), which allowed a more flexible address allocation, or through Network Address Translation (NAT). Both mechanisms have succeeded in extending the life of IPv4 at the expense of increased complexity and the limitation of certain functionalities, such as end-to-end connectivity through the reuse of private addresses.

It was not until the mid-1990s that the Internet Engineering Task Force (IETF) began developing a new protocol known as IPng (Internet Protocol next generation). As a result of this effort, Steve Deering and Robert Hinden [1] published, through an RFC[1], the specifications of a new protocol designed to eliminate the address exhaustion problem, called IPv6. In addition to solving this issue—expanding from a 32-bit address space in IPv4 to 128 bits—IPv6 introduces several improvements such as Stateless Address Autoconfiguration (SLAAC), the Neighbor Discovery Protocol (NDP), a simplified header, and integration with the IPsec security mechanism, among others.

Despite IPv6 being standardized nearly three decades ago, its deployment has not been immediate. Migration involves significant hardware upgrade costs, required training, and planned network downtime to perform the necessary changes. This has led to the creation of coexistence mechanisms that enable a gradual transition, such as dual-stack configurations, tunneling, or address translation. However, these mechanisms also introduce new challenges in management, performance, and security.

The complexity of the transition process and IPv6 usage remains an active area of study. According to statistics provided by Google [2], 45 % of current traffic already runs over IPv6. This study identifies countries such as France and India as pioneers in IPv6 adoption, while Spain remains one of the most delayed European countries in this transition, with barely 10 % of its traffic over IPv6. On the other hand, studies like 'Towards a Non-Binary View of IPv6 Adoption' [3] focus not only on how much IPv6 is used but also on what types of traffic rely on it. Their findings show that many frequently accessed services remain IPv4-exclusive and that the transmission of embedded resources, such as images, is still rare. The study stresses that the issue is not simply whether IPv6 is adopted or not—it is essential that third-party resources, companies, and cloud providers fully

[1] Request for Comment

implement IPv6 to ensure complete functionality of the protocol.

With this context in mind, this work aims to emphasize the need for a detection, mitigation, and support system for both IPv4 and IPv6 communications. Environments such as GNS3 allow the reproduction of networks supporting dual IPv4/IPv6 operation and enable evaluation of routing behavior, QoS implications, firewall interactions, and threat detection within controlled settings.

The motivation behind this project stems from my interest in computer networking and my desire to deepen my understanding of the various protocols in use. Beyond theoretical knowledge, I consider it crucial to use test environments to study the issues that IPv6 deployment may cause, as well as the mechanisms required for coexistence with IPv4. Moreover, the projected increase in IPv6 adoption and the ongoing protocol transition make this area highly suitable for experimentation, as it allows a realistic assessment of its impact on network infrastructure and management.

2.2. Objectives

The main objective of this project is to develop a prevention, detection, and mitigation system for IPv6 attacks that could affect a large-scale infrastructure, such as the UCM network. It also aims to provide a development environment for deployment simulation to train personnel and improve deployment planning. These objectives will be achieved through:

- Designing a deployment plan that enables scalability of the UCM network.
- Simulating a representative network topology in GNS3 to identify and assess the impact of potential IPv6-specific attacks.
- Creating a dashboard to collect and display generated alert detections and provide a means for attack mitigation.
- Testing both the infrastructure and the mitigation system by carrying out IPv6-based attacks.

The structure of this document is as follows:

- Chapter 2 details the tools used in this work, as well as the protocols and coexistence mechanisms between IPv4 and IPv6.
- Chapter 3 describes the current UCM network environment and the GNS3 environment created for this project. It outlines subnet division per faculty and the configuration used.
- Chapter 4 explains the architecture of the proposed system and how communication occurs between different tools.
- Chapter 5 tests the monitoring system by conducting a series of IPv6 attacks on various infrastructure points.
- Chapter 6 compares the proposed system with an established monitoring solution such as Fortinet.
- Finally, Chapter 7 presents the general conclusions and potential directions for future work.

Capítulo 3

Estado del arte

En este capítulo se darán a conocer las distintas herramientas utilizadas para la creación de la infraestutura de red, protocolos de red utilizados y los mecanismos de mitigación de ataques IPv6. Para cada herramienta se aporta la introducción necesaria para poder entender sin complicaciones el trabajo realizado. Para más información, se puede acudir a la documentación de la herramienta correspondiente.

3.1. Herramientas

En primer lugar, se realiza una descripción detallada de las herramientas utilizadas.

3.1.1. Tecnologías de virtualización

La virtualización a nivel de sistema operativo consiste en la abstracción del *kernel* que se quiere ejecutar sobre el sistema operativo anfitrión. Para esta abstracción, es necesario una capa que diferencie el *hardware* necesario entre ambos sistemas.

Docker

Docker [4] es una plataforma de contenedores[1] que permite la ejecución aislada del sistema operativo. Entre sus numerosas ventajas se incluye el reducido uso de memoria que realiza, comparado con las máquinas virtuales tradicionales, y la rapidez de ejecución. Además, Docker es altamente portable ya que se encarga de gestionar el contenedor y las aplicaciones que contiene.

QEMU

QEMU [5] (*Quick Emulator*) es un emulador y virtualizador de código abierto para máquinas virtuales que funciona en sistemas operativos como Linux, Windows y macOS. Existen dos modos de funcionamiento:

- Emulador: QEMU realiza una simulación completa de los componentes físicos a través de software. Esto permite ejecutar Sistemas Operativos (SO) de distintas arquitecturas.
- Virtualizador: QEMU ejecuta directamente en el procesador del anfitrión. Depende del modo acelerado, también llamado *driver KVM (Kernel-based Virtual Machine)*. Sólo se permite para cierto *hardware*.

[1] Unidad ligera que incluye la aplicación y sus dependencias.

GNS3

GNS3 (*Graphical Network Simulator-3*) [6] es un *software* gratuito y libre, que permite realizar simulaciones de redes complejas mediante una serie de componentes como *routers*, *switches*, IPS, IDS, máquinas con distintos sistemas operativos como Windows o Linux, accesos a NAT, etc. Esta aplicación permite correr imágenes *CiscoIOS* utilizando `Dynamips`[2] [7] de forma sencilla, pudiendo conseguir crear contenedores personalizados e importar contenedores creados por terceros. De hecho, es el propio GNS3 el que proporciona una sección en su página web donde se pueden subir distintos contenedores (*appliance*), sean versiones de CISCO, FortiGates, Windows, máquinas Kali, etc. La herramienta de *front-end* que utiliza GNS3 para la emulación de routers se llama *DynaGen* [8].

La arquitectura de GNS3 se basa en el modelo cliente-servidor, donde el cliente gráfico se comunica con un servidor local o remoto que ejecuta las máquinas virtuales o contenedores. Además, no existe limitación en cuanto a dispositivos conectados más allá de la limitación *hardware* de la CPU y de la memoria del dispositivo local. GNS3 es utilizado principalmente en entornos educativos, aunque también se utiliza en pruebas de concepto en el mundo profesional. Permite emular dispositivos reales, poner en práctica lo aprendido mediante el uso de numerosos protocolos de encaminamiento y utilizar otros conceptos de red como VLAN, VPNs, entre otros. El objetivo de esta herramienta es proporcionar un laboratorio personalizado que permita crear una red virtual sin necesidad de comprar el hardware y que, a la vez, sea lo más realista posible.

No obstante, GNS3 presenta ciertas limitaciones en cuanto al elevado consumo de recursos, especialmente de memoria, y la necesidad de contar con imágenes compatibles con el sistema donde se esté ejecutando la simulación.

3.1.2. Suricata

Los Sistemas de Detección de Intrusos (**IDS**) y Sistemas de Protección contra Intrusos (**IPS**) son programas que sirven como detectores de ataques sospechosos y maliciosos. La diferencia entre ambos es que el IDS únicamente informa sobre las amenazas mientras que el IPS también las elimina.

Suricata [9] es el motor IDS elegido para realizar la inspección de paquetes en tiempo real, identificar patrones y avisar. Suricata se ha elegido debido a que es *open-sourced* y por su arquitectura *multi-threading* que permite el procesamiento de un alto volumen de datos. Esto permite, a través de reglas, actuar con rapidez y tener una estructura escalable con el tiempo. Esto difiere de las herramientas utilizadas en entornos reales, ya que se utilizan otros sistemas de seguridad de pago como Palo Alto (actualmente utilizado en la UCM), Fortinet, CheckPoint, etc. Además, soporta IPv6, fundamental para este trabajo. La estructura de las reglas, también llamadas firmas, de Suricata es la siguiente:

- *Action*: determina la acción a realizar si el tráfico interceptado coincide con la firma. Puede ser: `alert, drop, reject, pass, rejectsrc, rejectdst` y `rejectboth`.
- *Header*: se determinan aspectos del tráfico como el protocolo (TCP, UDP, ICMP, ICMPv6, IPv4, IPv6, DNS, SSH, etc.), direcciones IP y puertos tanto de origen como de destino o la dirección del flujo.
- *Options*: se definen las condiciones y metadatos de la regla. Incluye un mensaje, estado de la conexión (`flow`), patrón a buscar (`content`), un identificador único de regla (`sid`), el número de versión de la regla (`rev`) y la clasificación del evento (`classtype`). Para poder realizar un buen patrón de búsqueda, se necesitan distintos mecanismos para moverse por el contenido de un paquete. Para lograrlo, Suricata ofrece las siguientes opciones:
  - `offset`: desde qué byte empezar a buscar la coincidencia.

[2] Emulador de IOS

- `depth`: hasta qué byte máximo buscar la coincidencia.
- `distance`: cuántos bytes después de la última coincidencia se empieza a buscar la siguiente.
- `within`: entre qué cantidad de bytes se puede buscar la coincidencia, contando desde el final de la coincidencia anterior.

Debido a la comparación que se hará en el capítulo 7, cabe mencionar que Suricata está inspirado en Snort, manteniendo la compatibilidad en el formato de reglas y subsanando ciertas limitaciones de este motor IDS. Para ejemplos y mayor información sobre Suricata, se debe acudir a la documentación de la creación de las reglas de Suricata en su página oficial [10].

Configuración

El archivo principal de configuración se ubica generalmente en `/etc/suricata` y recibe el nombre de `suricata.yaml`. En este archivo se definen distintos parámetros del funcionamiento de Suricata como los interfaces de red a escuchar, la salida de los ficheros, la carpeta donde se encuentran las reglas que se deben aplicar para el filtrado así como opciones de rendimiento, entre otras cosas. Entre estos ficheros destacan los situados en el directorio `/etc/suricata/rules`, donde se definen y gestionan la definición de las reglas. De este modo se definen distintos ficheros como `local.rules`, para reglas personalizadas, o `threshold.config`, que controla la generación de alertas. Estos aspectos están detallados en la documentación técnica de la herramienta [11].

En cuanto al almacenamiento del tráfico, se realiza el guardado en formato EVE JSON que registra de manera estructurada los distintos eventos generados. También se puede guardar cierto tráfico selectivo en archivos PCAP, lo que facilita un estudio del tráfico detallado con ayuda de otras herramientas como *Wireshark*. De nuevo, para más información, acuda a la documentación [12]. Por defecto, los ficheros generados por Suricata se almacenan en `/var/log/suricata`. Entre los archivos de salida más relevantes se encuentran:

- `eve.json`: contiene eventos de seguridad en formato JSON.
- `fast.log`: registra las alertas en un formato más simple, útil para depuración.
- `alert-debug.log`: incluye detalles adicionales de las alertas.
- `stats.log`: recoge métricas sobre rendimiento, como paquetes procesados, descartados, uso de hilos, memoria, etc.

3.1.3. FRRouting

FRR [13], evolución de *Quagga* [14] que a su vez evoluciona de Zebra [15], es una herramienta de encaminamiento dinámico que está formada por una serie de demonios[3] que definen el encaminamiento. Cada uno de los protocolos de encaminamiento como BGP 3.3.2 o OSPF 3.3.1 posee un demonio.

3.1.4. Vtysh

Vtysh [16](*Virtual Teletype Shell*) es una interfaz de línea de comandos (CLI) utilizada por *FRRouting* (FRR) que actúa como capa de abstracción proporcionando un entorno unificado para la configuración de los distintos protocolos de encaminamiento. De esta forma, se evita tener que realizar la modificación uno a uno de los demonios del sistema. La sintaxis familiar para los administradores de red y su funcionamiento en tiempo real hacen de ella una herramienta potente para la configuración de equipos en GNS3.

[3] Programas que se ejecutan en segundo plano.

3.1.5. Nftables

Nftables [17] (*Netfilter Tables*) es un proyecto de filtrado y clasificación de tráfico en Linux. Esta herramienta da la opción de sustituir a herramientas como *iptables, ip6tables, arptables* o *ebtables*, proporcionando una unificación de todas ellas. *Nftables* proporciona una sintaxis a través de expresiones regulares con la diferencia de que permite tomar varias acciones en una sola regla.

Las tablas *nftables* están compuestas de cadenas que, a su vez, contienen reglas. Para realizar un almacenamiento de ellas se utiliza el fichero `/etc/nftables.conf`. Para poder aplicar los cambios realizados en el fichero se debe reiniciar los cambios mediante la instrucción `systemctl restart nftables.service` o `nft -f /etc/nftables.conf`. Se puede conocer las instrucciones activas mediante la instrucción `nft list ruleset`, la cual tiene opciones como -a para conocer el `identificador/handler` de cada tabla y regla creada.

Cada tabla se asocia a una familia, ya sea `ip`, `ip6`, `inet`, `arp`, `bridge` o `netdev`. Esto crea distintos tipos de tablas mediante la instrucción `nft add table [familia] [nombre_tabla]`. De la misma forma, también se puede eliminar una tabla usando *delete* o vaciarla usando *flush*.

A su vez, las cadenas pueden ser de dos tipos: normales o de tipo base. Las normales permiten tener una mejor organización y se crean mediante `nft add chain [familia] [tabla] [nombre_cadena]`. En cambio, las cadenas de tipo base son puntos de entrada al sistema de filtrado del kernel. Estas tienen la estructura `nft add chain [familia] [tabla] [cadena] type filter hook input priority 0;` , donde el type puede ser filter (para familias ip, ip6 e inet), route o nat (ambas para familias `ip` e `ip6`). En cuanto al *hook*, correspondiente con una etapa del tratamiento del paquete, puede ser de distintos tipos, como se observa en la figura 3.1:

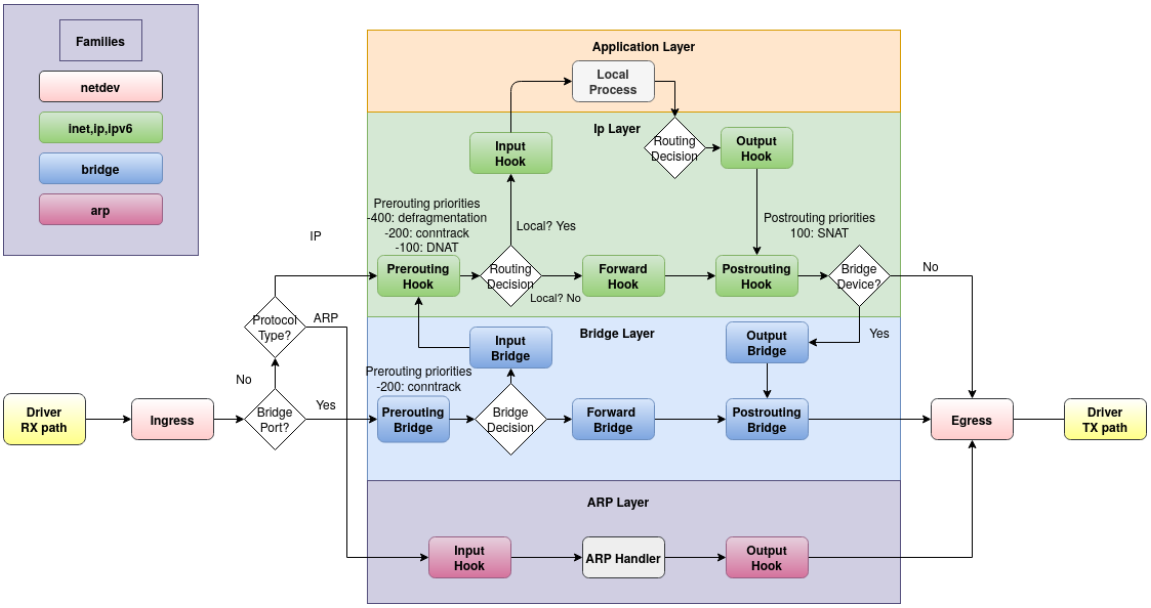


Figura 3.1: Funcionamiento de hooks en reglas *Nftables*

- `ingress`: disponible en la familia `netdev`. Realiza un filtrado muy temprano de paquetes entrantes, tomando la acción antes de encaminar el tráfico. Debido a esto, no se permite filtrar por *NAT* ni *conntrack*.
- `prerouting`: disponible en las familias `inet`, `ip`, `ipv6` y `bridge`. Se ejecuta una vez entra al sistema y su uso principal es filtrar por *DNAT*, marcado de paquetes y tomas de decisiones tempranas.

- `input`: disponible en las familias `ip`, `ipv6`, `inet`, `arp` y `bridge`. Se ejecuta en paquetes cuyo destino es el propio sistema (*Local? Yes*) una vez se ha encaminado.
- `forward`: disponible en las familias `ip`, `ipv6`, `inet` y `bridge`. Se emplea para el tráfico que no es destinado al sistema local.
- `output`: disponible en las familias `ip`, `ipv6`, `inet` y `bridge`. Se ejecuta para paquetes generados por los propios procesos del sistema.
- `postrouting`: disponible para las familias `ip`, `ipv6`, `inet`, `bridge` y `arp`. Opuesto a `prerouting`, sirve para realizar un filtrado antes de que salga por la interfaz de red, útil para *SNAT*.
- `egress`: disponible para la familia `netdev`. Sirve para realizar filtrado rápido sobre el tráfico saliente, no soporta *NAT* ni `conntrack`.

En cuanto a la `priority`, punto muy importante para el buen funcionamiento del método ya que facilita el orden en el que se ejecutan las reglas, las cadenas con menor número se procesan primero. Por último, la política de la cadena puede ser `accept`, `drop`, `queue`, `continue` o `return`.

La sintaxis para añadir una regla es la siguiente: `nft add rule [familia] [tabla] [cadena] handle [identificador] [declaración]`. El *handle* es opcional, se añadirá uno por defecto si no se indica. Para eliminar la regla se puede hacer mediante `nft delete rule [tabla] [cadena] handle [identificador]`. Debido a su posterior uso en el sistema de monitorización creado, la explicación de los campos de las reglas *nftables* es realizada en el capítulo 5.2.5.

3.1.6. *Open vSwitch*

Antes de pasar a explicar *Open vSwitch* (OVS) [18] se debe conocer *OpenFlow* [19], un protocolo abierto que se utiliza para las Redes Definidas por Software (SDN). La idea de las SDN es realizar una separación entre la capa de control y la capa de datos (capas 2 y 3) para poder gestionar el encaminamiento del tráfico a través de tablas de flujo.

Open vSwitch es un software abierto que implementa un *switch* virtual con la principal intención de poder controlar de manera dinámica el flujo de datos que atraviesa por él. Un flujo se define como un conjunto de paquetes que cumplen una determinada regla, los cuales son agrupados en tablas (de 0 a 253) y ordenados por prioridad. A su vez, cada regla está dividida en dos partes:

- `Match`: realiza la comprobación del protocolo o cierta característica de configuración.
- `Actions`: indica la acción a realizar sobre el paquete.

Para realizar la gestión del tráfico *Open vSwitch* está compuesto por dos componentes:

- `Ovsdb-server`: mantiene su BBDD y la configuración en memoria no volátil.
- `Ovs-vswitchd`: funciona como gestor, manteniendo actualizada la información respecto a la base de datos, configurando el módulo OVS y gestionando los interfaces virtuales. También admite mecanismos de configuración como *mirroring*, *trunking*, creación de VLAN y, en ciertas versiones, *patching*.

Para realizar uso de *ovs-vsctl*, el demonio encargado de configurar *Open vSwitch* es *ovs-vswitchd*. Existen numerosos comandos, entre los más comunes están:

- `ovs-vsctl add-br <bridge>`: crea un puente.
- `ovs-vsctl add-port <bridge><port>`: añade un puerto a un puente.
- `ovs-vsctl list-br`: lista los puentes existentes.

Para poder definir flujos en los *bridges* se utiliza la instrucción `ovs-ofctl`. Un ejemplo puede ser querer bloquear todos los paquetes ICMPv6 de tipo 134 (*Router Advertisement*) en el *bridge* br0 y realizarlo con alta prioridad, cuya instrucción es:

```
ovs-ofctl add-flow br0 table=0, priority=200 icmp6 icmp_type=134
    actions=drop
```

Para más información, comandos y ejemplos sobre la creación de flujos *OpenFlow*, acuda a la documentación [20].

3.1.7. Fortinet

Fortinet [21] es una empresa líder en el ámbito de la ciberseguridad encargada de proporcionar los mecanismos necesarios para proteger redes, dispositivos, aplicaciones y datos frente a amenazas.

Su producto más destacado, el *FortiGate*, es un *firewall* de nueva generación. Posee numerosas funcionalidades como el filtrado de tráfico, actúa de IPS, realiza control de aplicaciones e inspección de paquetes, entre otras funciones. Además, Fortinet contiene numerosos productos además de *FortiGate* como *FortiManager*, una herramienta que proporciona un panel que centraliza políticas, realiza cambios de configuración de forma rápida y analiza la red en tiempo real. Otro de los dispositivos que proporciona es *FortiAnalyzer*, una herramienta orientada a la generación de registros y su análisis. Fortinet también desarrolla *FortiOS*, el sistema operativo de la mayoría de los dispositivos y *Fortiguard*, la parte que incluye la investigación e inteligencia de amenazas.

Cabe mencionar que en el la *UCM* no se dispone de Fortinet sino de *Palo Alto*, pero debido a un mayor conocimiento de esta herramienta se ha decidido utilizarla para realizar la comparación en el capítulo 7.

3.2. Protocolo de red: IPv6

El protocolo IPv6 es la versión más reciente del protocolo de Intenet. Utiliza direcciones de 128 bits para identificar de forma única a los dispositivos en la red, las cuales se escriben en 8 grupos de 16 bits cada uno, separados por ":". Además, IPv6 introduce mejoras de seguridad como IPsec nativo o la autoconfiguración. Dentro de IPv6 existen tres tipos distintos de direcciones: *Unicast*, *Multicast* y *Anycast*. A su vez, existen varios tipos de direcciones *Unicast*, siendo las más utilizadas las direcciones *Unicast* Global Agregables, las direcciones *Unicast* de Enlace Local y las direcciones *Unique Local IPv6 Unicast Addresses*.

- Direcciones *Unicast* Global Agregables: sirven para interfaces conectadas a internet y permiten la autoconfiguración. Están definidas en el RFC 4291 [22].
- Direcciones *Unicast* de Enlace Local: sirven como direcciones privadas para dentro del mismo enlace local y no son encaminadas hacia el exterior. También funciona para el descubrimiento de vecinos y la autoconfiguración de interfaces, por lo que toda interfaz posee una. Las direcciones unicast de enlace local están definidas en el RFC 4291 [22] y el uso de descubrimiento de vecinos que realiza se describe en el RFC 4861 [23]. Estas direcciones utilizan el prefijo `FE80::/10`.
- Direcciones *Unique Local IPv6 Unicast Addresses* (ULA): sirven como direcciones privadas para dentro del mismo enlace local y no son encaminadas hacia el exterior. Están definidas en el RFC 4193 [24] y su prefijo de formato es `FD00::/8`.
- Direcciones *Multicast*: están definidas en el RFC 4291 [22], con asignaciones específicas descritas en el RFC 3306 [25]. Están asignadas a grupos de máquinas y un datagrama que vaya dirigido a un grupo se entrega en todas las interfaces.

- Direcciones *Anycast*: están definidas en el RFC 4291 [22]. Están asignadas a grupos de interfaces y, a diferencia de *Multicast*, un datagrama que vaya dirigido a un grupo se entrega solo en la interfaz más cercana.

Como se muestra en la figura 3.1, las cabeceras IPv6 están compuestas por varios campos de tamaño fijo que permiten el encaminamiento del tráfico.

Version (4 bits) — **Traffic Class (8 bits)** — **Flow Label (20 bits)**
Payload Length (16 bits) — **Next Header (8 bits)** — **Hop Limit (8 bits)**
Source Address (128 bits)
Destination Address (128 bits)

Tabla 3.1: Formato de la cabecera IPv6

- *Version*: indica la versión del protocolo IP: 4 o 6.
- *Traffic Class*: indica la prioridad del paquetes y la calidad de servicio (QoS - *Quality of Service*).
- *Flow Label*: permite a los routers reconocer paquetes de un mismo flujo.
- *Payload Length*: indica el tamaño del datagrama, sin contar la propia cabecera.
- *Next Header*: indica la cabecera de extensión o el protocolo de la capa superior de datos.
- *Hop Limit*: mide los saltos del datagrama, parecido aL TTL en IPv4.
- *Source and Destination Address*: dirección IP de origen y destino.

Después de la propia cabecera existen distintas opciones de cabeceras de extensión. Entre las más comunes están:

- *Hop-by-Hop*: indica que el paquete se debe procesar en cada nodo intermedio que atraviesa.
- *IPv6 Routing Header*: se trata de una lista de nodos que deben ser atravesados por el datagrama. Este encabezado se encuentra obsoleto debido a problemas de seguridad.
- *IPv6 Fragment Header*: debido a que la fragmentación siempre se realiza en el origen, esta cabecera tiene la información necesaria de los desplazamientos e identificadores para obtener el datagrama original.
- *Encapsulating Security Payload (ESP)*: proporciona confidencialidad al datagrama.
- *IPv6 Authentication Header (AH)*: proporciona integridad y autenticación al datagrama.
- *Destination Options Header*: indica que el paquete debe ser examinado únicamente por el nodo final.
- *No Next Header*: indica que no hay más encabezados después del actual.

3.3. Protocolos de encaminamiento

Antes de pasar a los protocolos de encaminamiento, es necesario entender cierto lenguaje: un sistema autónomo, como bien se describe en el RFC 1930 [26], es un conjunto conexo de redes *IP* y encaminadores bajo el control de una o varias organizaciones y con política de encaminamiento común. Utiliza un protocolo de pasarela interior, generalmente OSPF o RIP. Se comunica con otros AS mediante un protocolo de pasarela frontera (*BGP*).

3.3.1. OSPF

OSPF (*Open Shortest Path First*) [27] es el protocolo IGP (*Interior Gateway Protocol*) más utilizado. Divide el SA en áreas, todas conectadas al área 0 que recibe el nombre de área troncal. El funcionamiento de este protocolo se basa en el descubrimiento de vecinos mediante mensajes "hello", a los cuales se responde con una descripción de la base de datos. A este mensaje, a su vez, se responde con el estado de los enlaces (*LSA, Link State Advertisement*). Dentro de la red OSPF existen distintos tipos de encaminadores:

- Internos: pertenecen sólo a un área.
- Troncales: pertenecen al área troncal/área 0.
- Frontera de área (ABR – *Area Border Router*): interconectan un área con la troncal.
- Frontera de AS (ASBR – *Autonomous System Boundary Router*): intercambia información de encaminamiento entre la red OSPF y otros algorítmos.

Para la comunicación entre áreas OSPF, cada datagrama se encamina hacia la frontera de área, el cual realiza el encaminamiento del datagrama hacia el destino.

OSPFv3

Definido en el RFC 5349 [28] y actualizado en RFC 6845 [29], 6860 [30] y 7503 [31], OSPFv3 únicamente funciona a través de IPv6 y añade dos tipos nuevos de LSA:

- *Link-LSA*: proporciona un único LSA por link y realiza la inundación a nivel de enlace en vez de subred.
- *Intra-Area-Prefix LSA*: se origina en un ABR y su función es inyectar en las demás áreas un resumen de los destinos alcanzables a través de él.

Es una mejora de OSPFv2 y distribuye prefijos IPv6 sin necesidad de realizar encapsulado.

3.3.2. BGP

BGP (*Border Gateway Protocol*) [32] es el protocolo de puerta de enlace externa (EGP) que intercambia información sobre las conexiones entre Sistemas Autónomos. La información de encaminamiento incluye la ruta completa a cada destino y BGP utiliza esa información para tomar decisiones de encaminamiento basadas en políticas. Con este fin, BGP utiliza una serie de atributos que permiten influir en el proceso de decisión. Los más conocidos son los siguientes:

- *Origin*: origen del paquete, IGP o *incomplete*.
- *As_path*: lista de ASN que ha atravesado el anuncio.
- *Next_hop*: dirección IP del encaminador frontera que debe utilizarse como salto en la ruta.
- *Multi_exit_disc*: cuando existen varias conexiones entre AS, sugerencia al otro AS de cuál es preferible. Cuanto menor es el valor, más se prefiere.
- *Local_pref*: cuando existen varias conexiones, de forma local al AS, cuál es preferible para él mismo. Cuanto mayor es el valor, más se prefiere.
- *Atomic_aggregate*: cuando un encaminador recibe varios anuncios para redes solapadas a través de distintas rutas, puede optar por anunciar sólo la red que engloba a las demás.
- *Aggregator*: último ASN que formó el agregado de rutas y la IP de quién lo hizo.

Entre los *SA* se realiza un intercambio de información de encaminadores con una serie de mensajes: *Open*, que establece relación con otro encaminador frontera, *Update*, que envía la información de encaminamiento, *KeepALive*, confirma y mantiene el establecimiento de vecindad y *Notification*, que informa sobre un error y termina la conexión.

Dentro de BGP existen dos tipos de intercambios de información: entre diferentes AS (eBGP) y dentro de un sólo AS (iBGP). Si la comunicación se realiza en el mismo AS se puede pensar por qué no usar un protocolo de encaminamiento IGP. El problema viene de la pérdida de información -los atributos de BGP- que esto supondría.

3.4. Mecanismos de coexistencia entre IPv4–IPv6

En este capítulo se expondrán las dificultades de la implementación de IPv6. Existen diferentes dificultades por las cuales IPv6 no se ha llegado a implantar de forma inmediata.

En primer lugar, técnicas como el surgimiento de NAT [33] o CGNAT [34](*Carrier Grade NAT*), utilizadas para conservar direcciones IPv4, han hecho que se prorrogue la vida de este protocolo. Al no verse realmente forzadas a realizar el cambio, muchas organizaciones y empresas han retrasado la migración. Asimismo, la falta de experiencia en el personal también puede provocar un desuso de la tecnología. Otra de las causas directas que han ralentizado la adopción de IPv6 es la incompatibilidad entre IPv4 e IPv6. Para que el cambio de protocolo se pueda realizar de forma escalonada, se necesitan una serie de mecanismos de coexistencia que complican la infraestructura ya existente y aumentan los costes de mantenimiento. Para realizar esta actualización, es necesario tener un *hardware* que lo soporte (*routers*, *firewalls*, sistemas de monitorización, etc.), lo que implica una inversión económica importante para renovar distintos dispositivos de red.

A día de hoy, existen tres mecanismos para permitir la coexistencia entre IPv4 e IPv6: *dual-stack*, *tunneling* y traducción. A continuación, se procede a realizar una comparación:

3.4.1. *Dual-stack*

El mecanismo de *dual-stack* [35] posibilita la comunicación mediante ambos protocolos IPv4 e IPv6 de forma simultánea. Esta forma de coexistencia facilita una transición de forma controlada y paulatina, manteniendo los dispositivos y servicios funcionando mediante IPv4 mientras se construye la red IPv6. Además, este modo evita acciones intermedias como la traducción o el uso de encapsulamiento de paquetes.

El mecanismo de selección de protocolo puede causar un retardo en las comunicaciones, aunque a este problema se le da solución en el RFC 6555 [36] con el mecanismo "Happy Eyeballs". Este enfoque consiste en realizar una petición IPv6 y, al instante, hacerla en IPv4, en vez de realizar una petición IPv6 y esperar tiempo pudiendo ser la respuesta lenta o inválida.

Las desventajas de este mecanismo son la necesidad de una infraestructura global que soporte ambos protocolos y un mayor trabajo administrativo en cuanto a la planificación y configuración. Ambos puntos aumentan en gran medida los costes de mantenimiento. Para la configuración de IPv6 se recomienda el uso de direcciones globales *Unicast* (`2000::/3`) manteniendo las direcciones IPv4 existentes.

3.4.2. *Tunneling*

El mecanismo de *tunneling* [35] consiste en encapsular paquetes IPv6 dentro de paquetes IPv4 (o tramas MPLS), o viceversa, para atravesar redes que solo soportan un tipo de protocolo. Existen distintos tipos de métodos para establecer los túneles pero se pueden separar en dos tipos:

- Configuración manual: se indica de forma explícita la IPv4 de origen y destino. Aunque el funcionamiento es correcto, no escala de forma adecuada en entornos de grandes dimensiones.

- 6in4 [35]: a través del protocolo 41 se encapsula IPv6 directamente en IPv4. El modo 6in4 utiliza un *Tunnel Broker* [37], que consiste en un servidor con una IPv4 estática, el cual desencapsula el paquete y lo encamina hacia el destino.
- 6to4 [38]: utiliza direcciones IPv6 creadas a partir de la dirección IPv4 pública del nodo, lo que elimina la necesidad de un servidor intermedio. Este mecanismo se considera obsoleto de acuerdo con lo establecido en el RFC 7526 [39].
- 6RD(*IPv6 Rapid Deployment*) [40]: variante de 6to4, con la diferencia de que emplea prefijos IPv6 gestionados por el operador.

▪ Túneles automáticos: el túnel se crea de forma dinámica. Entre los más relevantes se encuentran:

- *Teredo [41]*: este mecanismo soluciona el problema de atravesar dispositivos NAT que no permiten el protocolo 41, encapsulando paquetes IPv6 dentro de UDP sobre IPv4. Es necesario un servidor Teredo que actúe de intermediario para extraer el paquete IPv6 y enviarlo por la red. Para ello se utilizan los *Teredo relays*, que sirven como punto de interconexión entre clientes Teredo y la red IPv6 nativa. A día de hoy, no se aconseja su uso.
- ISATAP (*Intra-Site Automatic Tunnel Addressing Protocol* [42]): similar a 6to4, permite encapsular IPv4 en IPv6. Hace que la dirección IPv6 esté formada por un prefijo (*global unicast* o *link-local*) de 64 bits, otro prefijo de 32 bits que indica que se trata de una dirección ISATAP y la propia dirección IPv4. Se debe activar el modo ISATAP en uno de los dispositivos para realizar el encapsulado y poder llegar a la otra red IPv4.
- DS-Lite (*Dual-Stack Lite*): este mecanismo, documentado en el RFC 6333 [43], solapa dos tecnologías: encapsulación de IPv4 en IPv6 y NAT. De esta forma, se consigue resolver varios problemas anteriormente vistos: la escasez de direcciones IPv4 y el alto coste de *dual-stack*.

La gran ventaja de los túneles frente a *dual-stack* es la no necesidad de tener una infraestructura completa que permita ambos protocolos, lo que abarata los costes. Por otra parte, un problema añadido es la sobrecarga de trabajo al realizar encapsular el paquete y desencapsurarlo, lo cual incrementa la carga de procesamiento.

3.4.3. Traducción de cabeceras

La traducción consiste en convertir paquetes IPv4 a IPv6 y viceversa. Para poder alcanzar redes donde sólo se utiliza IPv4, la traducción de direcciones utiliza el mecanismo NAT, concretamente NAT64. Asimismo, se debe poder utilizar DNS, DNS64 en este caso, que genere registros AAAA cuando el destino no tenga IPv6 y así se usen los registros de tipo A. Este mecanismo permite abstraerse de qué servicios están disponibles en IPv4 o IPv6 y no se requiere un soporte de la infraestructura para usarlo. En cambio, las desventajas son el uso de la comunicación extremo a extremo y la complejidad a la hora de realizar la configuración.

En principio, no haría falta *tunneling* puesto que, hasta donde sabemos, la UCM ya tiene prefijo asignado, por lo que se podría hacer nativo y aplicar *dual-stack* directamente. Debido a que IPv4 no se eliminará en un futuro próximo, no haría falta traducción de cabeceras, aunque se podría estudiar las implicaciones de su uso si fuera necesario.

Capítulo 4

Plan de despliegue e implementación

En este capítulo se explicará el plan de despliegue realizado, detallando las subredes IPv6 asignadas a cada facultad, la infraestructura para la simulación creada, la configuración de los distintos protocolos de encaminamiento para dicha red, la forma de monitorizar el tráfico y una nueva propuesta de red.

4.1. RedIRIS y REDIMadrid

Para entender el entorno de la red de la UCM primero se debe hablar de la red de comunicaciones de las universidades españolas: RedIRIS [44]. RedIRIS posee una política de *peering* abierto, tiene asignado el ASN público AS766 y, por parte de RIPE, se ha asignado el rango *Unicast* `2001:0720::/35`[1]. Aunque da soporte tanto a IPv4 como a IPv6, recomienda el uso de IPv6 para mejorar la calidad de la red. Para el intercambio de rutas utiliza BGP-4. A la hora de realizar la conexión de cada universidad con RedIRIS, la propia institución clarifica que no está permitido el peering estático hacia la red ni la modificación del campo *next hop* con uso indebido. Para realizar la implementación y modificación de la red que se planifica en este trabajo, se debe notificar a RedIRIS. Desde la página web se puede observar el mapa de la red que posee en España y Portugal, las conexiones externas tanto con Internet Global como con CDN[2] (*Content Delivery Network*), los puntos de presencia y herramientas para comprobar el estado de la red.

En el ámbito regional, REDIMadrid [45] es la infraestructura de red para las instituciones educativas y de investigación de la Comunidad de Madrid. No solo ofrece conectividad, sino que también la posibilidad de establecer VPN/VLANs, soporte nativo para IPv6, mecanismos de mitigación de ataques DDoS y QoS.

4.2. Entorno actual: características de la UCM

Gracias a la web de la UCM [46] se sabe que la red de comunicaciones de la Universidad Complutense de Madrid está compuesta por dos infraestructuras, diferenciadas según el modo de transmisión: infraestructura de red cableada e infraestructura inalámbrica. También se dispone del servicio de acceso remoto por VPN. También se dispone de la red troncal dispuesta en la figura 4.1. La red se divide en varias partes:

- El CPD/Rectorado

[1] En este trabajo se ha utilizado 2001:db8:abcd::/48 para la UCM, como ejemplo.
[2] Red de servidores distribuidos geográficamente

- 29 edificios en Moncloa
- 9 edificios en MAN (*Metropolitan Area Network*) UCM
- 5 Colegios Mayores
- 4 edificios en Somosaguas

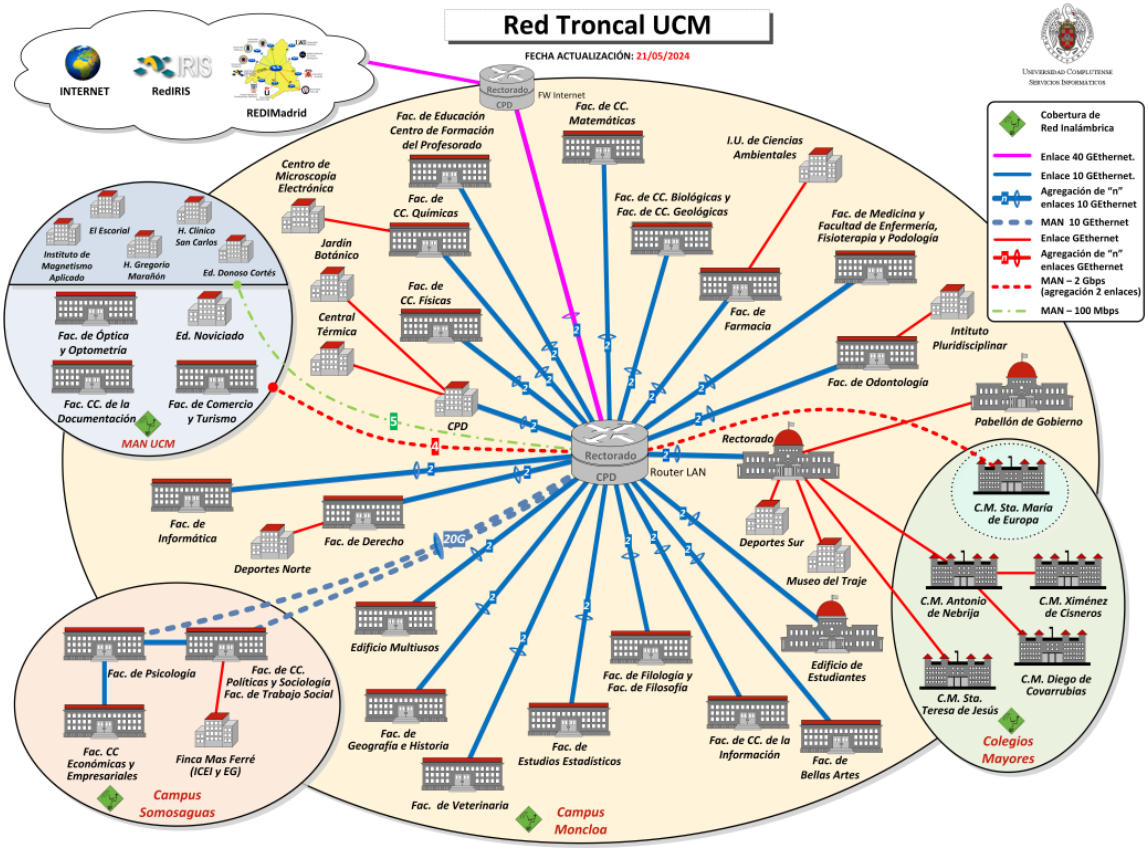


Figura 4.1: Red Troncal de la UCM a 21/05/2024. [46]

4.3. Direccionamiento

Para realizar una asignación correcta de direcciones IPv6, el despliegue se ha realizado teniendo en cuenta las recomendaciones que se realizan en el libro *IPv6 Address Planning* [47]. Debido a que se trata de una red de grandes dimensiones distribuida a lo largo de distintos puntos de Madrid, se debe realizar un despliegue que facilite un crecimiento en el número de edificios, así como la seguridad de las comunicaciones entre ellas y con el exterior. Debido a que en el futuro posiblemente se quieran realizar cambios en cuanto a la estructura de la universidad añadiendo facultades nuevas o más edificios, se debe asegurar que la infraestructura tecnológica para las comunicaciones no sufra cambios drásticos. Con este fin, y con la recomendación del libro mencionado anteriormente, se ha tomado la decisión de realizar una división dispersa o no contigua de las direcciones. Esto consiste en la asignación de redes /64 de forma distribuida, partiendo de un /48

(2001:db8:abcd::/48 [3] ha sido el tomado como ejemplo). A partir de ahí, como se observa en la figura **??**, se realiza la primera asignación en `2001:db8:abcd::/64` a la red *WiFi*. A continuación, se divide la red en dos y se asigna la siguiente red: de esta forma se asigna la siguiente red 2001:db8:abcd:8000::/64 a la facultad de Farmacia, se realiza una nueva división de la primera mitad y se asigna la `2001:db8:abcd:4000::/64` a la Facultad de Informática. De nuevo, se divide en este caso la segunda mitad y se asigna `2001:db8:abcd:2000::/64` a la Facultad de Psicología en Somosaguas.

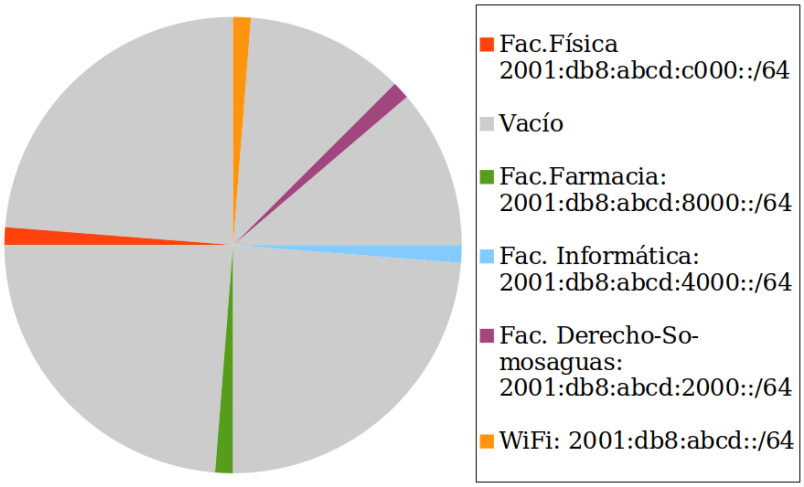


Figura 4.2: Subdivisión de la red de la UCM (2001:db8:abcd::/48)

4.4. Inventario

A continuación, se realiza una planificación de prefijos IPv6, con cada edificio en la UCM, a partir de la división *Sparse Allocation*:

Tabla 4.1: Plan de direccionamiento IPv6

Prefijo IPv6	**Asignación**
2001:db8:abcd:0000::/64	Wi-Fi
2001:db8:abcd:8000::/64	Fac. de Farmacia
2001:db8:abcd:4000::/64	Fac. Informática
2001:db8:abcd:c000::/64	Fac. de CC. Físicas
2001:db8:abcd:2000::/64	Fac. Psicología Somosaguas
2001:db8:abcd:a000::/64	CPD
2001:db8:abcd:6000::/64	IDS-Interno
2001:db8:abcd:e000::/64	IDS-Externo
2001:db8:abcd:1000::/64	IDS-Externo
2001:db8:abcd:9000::/64	Rectorado
2001:db8:abcd:5000::/64	Fac. de Medicina y Fac. de Enfermería, Fisioterapia y Podología
2001:db8:abcd:d000::/64	Fac. Químicas
2001:db8:abcd:3000::/64	Fac. de CC. de la Información
2001:db8:abcd:b000::/64	Fac. de CC. Biológicas y Fac. CC. Geológicas
2001:db8:abcd:7000::/64	Fac. de Bellas Artes

[3] El prefijo 2001:db8::/32 está reservado para el uso de ejemplos, según se describe en el RFC 3849[48]

Prefijo IPv6	Asignación
2001:db8:abcd:f000::/64	Fac. de Educación de Formación del Profesorado
2001:db8:abcd:0800::/64	Central térmica
2001:db8:abcd:8800::/64	Fac. de CC. Matemáticas
2001:db8:abcd:4800::/64	Jardín botánico
2001:db8:abcd:c800::/64	Fac. de Odontología
2001:db8:abcd:2800::/64	Centro de Microscopia Electrónica
2001:db8:abcd:a800::/64	I.U de Ciencias Ambientales
2001:db8:abcd:6800::/64	Instituto Pluridisciplinar
2001:db8:abcd:e800::/64	Fac. de Derecho
2001:db8:abcd:1800::/64	Edificio Multiusos
2001:db8:abcd:9800::/64	Fac. de Geografía e Historia
2001:db8:abcd:5800::/64	Fac. de Veterinaria
2001:db8:abcd:d800::/64	Deportes Norte
2001:db8:abcd:3800::/64	Fac. de la Filología y Fac. de Filosofía
2001:db8:abcd:b800::/64	Edificios Multiusos
2001:db8:abcd:7800::/64	Fac. de Estudios Estadísticos
2001:db8:abcd:f800::/64	Edificio de Estudiantes
2001:db8:abcd:0400::/64	Deportes Sur
2001:db8:abcd:8400::/64	Museo del Traje
2001:db8:abcd:4400::/64	Pabellón del Gobierno
2001:db8:abcd:c400::/64	C.M. Sta. Teresa de Jesús
2001:db8:abcd:2400::/64	C.M. Antonio de Nebrija
2001:db8:abcd:a400::/64	C.M. Sta.María de Europa
2001:db8:abcd:6400::/64	C.M. Diego de Covarrubias
2001:db8:abcd:e400::/64	C.M Ximénez de Cisneros
2001:db8:abcd:1400::/64	El Escorial
2001:db8:abcd:9400::/64	H. Clínico San Carlos
2001:db8:abcd:5400::/64	H. Gregorio Marañón
2001:db8:abcd:d400::/64	Ed. Domoso Cortés
2001:db8:abcd:3400::/64	Instituto de Mangnetismo Aplicado
2001:db8:abcd:b400::/64	Fac. de Óptica y Optometría
2001:db8:abcd:7400::/64	Fac. CC. de la Documentación
2001:db8:abcd:f400::/64	Fac. de Comercio y Turismo
2001:db8:abcd:0500::/64	Fac. de CC. Políticas y Sociología y Fac. de Trabajo Social
2001:db8:abcd:8500::/64	Fac. CC Económicas y Empresariales
2001:db8:abcd:4500::/64	Fianca Mas Ferré (ICEI y EG)

4.5. Simulación en GNS3

En esta sección se detallan las distintas tomas de decisiones a la hora de realizar la simulación, las limitaciones que se tienen y la forma en configurar de cada uno de los componentes del sistema.

4.5.1. Topología

A la hora de realizar la topología obviamente no se despliegan el número total de facultades existentes ya que eso causaría una red demasiado grande para poder realizar simulaciones y no aporta nada extra al trabajo. Como representación de la red, se han tomado cuatro facultades: Fac. de Física, Fac. Informática, Fac. de Farmacia y Fac. Psicología Somosaguas[4].

[4]Creadas a partir de un OVS y un SRV.

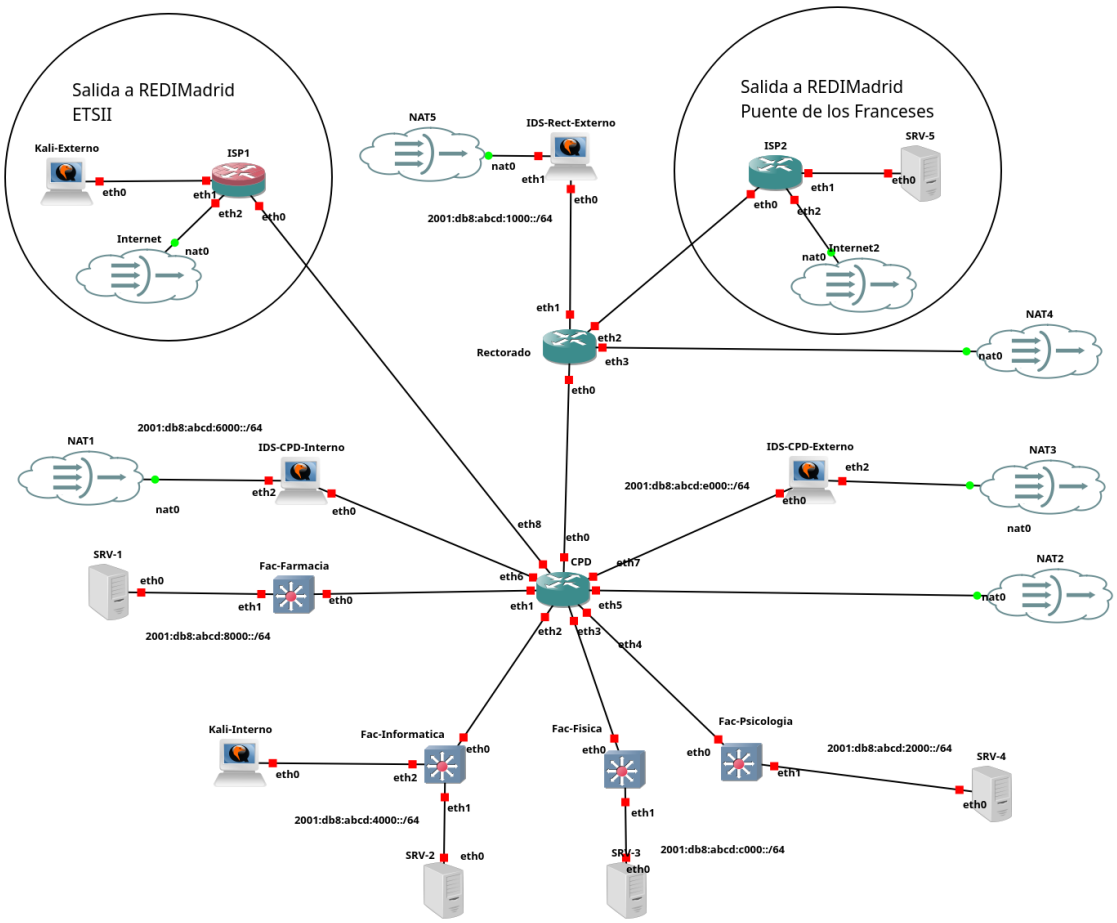


Figura 4.3: Topología simulada de la red UCM en GNS3

Como ya hemos mencionado en el capítulo 4.2, la red de la UCM tiene un *router* principal que corresponde al CPD y un *router* de *backup*, el Rectorado. De esta forma, la topología creada tiene un enlace entre CPD y Rectorado que sólo se usará en caso de que la salida del CPD con REDIMadrid a través de ETSIT (Escuela Técnica Superior de Ingenieros de Telecomunicaciones) se vea interrumpida.

Ya teniendo los principales componentes, se procede a realizar la comunicación pertinente. Para la conexión entre los ordenadores de las facultades y el CPD, cada facultad dispone de un *switch OVS* que sirve como punto de acceso a nivel de capa 2 para la conectividad con las demás facultades y con el exterior.

Para la simulación de la conexión con REDIMadrid se han añadido dos routers ISP (*Internet Service Provider*) que sirve como gateway con Internet. Al igual que en el apartado 4.2, el que está conectado al CPD corresponde a la salida por ETSIT y el que se conecta al Rectorado corresponde con el Puente de los Franceses. Conocida la estructura principal empleada para la conexión entre facultades y con el exterior, es necesario centrarse en la seguridad de la red. Con este fin, se han añadido tres IDS[5] (IDS-CPD-Externo, IDS-CPD-Interno e IDS-Rectorado-Externo) con los siguientes objetivos:

- Realizar un análisis del tráfico proveniente del exterior y el generado en el interior de las

[5]Creados con Ubuntu Desktop 22.04 [49].

facultades.

- Mandar un aviso a las personas que trabajan en el CPD cuando se detecte tráfico que pueda ser un ataque.
- Filtrar el tráfico entrante y saliente a partir de reglas *nftables*.

La decisión de añadir varios IDS es para no saturar la red. De esta forma, tanto el IDS-CPD-Interno como el IDS-CPD-Externo están en todo momento comprobando el tráfico que atraviesa el CPD. Se añaden dos IDS para repartir el trabajo, de forma que el interno supervise el tráfico saliente de la red y el externo filtre el tráfico entrante. Se tiene el tercer IDS-Rect-Externo para que, en caso de caída del enlace por ETSIT, se realice una supervisión antes de entrar al CPD.

Finalmente, se han añadido dos máquinas Kali [50], una simulando que está en el exterior al conectarla al ISP1 y otra desde una de las facultades. Estas máquinas se usarán al final del proyecto (capítulo 6) para poner a prueba la infraestructura y el sistema de monitorización creado. Como se verá más tarde en el apartado 4.5.3, debido a que los IDS no están en el camino del recorrido normal del paquete, se utiliza una técnica llamada *mirroring* para duplicar el tráfico a partir de las interfaces.

Con todo lo anteriormente descrito, se dispone de una topología sólida muy similar a la actual, que añade distintos tipos de configuraciones, como VLAN o túneles, y ofrece un amplio abanico para la realización de ataques desde distintos puntos de la red.

4.5.2. Configuración de encaminamiento

En este apartado se explicarán las distintas tomas de decisiones en cuanto a la configuración global de la simulación. Además, se verán las dificultades que han existido y cómo se han resuelto.

Cada una de las redes de las facultades contiene una red IPv6 ::/64 como se ha visto anteriormente en el apartado 4.3. A la vez, cada una de las zonas de la UCM (Moncloa, Somosaguas, MAN y Colegios Mayores) es una red OSPFv3 distinta, según la tabla 4.1.

Encaminamiento interno

Para realizar el encaminamiento interno entre facultades se utiliza OSPF, asignando una zona a cada campus. De esta forma, queda la siguiente tabla:

Zona universitaria	Zona OSPF
Campus Moncloa	Área 0
Campus Somosaguas	Área 10
MAN UCM	Área 11
Colegios Mayores	Área 12

Tabla 4.2: Tabla de zonas OSPF

Por tanto, la configuración OSPFv3 en el CPD es la siguiente:

```
interface eth0
 ipv6 ospf6 area 0.0.0.0

interface eth1
 ipv6 address 2001:db8:abcd:8000::1/64
 ipv6 nd prefix 2001:db8:abcd:8000::/64
 ipv6 ospf6 area 0.0.0.0
 ipv6 ospf6 passive
 no ipv6 nd suppress-ra
```

```
 interface eth4
 ipv6 address 2001:db8:abcd:2000::1/64
 ipv6 nd prefix 2001:db8:abcd:2000::/64
 ipv6 ospf6 area 0.0.0.10
 ipv6 ospf6 passive
 no ipv6 nd suppress-ra

 interface eth7
 ipv6 address 2001:db8:abcd:e000::1/64
 ipv6 ospf6 area 0.0.0.0


router ospf6
 ospf6 router-id 0.0.0.2
```

Para la interfaz conectada al Rectorado (eth0), únicamente es necesario la configuración del área ospf6. En cada interfaz conectada a las facultades (eth1, eth2, eth3 y eth4), lo primero de todo es asignar una dirección IPv6 según la tabla 4.1 (ejemplo: `ipv6 address 2001:db8:abcd:8000::1/64`) y anunciar su prefijo (ejemplo: `ipv6 nd prefix 2001:db8:abcd:8000::/64`). A continuación, se configura con el área correspondiente según la tabla 4.5.2. También se añade como interfaz pasiva con la instrucción `ipv6 ospf6 passive`. Esto evita que la interfaz intente formar adyacencias OSPFv3 con otros routers, debido a que no existen, y anuncie únicamente la red asignada. Para habilitar el envío de *Router Advertisements (RA)* se añade la instrucción `no ipv6 nd suppress-ra`. Además, se debe incluir un id (X) al *router* para el protocolo OSPF6: `ospf6 router-id [X]`. Para la configuración de los IDS, únicamente es necesario añadir la IP a la interfaz y el área ospf6 (interfaces eth6 y eth7).

Encaminamiento externo

Para la simulación de salida al exterior, como se comentó anteriormente, se añadieron dos vías a través del ISP1 y del Rectorado. La configuración en el CPD, conectado vía eth0 con el Rectorado y vía eth8 con el ISP1, es la siguiente:

```
route-map ISP1 permit 10
 set local-preference 200
exit
!
route-map Rectorado permit 10
 set local-preference 50

router bgp 65002
 bgp router-id 2.2.2.2
 no bgp ebgp-requires-policy
 no bgp default ipv4-unicast
 no bgp network import-check
 neighbor eth0 interface remote-as 65002
 neighbor eth2 interface remote-as 65004
 !
 address-family ipv6 unicast
  network 2001:db8:abcd::/48
  neighbor eth0 activate
  neighbor eth0 next-hop-self
  neighbor eth2 activate
```

```
  neighbor eth2 next-hop-self
 exit-address-family
```

La configuración BGP se usa para la salida hacia REDIMadrid, mientras que en la comunicación entre CPD y Rectorado no es necesario la asignación de IP públicas, ya que se puede realizar mediante la IPv6 generada por link-local. Al igual que en OSPF6, se añade un identificador al router: `bgp router-id [X]`, donde X en el identificador del router. Además, se añade la configuración para permitir anunciar rutas eBGP sin políticas explícitas a través de la instrucción `'no bgp ebgp-requires-policy'`. También se anuncian prefijos aunque no estén presentes en la tabla de encaminado local: `'no bgp ebgp-requires-policy'`. Es importante entender que estas dos opciones no deben incluirse en un sistema en producción y solo son para simplificar este trabajo, evitando añadir políticas de encaminamiento. En cuanto a la primera regla, por defecto eBGP no acepta rutas si no existe una política explícita que las acepte, ya sea a través de *route-map* o *prefix-list*. Por parte de la segunda regla, por defecto BGP solo anuncia los prefijos que existen en la tabla de rutas (RIB). Con estas opciones en producción se desactiva el funcionamiento por defecto de BGP, lo que produce que existan riesgos como anunciar rutas internas o anunciar prefijos inexistentes a Internet y, por tanto, posibilidad de caídas de Internet. Por último, se indica a qué vecino se alcanza por cada una de las interfaces a través de la instrucción `neighbor eth8 interface remote-as 65003`. La configuración `neighbor eth0 interface remote-as 65002` indica que se está conectando vía iBGP hacia el Rectorado, ya que se indica que se llega al mismo AS al que pertenece a través de otra interfaz, en este caso eth0.

Para la configuración de BGP se debe activar la interfaz `(neighbor eth0 activate)` y anunciarse como `next-hop-self` para el prefijo, necesario cuando el vecino no conoce el siguiente salto original. Tanto el CPD como el Rectorado únicamente anuncian hacia el exterior el prefijo asignado, `network 2001:db8:abcd::/48`. Como indica el RFC 4632 [51], esto se hace con el fin de limitar el crecimiento de la información de encaminamiento global. Toda la configuración anteriormente mencionada se añade de la misma forma al Rectorado:

```
router bgp 65002
 bgp router-id 2.2.2.2
 no bgp ebgp-requires-policy
 no bgp default ipv4-unicast
 no bgp network import-check
 neighbor eth0 interface remote-as 65002
 neighbor eth2 interface remote-as 65004
 !
 address-family ipv6 unicast
  network 2001:db8:abcd::/48
  neighbor eth0 activate
  neighbor eth0 next-hop-self
  neighbor eth2 activate
  neighbor eth2 next-hop-self
 exit-address-family
```

4.5.3. Redirección del tráfico a los IDS

Para poder añadir un IDS que pueda cumplir sus funcionalidades básicas de monitorizar el tráfico y poder detectar actividad peligrosa, se pensó de varias formas. En primera instancia, se utilizó un *Ubuntu Desktop Guest 22.04* con la idea de englobar el CPD y el Rectorado con el IDS. Aunque el funcionamiento de esto era correcto, al no ser del todo realista se probó a separar el IDS y el router. Finalmente, se decidió realizar el reenvío de los paquetes por *nftables* a través de *mirroring*, pero no sin antes darse cuenta de un detalle. En el caso de no especificar las interfaces de entrada del tráfico que se debía reenviar, también se estaban reenviando los paquetes de control

entre el IDS y el router, por lo que se creaba un bucle infinito. El bucle causaba una paralización del contenedor y, al cabo de varios segundos, aparecía un mensaje de error que obligaba a apagar el nodo.

```
nft add rule ip6 dup_table6 prerouting dup to 2001:db8:abcd::100
nft add rule ip6 dup_table6 prerouting ip6 daddr != 2001:db8:abcd
    ::100 dup to 2001:db8:abcd::100
```

La configuración nft se guarda en el archivo `/etc/nftables.conf`, por lo que se guarda de la siguiente forma: `sudo nft list ruleset >/etc/nftables.conf`. El reenvío de paquetes mediante la instrucción anteriormente mencionada se debe realizar a nivel de capa 2 debido a que sólo en esa capa el kernel controla los tramas Ethernet antes de que entren a la pila IP. Realizar la duplicación en capa 3 implicaría realizar un encaminado, alterando el tráfico original y, por tanto, borrando el rastro de la IP desde donde se origina el ataque. Debido a lo anteriormente descrito, la *nftables* para el CPD de forma inicial es la siguiente:

```
table inet filter {
        chain input {
                type filter hook input priority filter; policy
                   accept;
        }

        chain forward {
                type filter hook forward priority filter; policy
                   accept;
        }

        chain output {
                type filter hook output priority filter; policy
                   accept;
        }
}
table netdev dup_table6 {
        chain prerouting_eth0 {
                type filter hook ingress device "eth0" priority
                   filter; policy accept;
                ip6 daddr != 2001:db8:abcd:e000::2 dup to "eth7"
        }

        chain prerouting_eth1 {
                type filter hook ingress device "eth1" priority
                   filter; policy accept;
                ip6 daddr != 2001:db8:abcd:6000::2 dup to "eth6"
        }

        chain prerouting_eth2 {
                type filter hook ingress device "eth2" priority
                   filter; policy accept;
                ip6 daddr != 2001:db8:abcd:6000::2 dup to "eth6"
        }

        chain prerouting_eth3 {
                type filter hook ingress device "eth3" priority
                   filter; policy accept;
```

```
                ip6 daddr != 2001:db8:abcd:6000::2 dup to "eth6"
        }

        chain prerouting_eth4 {
                type filter hook ingress device "eth4" priority
                   filter; policy accept;
                ip6 daddr != 2001:db8:abcd:6000::2 dup to "eth6"
        }
}
```

Y, para el Rectorado:

```
table inet filter {
        chain input {
                type filter hook input priority filter; policy
                   accept;
        }

        chain forward {
                type filter hook forward priority filter; policy
                   accept;
        }

        chain output {
                type filter hook output priority filter; policy
                   accept;
        }
}
table netdev dup_table6 {
        chain prerouting_eth0 {
                type filter hook ingress device "eth0" priority
                   filter; policy accept;
                ip6 daddr != 2001:db8:abcd:1000::2 dup to "eth2"
        }
}
```

4.5.4. Salida por NAT

La conexión con el sistema anfitrión ha sido utilizada para la instalación de herramientas como vtysh, suricata o sshd, entre otros. Debido a que la compañía de Internet contratada no ofrece IPv6, para la conexión con el exterior se ha utilizado IPv4 mediante NAT (*Network Address Translation*). Como su propio nombre indica, NAT es una herramienta de traducción de direcciones IP entre una red privada (GNS3) y una red pública. Para hacer uso se asigna una IP que pertenezca a la red privada del sistema anfitrión, en mi caso, 192.168.122.0/24, y tenga conexión con la puerta de enlace: 192.168.122.1.

```
6: virbr0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
   state UP group default qlen 1000
    inet 192.168.122.1/24 brd 192.168.122.255 scope global virbr0
       valid_lft forever preferred_lft forever
```

Así, la configuración estática de una de las conexiones es la siguiente:

```
auto eth3
iface eth3 inet static
```

```
address 192.168.122.200
netmask 255.255.255.0
gateway 192.168.122.1
```

La otra opción era levantar un túnel con IPv6 habiendo recibido un /48 de *tunnelbroker*. Esto no se realizó debido a la toma de decisión de no realizar ataques desde el exterior para no poner en peligro tanto la seguridad de las máquinas atacantes como la mía. Por lo tanto, se simula que todo lo que llegue por *eth2* del Rectorado y *eth8* del CPD es parte de Internet.

4.5.5. Propuesta de nueva topología

Para poder añadir IPv6, mi propuesta es realizar un cambio en la topología actual. Lo que se propone es dividir la infraestructura en cuatro zonas OSPF, de forma que se tenga un router por cada facultad. De cara al exterior, se propone una conexión iBGP entre el CPD-Rectorado y salida de ambos routers vía BGP con REDIMadrid. Esta propuesta tiene distintas ventajas:

- Facilita el crecimiento futuro de la red.
- Limita la propagación de LSAs.
- Permite la coexistencia de IPv4/IPv6.
- Mejora la seguridad y facilita la aplicación de políticas.
- Aisla los problemas internos, reduciendo el impacto sobre el resto de la red.

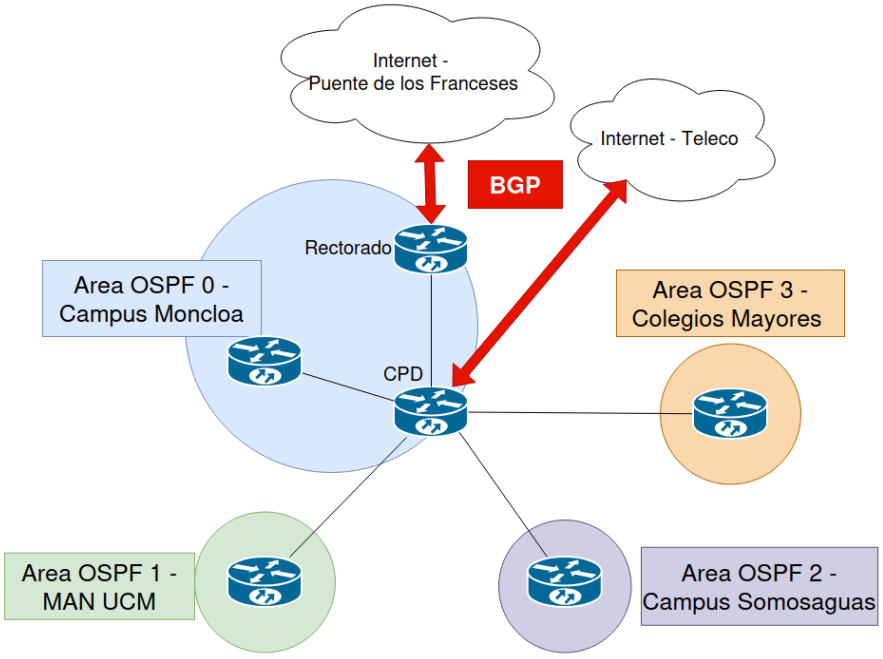


Figura 4.4: Propuesta de nueva topología para IPv6

Capítulo 5

Arquitectura de la propuesta

En este capítulo se describe en detalle la arquitectura de la propuesta planteada en el trabajo. Dicha arquitectura está dividida en dos componentes: los IDS y el dashboard.

El objetivo de la arquitectura es proporcionar un sistema de detección y mitigación de ataques, especialmente sobre la infraestructura IPv6 creada en el capítulo 4. En dicho capítulo, se integraron tres IDS a la arquitectura generada en GNS3. A continuación, se introduce un dashboard que monitorice el sistema. De esta forma, la información producida en los ataques y la configuración de cada nodo está centralizada, pudiendo tomar decisiones de bloqueos de forma eficaz.

El dashboard no se limita a una interfaz web, sino que abarca tanto el *frontend* como el *backend* del sistema. El *frontend* proporciona la interfaz gráfica a través de vía web. Por su parte, el *backend* está implementado mediante el *framework* Django, que se encarga de la lógica del sistema y de las comunicaciones con GNS3 y MongoDB; ésta última herramienta es utilizada para mantener un historial de las alertas generadas por Suricata. De este modo, la arquitectura propuesta interconecta las distintas herramientas utilizadas a lo largo del proyecto, facilitando el bloqueo ante ataques IPv6.

5.1. Funcionalidades

Las funcionalidades principales que debe tener la interfaz web, de cara a centralizar tanto la configuración como los ataques realizados, son las siguientes:

- Proporcionar una visualización de la infraestructura.
- Dar acceso a la terminal de cada nodo.
- Mostrar las alertas generadas por los IDS.
- Proporcionar una forma de bloquear el ataque.
- Monitorizar si se sigue produciendo el ataque en el CPD y en el Rectorado.

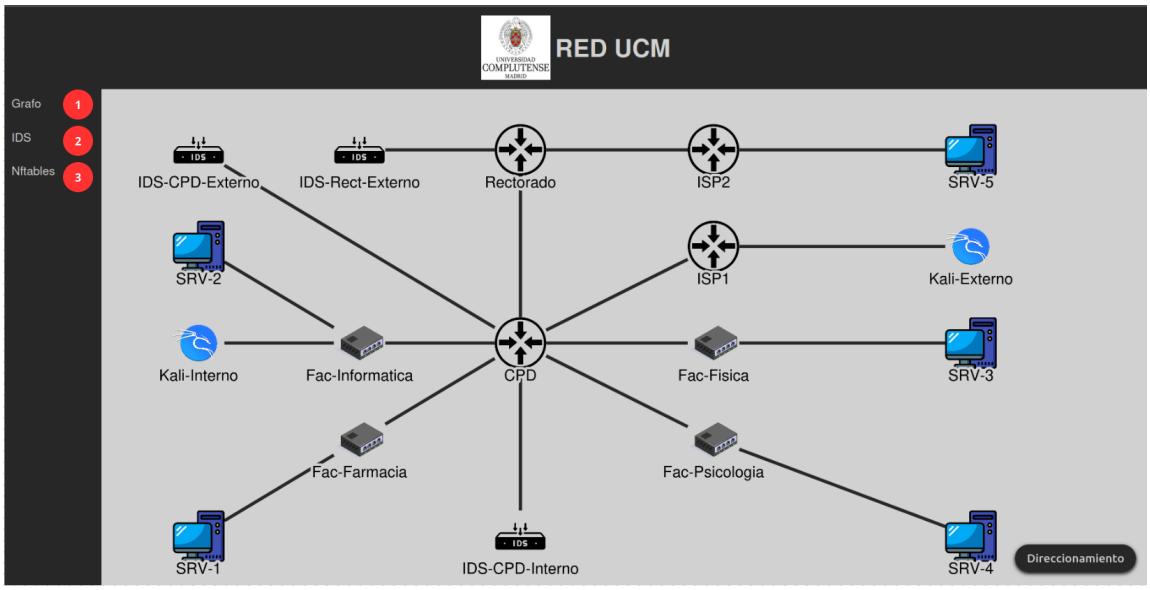


Figura 5.1: Dasboard inicial con distintas pestañas.

Para cumplir dichas funcionalidades,como se observa en la figura5.1, el dashboard está dividido en tres apartados: grafo de la red, alertas del IDS y reglas *nftables*. La pestaña del grafo tiene la funcionalidad de mostrar los distintos nodos que existen en la red central simulada en GNS3. Para tener un acceso rápido a los dispositivos, cada nodo funciona a su vez de punto de acceso al dispositivo. Al pinchar sobre el dispositivo, se abre una pestaña web con el acceso al mismo, ya sea *Docker* o *QEMU*.

La pestaña de IDS muestra una tabla de las distintas alertas que han sido generadas por los tres IDS añadidos en el capítulo 4.5.1. En esta tabla se muestran tanto quién ha generado la alerta (IDS-CPD-Externo, IDS-CPD-Interno o IDS-Rectorado) como los distintos campos de la alerta generada por Suricata: hora de la alerta, tipo de alerta, categoría, severidad, ip origen/destino, puerto origen/destino e información extra. Por cada alerta se dispone de un botón que lleva a un formulario web que permite al usuario crear una regla *nftable* ya sea en el CPD o en el Rectorado. De esta forma, se impide que se vuelva a realizar un ataque ya producido o se bloquea una IP temporalmente.

Por último, se dispone de la pestaña para mostrar las *nftables*. Debido a que es la herramienta principal utilizada para impedir el tráfico necesario, se ha realizado una pestaña que muestra las distintas *nftables*. En la parte inferior derecha de la pantalla se encuentra el desplegable 'Direccionamiento', en el cual se muestran cada uno de los dispositivos disponibles, con un botón que despliega a su lado un listado de las direcciones que tiene en cada interfaz. Esta parte está automatizada, realizando una petición de las direcciones a cada nodo una vez se abre el desplegable.

A continuación, se explicará cómo se ha conseguido cada funcionalidad con mayor detalle.

5.2. Arquitectura y tecnologías

Para realizar el dashboard se ha decidido utilizar el *framework* Django, principalmente por su facilidad de uso. Django es una herramienta de desarrollo web, además de permitir combinarlo con código Python, esencial para este proyecto. Mantiene una estructura muy sencilla gracias a la arquitectura MVC (*Model-View-Template*) que utiliza, la cual trabaja de forma modular:

- *Model*: se definen las estructuras de datos.

- *View*: contiene la lógica de la aplicación.
- *Template*: gestiona la capa de presentación, separando diseño y código HTML.

En la figura 5.2 se muestra un grafo global de las comunicaciones que abarca el proyecto para su correcto funcionamiento.

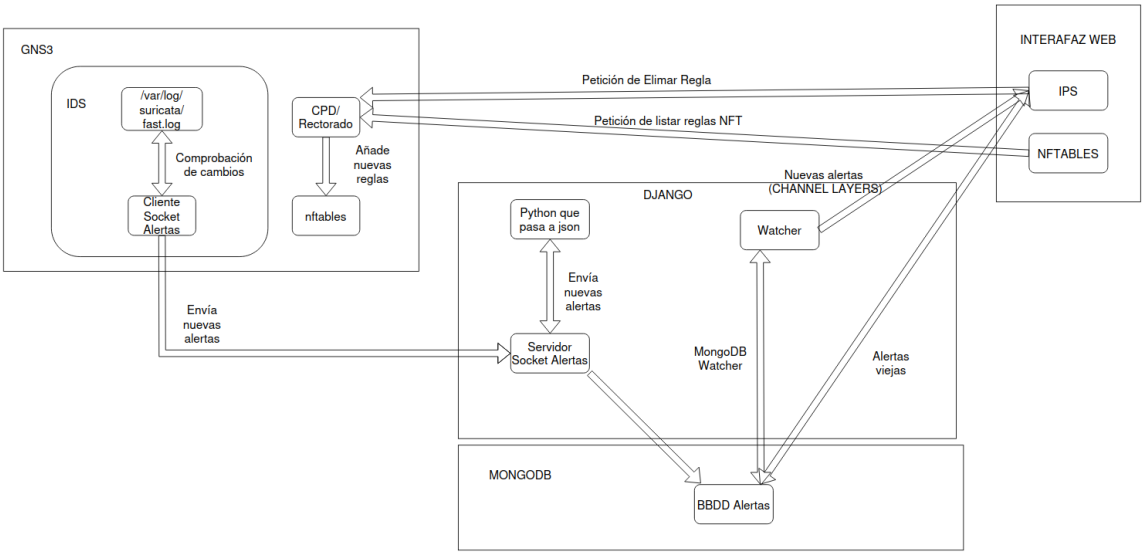


Figura 5.2: Diagrama de las comunicaciones entre GNS3 - Django - Interfaz Web.

A continuación, se realiza una explicación detallada de cada módulo del grafo, describiendo con mayor precisión las comunicaciones entre las tecnologías utilizadas, así como las decisiones adoptadas durante el desarrollo.

5.2.1. Creación del grafo

El trazado del grafo se ha realizado mediante la herramienta *Cytoscape* [52]. *Cytoscape* es un software libre que facilita el visualizado y la interacción de redes. Para la creación del grafo se realiza un análisis de los datos del fichero `.gns3` del proyecto y se construye un `.json` con la topología. Gracias a la información de los nodos y los enlaces, se consigue transformar al formato compatible con *Cytoscape.js*, pudiendo, por lo tanto, construir la red con facilidad. Lo único que falta es asignar iconos a cada nodo según su nombre y asignar una posición fija dentro del grafo.

Para obtener acceso a las máquinas se proporcionan dos mecanismos distintos. Por un lado, se ofrece abrir un terminal web (*WebTTY*) que se conecta mediante Telnet. Esto es posible en aquellos contenedores Docker. Por otra parte, se accede a ciertas máquinas virtuales QEMU a través de una interfaz gráfica, usando noVNC.

La forma que existe para crear una pasarela entre una consola y una interfaz web es mediante el uso de dos herramientas: Telnet[1] y un servidor ttyd[2]. Utilizando la información que aporta GNS3 de los puertos Telnet donde se está ejecutando cada máquina, se lanza un servidor ttyd que crea un terminal accesible a través del navegador y se conecta de forma interna al puerto Telnet asociado. El proceso se lleva a cabo a través del *endpoint* "/launch_ttyd", encargado de arrancar un proceso ttyd para el puerto Telnet solicitado.

[1] Protocolo de red que posibilita el acceso a una máquina de forma remota.
[2] Herramienta de línea de comandos que permite compartir una sesión a través de un sitio web.

Para habilitar el terminal gráfico en las máquinas QEMU desde el dashboard, se hace uso de la biblioteca noVNC [53]. VNC (*Virtual Network Computing*) es un protocolo que permite acceder a la interfaz gráfica a través de una interfaz web mediante la exposición del puerto VCN. Habitualmente, esto requiere la instalación de un cliente VNC, lo cual se ha querido evitar en este trabajo. En vez de realizar la instalación de la herramienta, he decidido usar la biblioteca noVNC, que actúa de intermediaria entre la conexión web y el servidor VNC a través de un proxy [3]. Esta funcionalidad integra un proxy noVNC asociado a un servidor VNC concreto mediante la instrucción:

```
../noVNC/utils/novnc_proxy --vnc localhost:5900 --listen 6080 &
```

En este caso, el proxy conecta el servidor noVNC, conectándolo al servidor VNC de la máquina virtual, la cual expone su interfaz web en el puerto 5900 y la publica a un servidor web a través del puerto 6080. Esta acción se realiza para cada una de las máquinas QEMU, realizando el siguiente mapeo de puertos:

- IDS externo: VNC localhost:5900 → noVNC en 6080
- IDS interno: VNC localhost:5901 → noVNC en 6081
- Kali interno: VNC localhost:5902 → noVNC en 6082
- Kali externo: VNC localhost:5903 → noVNC en 6083

Para poder ejecutar una interfaz web en Docker como en QEMU de forma automática, se añaden al script *execute.sh*, que facilita la ejecución de la aplicación. Para hacer uso de estas tecnologías es necesario tener acceso por `ssh` a la máquina. Debido a que el contenedor asignado por el profesor para realizar el proyecto no posee `ssh` (además de otras herramientas que más tarde se usarán para el dashboard), se debe realizar la instalación. Debido a que los contenedores de GNS3 no guardan toda la información y para no tener que realizar esta instalación cada vez que se inicia, en el Anexo A se describen los pasos necesarios para la creación de un nuevo contenedor que incluya los paquetes necesarios, además de los ya existentes. A la hora de utilizar `ssh` se debe haber activado previamente mediante el comando `/usr/sbin/sshd`, por lo que se crea la carpeta `mkdir -p /run/sshd` y se le asignan permisos mediante `sudo chmod 755 /run/sshd`. Para poder guardar de forma permanente la carpeta, se guarda en GNS3 mediante la configuración de la figura 5.3.

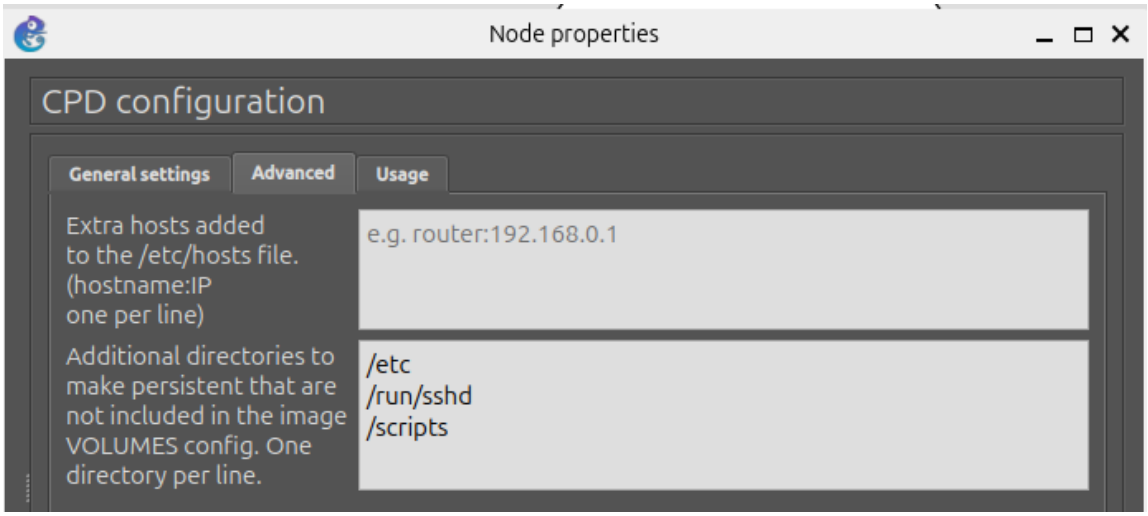


Figura 5.3: Guardar nueva carpeta en GNS3

[3]Componente intermedio que recibe la conexión de un cliente y se la reenvía al servidor, ocultando la adaptación del protocolo.

Para realizar la prueba de su correcto funcionamiento, se puede ejecutar la siguiente prueba: `ssh -b 2001:db8:abcd::101 root@2001:db8:abcd::27 ïp a"`

Como resultado, se obtiene en la figura 5.4 un grafo interactuable que facilita un rápido acceso a cada uno de los nodos de la red.

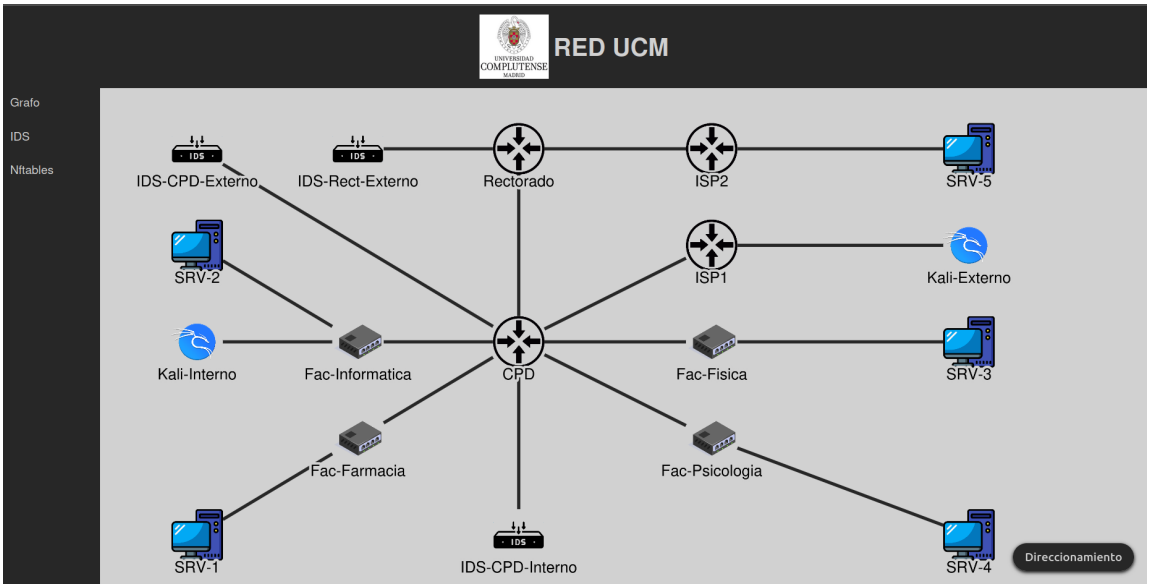


Figura 5.4: Grafo con acceso a los nodos

5.2.2. Creación de alertas

En esta sección se describe la creación de alertas por parte de Suricata. Como ya se ha visto en la sección 4.5.3, el tráfico proveniente de las facultades se duplica a IDS-CPD-Interno, el que proviene de ETSIT va a IDS-CPD-Externo y, en caso de backup, el tráfico proveniente de Puente de los Franceses se duplica a IDS-Rectorado. Para que Suricata pueda monitorizar el tráfico de una interfaz, se debe ejecutar el comando `suricata -c /etc/suricata/suricata.yaml -i ens3`. La gestión automática de alertas se realiza mediante la instalación de `suricata-updates`, que al ejecutar con el comando `suricata-updates()`, instala las reglas en `/var/lib/suricata/rules`. La configuración principal de Suricata se encuentra en `/etc/suricata/suricata.yaml`.

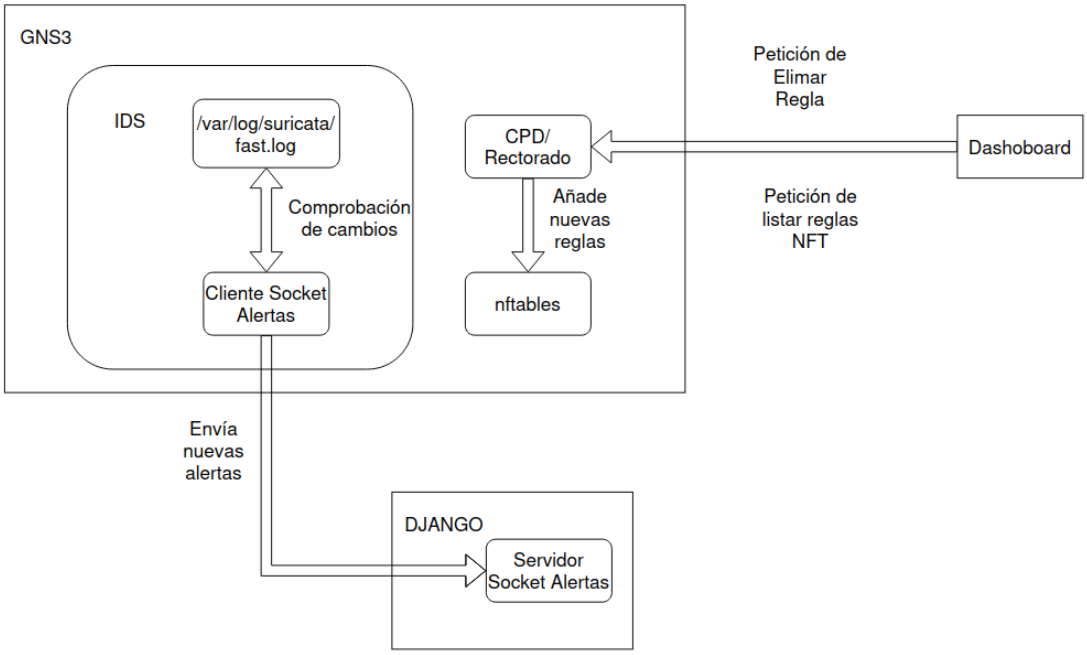


Figura 5.5: Grafo con acceso a los nodos

Para poder tener las alertas creadas en los IDS, primero se debe conseguir una comunicación entre las máquinas de GNS3 y Django. Debido a que se trata de un sistema en tiempo real, se ha propuesto realizar la comunicación mediante *sockets*[4] entre los dos componentes. Mientras que Django actúa como el servidor que está a la escucha de conexiones, los contenedores -específicamente los routers- GNS3 son los clientes que se conectan al servidor para mandar las alertas.

En primer lugar, debe ser el servidor el que empiece a escuchar y el cliente el que se intente conectar a él. Una vez realizada la conexión se realiza una comprobación constante del archivo `/var/log/suricata/fast.log` por parte del cliente, que, como se ha visto en el capítulo 3.1.2, es donde se almacenan las alertas en BSON. Esta acción se realiza posicionando el cursor al final del fichero (*file.seek*) y leyéndolo de forma continua (*file.readline()*). Una vez se realice una escritura, gracias a que Suricata la escribe de forma conjunta al estar almacenada en un buffer, se envía al servidor y se sigue con la misma acción de esperar la próxima alerta.

El servidor realiza una escucha continua para saber qué cliente se intenta conectar a él y por cada máquina que se conecte, crea un *thread* para procesar la comunicación de forma paralela. Por cada línea que recibe se realizan dos acciones:

- Realizar la conversión de los datos a JSON.
- Insertar la alerta en la colección MongoDB correspondiente.

Como se ha indicado anteriormente, la idea de la pestaña IDS es mostrar las alertas generadas por Suricata en los tres IDS. Para conseguirlo, se muestra una tabla con las alertas registradas. La idea es que tras recibir una alerta, se pueda actuar con rapidez en la mitigación, implementando una nueva regla que restrinja el tipo de tráfico específico.

[4]Canal de comunicación a través de la red

Desde la propia ventana del dashboard mostrado en la figura 5.6 se tiene un botón para realizar un limpiado de alerta. Este botón es realmente útil a la hora de realizar pruebas y comprobar que las nuevas reglas *nftables* insertadas para bloquear un ataque están haciendo efecto.

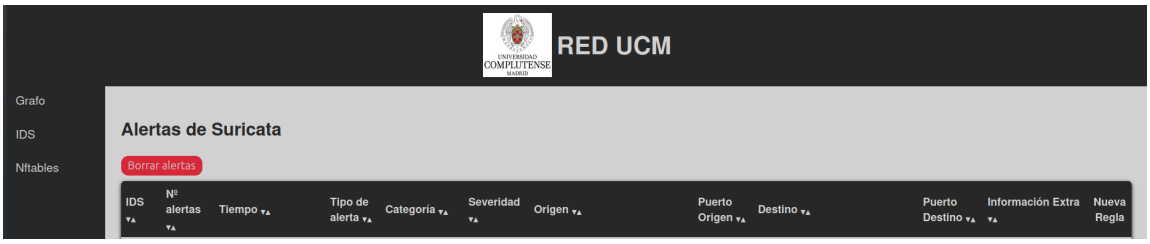


Figura 5.6: Dashboard alertas Suricata

5.2.3. Almacenamiento de alertas

Antes de pasar a explicar cómo se realizan las comunicaciones entre las distintas partes que conforman el sistema de mitigación, se debe explicar el sistema de almacenamiento de las alertas generadas por Suricata. Para conseguirlo, se ha decidido hacer uso de MongoDB, una base de datos NoSQL de código abierto, que realiza uso de documentos flexibles para procesar y almacenar múltiples tipos de datos.

MongoDB usa el formato de almacenamiento BSON, el cual está basado en JSON. Esto hace que en el momento en el que se realiza la transmisión de la alerta entre GNS3 y Django se puede realizar una normalización de los datos para posteriormente ser almacenados.

Una de las principales ventajas de MongoDB es su flexibilidad. Al tener una arquitectura de esquema que funciona con datos no estructurados, los datos no tienen por qué estar predefinidos y los esquemas se pueden variar dinámicamente. No obstante, para definir la estructura de datos y que los datos sean normalizados para ser mostrados en el dashboard, se ha utilizado ORM (*Object-Relational Mapping*). ORM actúa como capa intermedia entre el dashboard y la base de datos.

La única complicación es que Django está diseñado para usar bases de datos relacionales, siendo MongoDB no relacional. Es por ello que se introduce Djongo, una herramienta de *backend* para Django que traduce las consultas SQL escritas en Python (ORM) de Django a consultas de MongoDB.

La base de datos utilizada en este trabajo se llama `ipv6_dashboard` y las colecciones dentro de la base de datos se llaman `alertas_suricata_cpd` para el IDS interno y externo del CPD y `alertas_suricata_rectorado` para el IDS del Rectorado.

Cabe destacar que, debido a la herramienta *ChangeStreams* que se utilizará en el capítulo 5.2.4, la base de datos debe ser una réplica.

5.2.4. Comunicación entre Django/MongoDB y Dashboard

Para entender cómo se obtiene una alerta desde que el servidor *socket* guarda la misma en MongoDB, hasta que es mostrada en el dashboard, se necesitan una serie de pasos intermedios. En este capítulo se describen estos pasos, que son aquellos dispuestos en la figura 5.7.

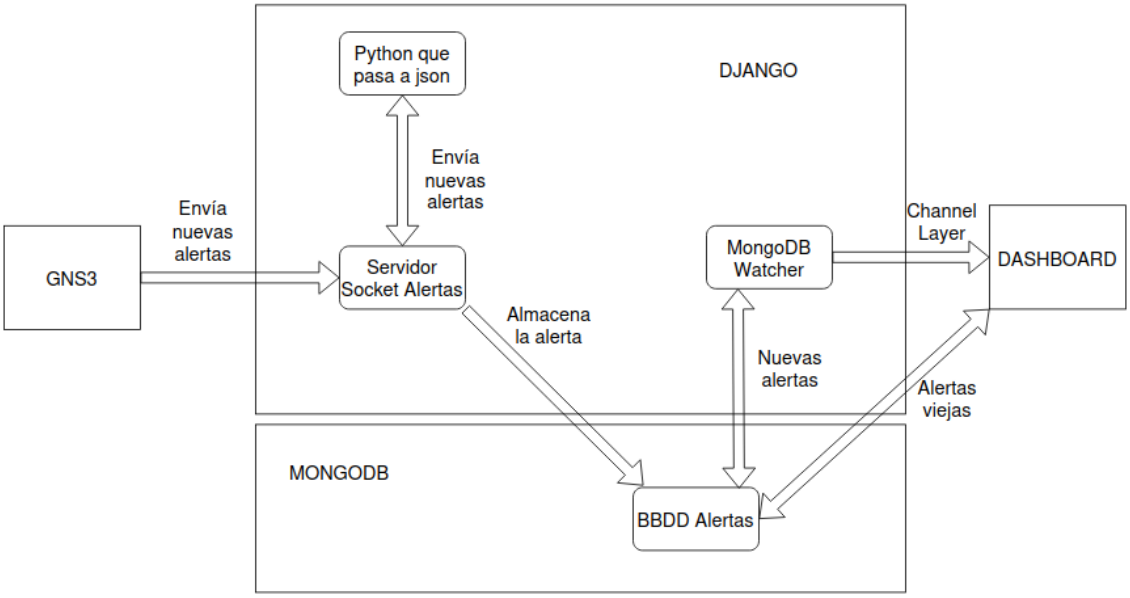


Figura 5.7: Comunicaciones entre Django, MongoDB y el *dashboard*

Monitorización de MongoDB

El dashboard obtiene la información de la base de datos de dos maneras: en primer lugar, cuando un usuario entra en el dashboard o recarga la página se realiza una petición a MongoDB de las alertas almacenadas para obtener el historial. En segundo lugar, es necesario que las alertas generadas por Suricata sean mostradas en el dashboard en tiempo real y para conseguirlo se ha utilizado *WebSockets* y el servidor ASGI, explicado en los siguientes capítulos.

Debido a que dentro de la BBDD se tienen distintas colecciones, primero se necesita saber cuándo se ha añadido la nueva alerta. Para conseguir esta información, por cada colección existente, se inicializa un visualizado/*watcher* que permanezca a la escucha de las operaciones. En este caso, sólo importan las operaciones de *insert*, por lo que las instrucciones necesarias son las siguientes:

```
with collection.watch(full_document="updateLookup") as stream:
    for change in stream:
        if change.get("operationType") == "insert":
```

Por cada inserción se crea un diccionario con el modelo de los datos de la alerta guardado en `models.py`.

WSGI frente a ASGI

Antes de pasar a explicar la comunicación en tiempo real, se deben entender las diferencias entre WSGI y ASGI. WSGI (ejecutado mediante `runserver`) es un protocolo síncrono que maneja únicamente comunicaciones HTTP basadas en el modelo *request/response* y funciona como modo de defecto en Django. Bajo esta premisa, el flujo de comunicación es el siguiente:

Cliente → *HTTP Request* → Servidor WSGI → Procesamiento → *HTTP Response* → Cliente

Por otro lado, ASGI es el protocolo asíncrono equivalente a WSGI. Este protocolo posibilita el uso de conexiones persistentes, ya sea mediante HTTP o mediante *WebSockets*. *WebSockets* es un protocolo de red, que, al igual que HTTP, funciona a través del protocolo TCP. Realiza

comunicaciones bidireccionales entre dos dispositivos, a diferencia de HTTP que sólo las permite conexiones del cliente al servidor. Además, añade que no se tenga que crear una nueva conexión cada vez que se quiera transmitir información, por lo que el canal entre ambas partes queda abierto. Por tanto, la comunicación queda:

Cliente ↔ *WebSocket* ↔ Servidor ASGI (conexión abierta)

Debido a la necesidad de usar ASGI para realizar la comunicación en tiempo real, se ha decidido usar *Daphne* como servidor, por lo que la instrucción a ejecutar es: `daphne -b 0.0.0.0 -p 8000 gns3_dashboard.asgi:application`. Daphne [54] es un servidor ASGI, desarrollado para el uso de *Django Channels* en aplicaciones asíncronas. Maneja tanto peticiones HTTP como conexiones *WebSocket* de forma simultánea.

Envío de alertas mediante *Channel Layers*

Una vez se tiene la estructura de datos gracias al watcher del capítulo 5.2.4, cada alerta se publica dentro del grupo llamado "alertas" de *Channel Layer* mediante:

```
async_to_sync(channel_layer.group_send)(
    "alertas",
    {"type": "alerta_message", "data": data}
)
```

Esta llamada invoca al consumidor *WebSocket* definido en `channel_layer_consumers.py`, que incluye funciones para conectarse y desconectarse del grupo, así como la función `alerta_message` para enviar alertas al cliente. La función `connect` permite unirse al mismo grupo "alertas", recibir los eventos por haber invocado al tipo `alerta_message`, normalizar ciertos campos y enviar el JSON al cliente *WebSocket* a través de `await self.send(text_data=json.dumps(data))`.

Configuración ASGI

Para que estas comunicaciones funcionen correctamente, es fundamental la configuración del fichero `asgi.py`. En él se inicializa el *watcher* de MongoDB y se configura *ProtocolTypeRouter* [55], un componente ASGI que redirige cada conexión al manejador adecuado dependiendo del protocolo, ya sea HTTP o *WebSocket*. Se configura de la siguiente forma:

```
ProtocolTypeRouter({
    "http": django\_asgi\_app,
    "websocket": AuthMiddlewareStack(
        URLRouter(
            websocket\_urlpatterns
        )
    ),
})
```

Para integrar el encaminado *WebSocket*, se importa `websocket_urlpatters` desde `routing.py`.

```
websocket_urlpatterns = [
    re_path(r'ws/alertas/$', channel_layer_consumers.
       SuricataConsumer.as_asgi()),
]
```

Comunicación con el cliente Web

El propio cliente *WebSocket* –el navegador– es el que establece una conexión a `ws://servidor/ws/alertas/`. Gracias a la instrucción `self.send()` realizada por Channel Layers para enviar el JSON exclusivamente al cliente, con los datos recibidos se crea la nueva línea de la tabla de IDS y se

añade al historial traído de MongoDB a través de `appendChild()`. Este código se ejecuta en la función `document.addEventListener("DOMContentLoaded", (...))` en `static/js/script.js` y está asociado a `ids.html` a través de `base.html`. Esta función se mantiene a la escucha de un evento y ejecuta el bloque que añade una alerta a la tabla IDS. También contiene la lógica de realizar el ordenado de la tabla según la columna donde se presione la flecha.

La comunicación completa queda así:

Suricata genera alerta → Servidor *socket* envía alerta al Cliente → Cliente *socket* guarda la alerta en MongoDB → *MongoDBWatcher* detecta el cambio y envía la alerta por *Channel Layer* → *Channel Layer* transforma y envía los datos → Cliente *WebSocket* (navegador) actualiza la UI (*User Interface*).

5.2.5. Mitigación

Para poder realizar la mitigación se ha creado un formulario, autocompletado con la información de la alerta, en el cual se crea una tabla *nftables* específica para cada mitigación de ataque con un conjunto de reglas que bloqueen el tipo de tráfico necesario. Debido a que estos no son los únicos parámetros posibles a introducir, también se da la opción de editar la regla una vez acabado el formulario.

Creación de reglas

Para la creación de reglas, como ya se ha visto en el apartado 3.1.5, se utiliza *nftables*. Para permitir una mitigación de ataque de forma rápida y eficaz, se ha decidido dar la opción de crear la propia regla en el dashboard. Es por ello que cada alerta o conjunto de alertas tiene un botón `+` al final de la alerta que lleva al formulario. A continuación, se muestran los distintos pasos que se deben seguir para crear una regla *nftables*. Cabe mencionar que no todas las opciones de la herramienta se han añadido a través del formulario.

En primer lugar, como se observa en la figura 5.8, se dan las casillas para introducir características generales de la regla:

- IDS: nodo donde se introduce la regla, pudiendo ser CPD/Rectorado.
- Familia: nombre de la familia, pudiendo ser *ip*, *ip6*, *inet*, *arp*, *bridge* o *netdev*.
- Tabla: nombre de la tabla.
- Cadena: nombre de la cadena.
- Modo de inserción de la nueva regla, dando la opción de realizar *add*, *replace* o *insert*.
- Índice: posición dentro de la tabla.
- Prioridad: prioridad de la cadena en caso de ser nueva.
- Comentario: descripción de la regla. Debido a la versión de *nftables*, no debe haber espacios, por lo que se sustituyen por `"_"`.

Figura 5.8: Formulario: Añadir información principal de la regla

En segundo lugar, como se observa en la figura 5.9, se permiten añadir las opciones para comparar la regla con el paquete y comprobar si coincide. Para esto se dan las siguientes opciones:

- Protocolo para filtrar en capa 4. Puede ser tcp, udp, icmp, icmpv6, all.
- IP Origen/Destino: dirección o red de origen y destino (CIDR).
- Set Origen/Destino: conjunto de *nftables* predefinidos. Un ejemplo de ello es crear un conjunto de IP.

  ```
  nft add set inet filter allowed_admins { type ipv4_addr\;
     flags interval\; }
  nft add element inet filter allowed_admins { 192.168.1.10,
     192.168.1.20 }
  ```

  Para después asignar a la regla:

  ```
  ip saddr @allowed_admins tcp dport 22 accept.
  ```

- Puerto Origen/Destino: puertos de origen y destino. Es importante mencionar que, debido a que se están realizando ataques ICMPv6, el uso del campo puerto origen y puerto destino no tiene sentido en paquetes de este protocolo. Como consecuencia, Suricata reutiliza estos campos como ayuda para identificar el tipo de mensaje que se está realizando. A continuación, se muestra una tabla de los valores más utilizados para puerto origen/destino:
  - 128 → 0 : *Echo request*
  - 129 → 0 : *Echo reply*
  - 133 → 0 : *Router Solicitation*
  - 134 → 0 : *Router Advertisement*
  - 135 → 0 : *Neighbor Solicitation*
  - 136 → 0 : *Neighbor Advertisement*
  - 137 → 0 : *Redirect*

  - 130 → 0 : *Multicast Listener Query*
  - 131 → 0 : *Multicast Listener Report*
  - 132 → 0 : *Multicast Listener Done*
- Interfaz de entrada/salida.
- *CT state*: también llamado *conntrack*, indica el estado de la conexión, pudiendo ser *new*, *established*, *related* o *invalid*. Un par de ejemplos son:

```
ct state new tcp dport 443 accept
ct state established,related accept
```

- Flags extra: expresiones nft avanzadas.
- Invertir *match*, ya sea de la red con !src o excluir el destino con !dst.

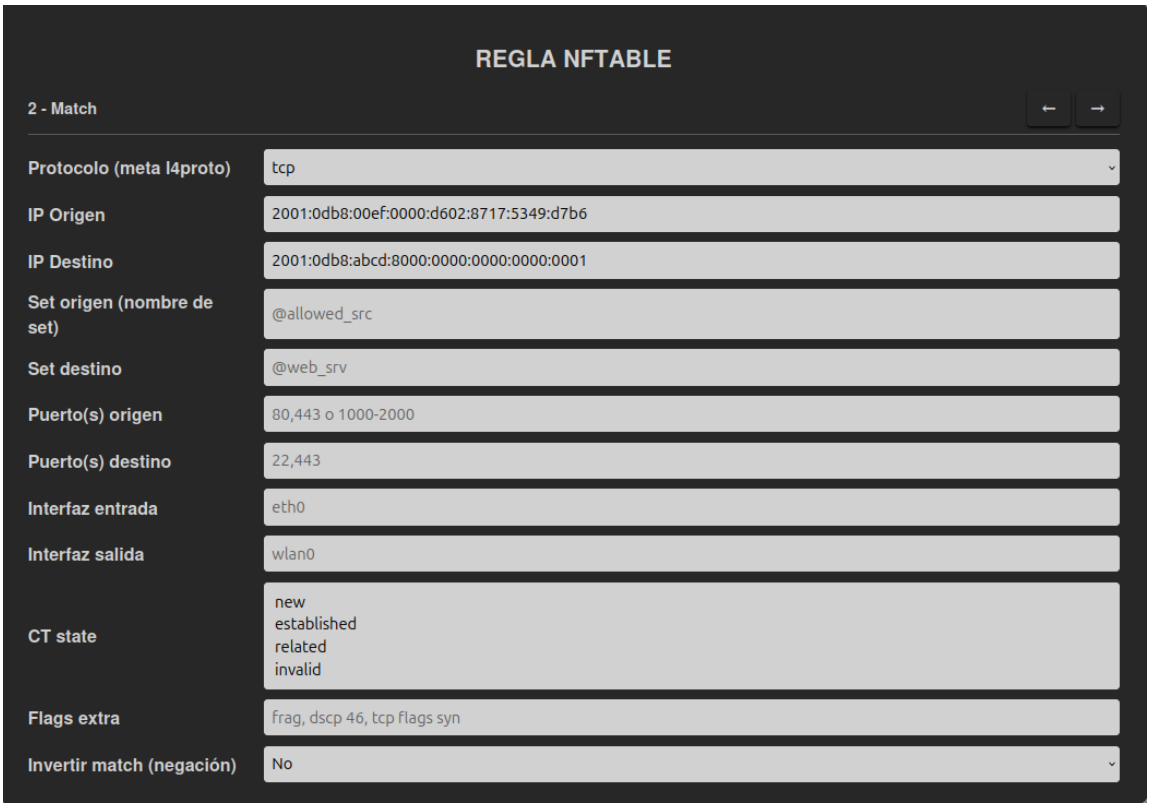


Figura 5.9: Formulario: añadir parámetros para comparar paquete

- Cadena destino: sirve para realizar saltos entre cadenas, haciendo que se pueda reutilizar la lógica y modular reglas.
- SNAT IP: traducción de origen, en caso de utilizar *postrouting*.
- SNAT puerto: puerto para SNAT.
- DNAT IP: redirección de destino.

- DNAT puerto: puerto para DNAT.
- *Log prefix*: prefijo identificativo en los logs. Puede ser info, que indica que el tráfico es permitido, *warning* que indica sospecha, error que indica bloqueo crítico y *debug*. Ejemplo: `tcp dport 22 log prefix "SSH-BLOCK: "drop`.
- *Reject type*: mensaje del tipo de rechazo activo, a diferencia de drop, reject responde al origen.
- *Meta/mark*: sirve para asignar una marca interna al paquete y con él poder realizar un routing más avanzado, QoS o clasificación de paquetes.

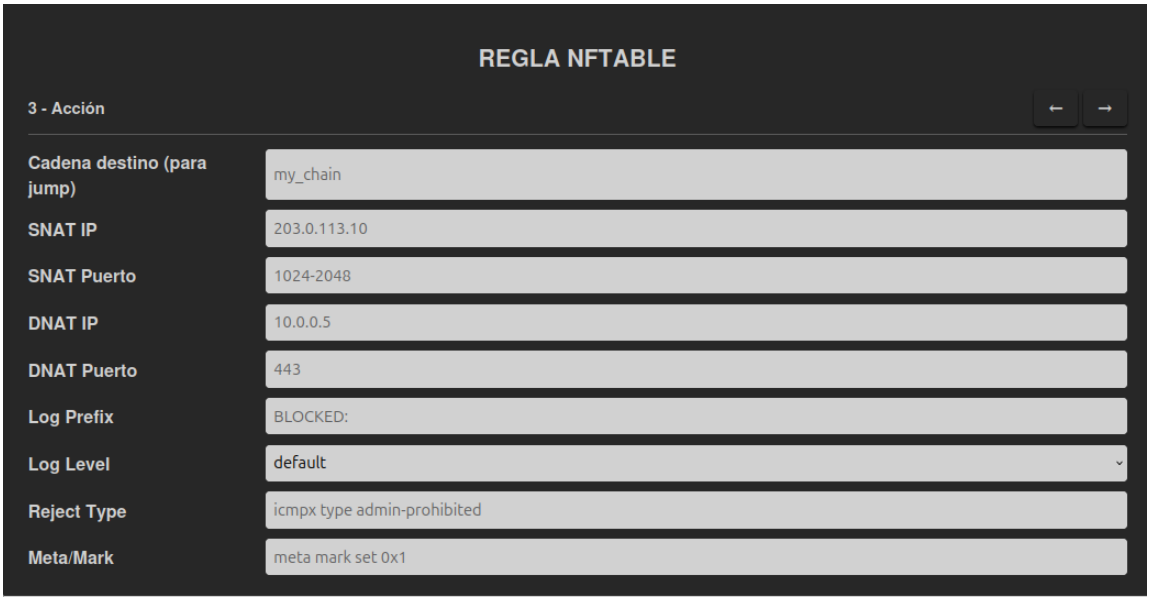


Figura 5.10: Formulario: añadir parámetros para NAT

Por último, se tiene el formulario para limitar el QoS del tráfico. Como opciones se puede incluir:

- *Limit rate*: se realiza una limitación del servicio.
- *Burst*: ráfaga inicial permitida.
- *Hashlimit*: limitar por IP origen.
- *DSCP set*: prioriza cierto tráfico.
- Expresión extra.

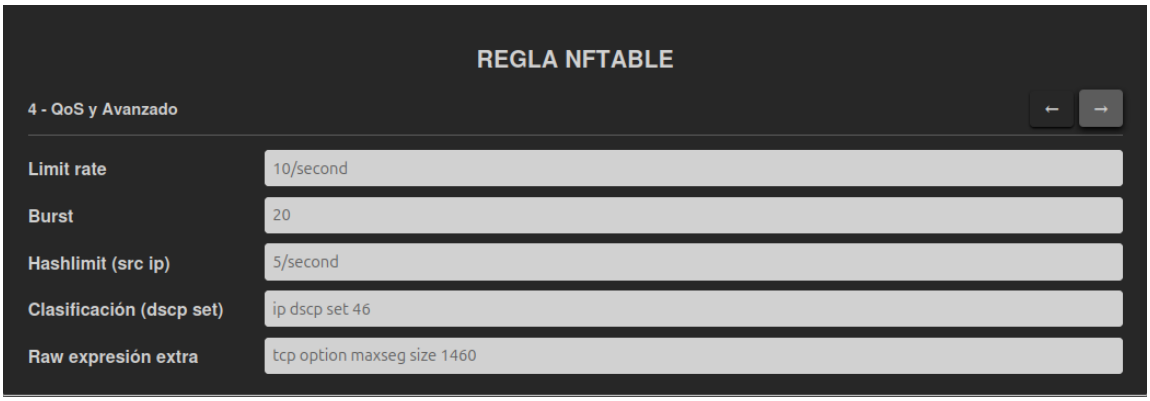


Figura 5.11: Formulario: añadir QoS

Una vez finalizado el formulario, se da la opción, como se observa en la figura 5.12, de revisar la regla y realizar modificaciones.

Figura 5.12: Formulario: Editar regla

Una vez completada, se ejecuta dándole al botón de `Añadir regla`. Acto seguido, el sistema proporciona el feedback correspondiente para saber si la regla se ha añadido correctamente (figura 5.13) o si ha habido algún error (figura 5.14).

Figura 5.13: Mensaje de éxito al ejecutar la regla

Figura 5.14: Mensaje de error al ejecutar la regla

Ejecución en los routers

La idea inicial de conectar el dashboard con los routers fue realizar una nueva conexión *socket* cliente-servidor y tener una nueva base de datos para guardar mediante índices las *nftables*. Debido a la complejidad, la gran carga de trabajo que supone la actualización de la BBDD de forma

constante y la no necesidad de realizar este proceso en tiempo real se abandonó la idea para mejorar la eficiencia del dashboard. Finalmente, la conexión se ha realizado mediante *Ansible* y un *endpoint*. Ansible es una herramienta DevOps capaz de automatizar tareas en cualquier dispositivo, en este caso llamado host o nodo. Ansible sustituye, en este caso, la tarea de realizar la ejecución de la regla en el nodo correspondiente. Esto se produce a través de instrucciones – llamadas «Playbooks» – que se ejecutan sobre los hosts que se indiquen. Para hacer uso de Ansible es tan simple como:

- Crear un registro con los *hosts* que se quieren utilizar, en este caso se separan en dos grupos para poder diferenciar entre los IDS.

```
[grupo_rectorado]
rectorado ansible_host=192.168.122.200 ansible_user=root
   ansible_ssh_pass=1234 ansible_python_interpreter=/usr/bin/
   python3

[grupo_cpd]
cpd ansible_host=192.168.122.201 ansible_user=root
   ansible_ssh_pass=12345 ansible_python_interpreter=/usr/bin/
   python3
```

- Crear un *playbook* en formato **yml** con la tarea a realizar, en este caso, ejecutar una regla, y una opción para almacenar el resultado.

```
- name: Ejecutar reglas NFT
  hosts: grupo_cpd, grupo_rectorado
  gather_facts: no
  vars:
    nft_rules: []

  tasks:
    - name: Aplicar cada regla NFT
      command: "{{ item }}"
      loop: "{{ nft_rules }}"
      register: nft_results
      ignore_errors: yes

    - name: Mostrar resultados
      debug:
        var: nft_results
```

Para poder visualizar las reglas *nftables* existentes hasta el momento y las creadas sobre la marcha se ha creado una nueva vista: *nftables*. Esta vista realiza una petición inicial mediante Ansible al Rectorado y CPD para obtener las *nftables* existentes y mostrarlas divididas en tablas.

Mitigación temporal

Para poder realizar un bloqueo de una IP de forma temporal se ha diseñado el programa (`counter.py`), el cual debe estar en continua ejecución en el CPD y Rectorado. Gracias a los parámetros *nftrace* y *counter* añadidos por defecto en todas las reglas creadas con el formulario, se comprueba si un ataque ya mitigado se sigue produciendo.

Gracias a la opción *nftrace*, se permite ejecutar la instrucción `nft monitor trace`, una herramienta de *nftables* que detecta cuando una regla es alcanzada por tráfico de red. De esta forma, mientras el ataque no se siga produciendo, el contador de la regla disminuye. En caso de que llegue a cero se levanta la suspensión a la IP o, en caso de que se siga produciendo el ataque, el contador

se reinicia. Debido a ello, existen dos casos: que se bloquee una IP porque se desconoce la forma que el atacante está realizando el ataque o que se bloquee una IP por realizar un ataque concreto.

Caso 1: se bloquean todas las comunicaciones a través de esta IP.

```
nft add rule netdev dup_table6 block_ipv6_eth2_dynamic ip6 saddr
   2001:db8:abcd:4000:0f76:a4a5:9c54:cb73  counter drop
```

Caso 2: se bloquean todos los mensajes ICMPv6 de tipo 200 a 205 provenientes de esta IP.

```
nft add rule netdev dup_table6 block_ipv6_eth2_dynamic ip6 saddr
   2001:0db8:abcd:4000:0f76:a4a5:9c54:cb73 ip6 nexthdr icmpv6 icmpv6
    type 200-205 counter drop
```

Para no realizar esta comprobación en todas las cadenas, he decidido crear cadenas *nftables* temporales, las cuales llevan `_dynamic` en el nombre. En primer lugar, el script `counter.py` obtiene las reglas mediante `nft -a list ruleset`. Tras encontrar las reglas que contengan `_dynamic` se realiza una monitorización en un hilo separado por cada una de las reglas. La monitorización consiste en comprobar cada cierto tiempo (`CHECK_INTERVAL`). En caso de que durante el tiempo (`INACTIVE_SECONDS`) no aumenten los campos de bytes/packets, es decir, no llegue ningún paquete que active la regla, se elimina el bloqueo impuesto a través del handle de la regla. En el caso de que se haya añadido una regla del segundo tipo y el ataque se siga produciendo, el tiempo de bloqueo `INACTIVE_SECONDS` se reinicia, alargando el bloqueo de manera indefinida mientras persista el ataque. Para realizar pruebas, se han establecido valores por defecto de `CHECK_INTERVAL` igual a un segundo e `INACTIVE_SECONDS` igual a 10 segundos. Estos valores se pueden variar para casos reales, haciendo, por ejemplo, que se compruebe la regla cada quince minutos y el tiempo de bloqueo sea de doce horas.

Lo que se realiza en este programa es tener un contador de segundos por cada regla que se reinicia cuando llega un trafico que coincide. En el momento en el que el contador llega a la hora configurada como máxima, se realiza un borrado de la tabla que contiene ambas reglas de bloqueo y chequeo.

Para realizar la comunicación entre los nodos de GNS3 y el dashboard se ha ejecutado un POST a la IP del ordenador local. Esto se ha conseguido ejecutando Django para que escuche en todas las direcciones IP (0.0.0.0 8000) y añadiendo como IDS pemitidas a las IPv4 adjudicadas a la conexión de CPD-NAT.

Cabe mencionar ciertas complicaciones a la hora de la creación de la regla. En primer lugar, se consideró realizar dos reglas distintas, una que invalide el tráfico y otra que compruebe que se sigue produciendo. Debido a que la opción *nftrace* se permite añadir a cualquier regla *nftable*, ambas acciones se pueden realizar en una sola regla. En segundo lugar, surgió el problema de bloquear el tráfico hacia el IDS pero que se siguiese encaminando hacia el destino. Debido al funcionamiento de las reglas, al aplicar el *mirroring* a nivel de capa 2 en tablas de tipo *netdev*, el tráfico ya había sido encaminado hacia el destino cuando se quería limitar en reglas de los distintos tipos. Como solución, se propuso realizar nuevas tablas de bloqueos de ataques de tipo *netdev*, pero cambiando la prioridad de la *chain*. Esto se realiza para que, dentro de *netdev*, se aplique antes que el *mirroring* , sin importar en qué orden estén escritas. Debido al funcionamiento de la prioridad en *nftables*, cuanto menor sea, antes se ejecuta.

5.2.6. Tabla de direccionamiento

Esto se ha realizado mediante `ssh` hacia la dirección IP de la máquina. Según el tipo de contenedor (Docker o QEMU) existen dos formas de obtener las IP de cada componente: Ejemplo de contenedores Docker

```
docker ps
```

Ejemplo de contenedores QEMU:

`arp -an (interfaces virtuales)`

Una vez se obtiene la IP a la que acceder a la VM, se realiza vía **ssh** la ejecución de `ip a` para listar las interfaces y sus direcciones IPv4 e IPv6. En el caso de fallo, devuelve la IP detectada por ARP. Como existen distintos nodos QEMU se intenta acceder primero con las credenciales de *osboxes* y, en caso de fallo, con las de *kali*.

Tabla de Direccionamiento

Host	Interfaz	Dirección IP
CPD		
	eth1	2001:db8:abcd:8000::1/64
	eth2	2001:db8:abcd:4000::1/64
	eth3	2001:db8:abcd:c000::1/64
	eth4	2001:db8:abcd:2000::1/64
	eth5	192.168.122.201
	eth6	2001:db8:abcd:6000::1/64
	eth7	2001:db8:abcd:e000::1/64
Rectorado		
	eth1	2001:db8:abcd:1000::1/64
	eth3	192.168.122.200
ISP1		
	eth1	2001:db8:ef::1/64
IPS-Externo		
	ens3	2001:db8:abcd:e000::2/64
	ens4	2001:db8:abcd:3000::2/64
	ens5	192.168.122.44

Figura 5.15: Tabla de direcciones de las máquinas

5.2.7. Mostrar *nftables*

Por último, se tiene la pestaña para mostrar las reglas *nftables*. En ella aparecen las reglas de ambos routers CPD y Rectorado. En cada acceso, el sistema ejecuta a través del *playbook* Ansible `get_nftable.yml` la llamada `nft -a list ruleset`. Cada tabla contiene su *handle*, sus cadenas y las reglas en ellas, cada una a su vez con su *handle*. Además, se incorpora un botón "Eliminar" para aquellas reglas temporales, identificadas por pertenecer a cadenas cuyo nombre finaliza con el sufijo `_dynamic`. Estas existen para el bloqueo de un puesto de trabajo durante un tiempo, permitiendo

la reactivación de la IP una vez se haya identificado la procedencia del ataque. Para ejecutar la acción, se lanza el *playbook* de Ansible `delete_nft_rule.yml`.

Para poder conocer si los ataques bloqueados se siguen produciendo, se debe observar si los campos *packets* o *bytes* en reglas de tipo *drop/reject* aumentan. Los contadores se reinician cada vez que se apaga/reinicia la máquina.

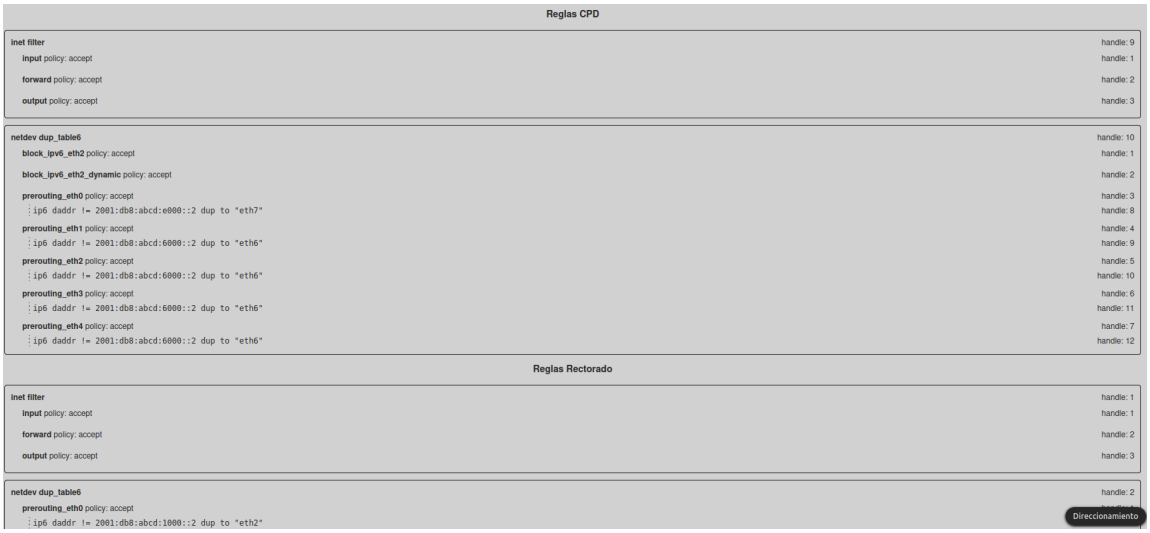


Figura 5.16: Listado de *nftables*

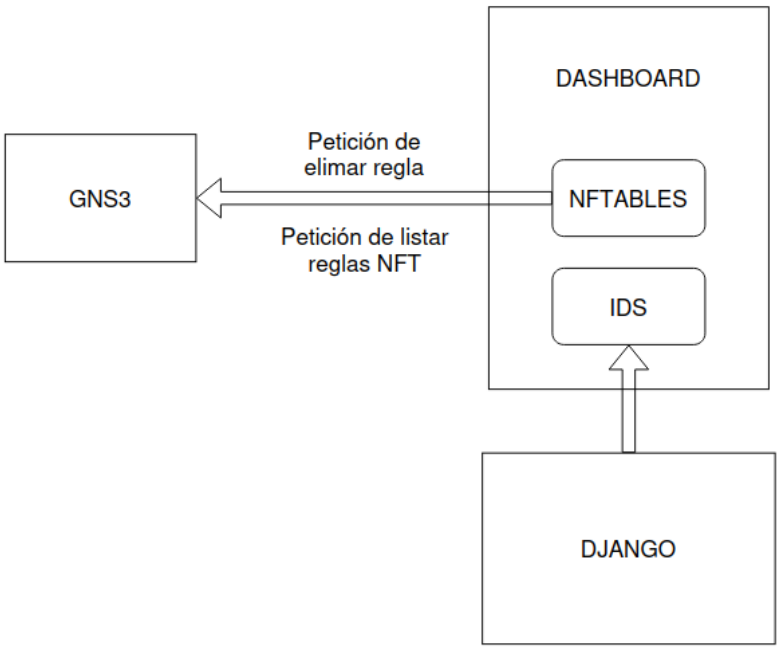


Figura 5.17: Dashboard alertas Suricata

5.2.8. Configuración adicional en Django

- `settings.py`: configuración global de proyecto Django. Las más importantes para este trabajo son las aplicaciones creadas (dashboard), el *middleware*, la carpeta con los ficheros estáticos como imágenes, la configuración ASGI/WSGI y la configuración de *channel layers*. También se guardan otras configuraciones, como la base de datos relacional en caso de hacer uso de ella.
- `urls.py`: asocia una URL con una vista y actúa como punto de entrada para las peticiones HTTP. En el proyecto existe uno para cada acción, ya sea mostrar cada vista web, abrir una terminal, borrar las alertas, borrar una regla, crear una nueva regla, etc.
- `wsgi.py`: archivo de configuración para WSGI, en este trabajo no se utiliza.
- `asgi.py`: archivo de configuración para ASGI. Se inicializa la comprobación de la base de datos MongoDB y se configura los Web Sockets.
- `db.sqlite3`: base de datos por defecto en Django, en este trabajo no se utiliza.
- `manage.py`: script de gestión del proyecto Django, no pertenece al propio proyecto.

A su vez, al crear una aplicación, como en este caso se crea `dashboard`, se genera de forma automática la siguiente estructura de ficheros:

- `apps.py`: en este archivo se configuran las aplicaciones registradas en `settings.py`. En este caso, se define la configuración de la aplicación `dashboard`.

- `models.py`: define los modelos. Aunque se esté usando una base de datos no relacional, se utiliza Djongo, que actúa como traductor SQL para poder usar el ORM de Django. Es por ello que se necesita una estructura tanto para las alertas generadas por Suricata como para las *nftables*.
- `views.py`: contiene la lógica de negocio. En este caso contiene las funciones necesarias para ejecutar cada uno de los playbooks, la lógica para abrir los distintos tipos de terminal, la obtención de alertas de MongoDB, además del parseo de datos correspondiente a cada acción.
- `migrations`: carpeta de migraciones que controla los cambios en la base de datos.
- `static`: conjunto de ficheros estáticos. Incluye imágenes, css, js, json.
- `templates`: carpeta con los html de cada aplicación, en este caso sólo existe la aplicación dashboard.

5.2.9. Licencias y bibliotecas utilizadas

A continuación, se muestra un listado de todas aquellas licencias y bibliotecas utilizadas para la realización del código.

- ansible 12.2.0
- ansible-core 2.19.4
- asgiref 3.9.1
- Daphne 4.1.1
- Channels 4.1.0
- Channels-Redis 4.1.0
- Django 5.2.7
- pymongo 3.11.4
- redis 6.4.0
- requests 2.32.4
- whitenoise 6.11.0
- Cytoscape 3.33.1
- WebSocket 8.19.0
- GNS3 2.2.54
- nftables 1.1.3
- Suricata
- Python 3.12.3
- JavaScript
- TTYD 1.7.4
- Telnet 2.5
- noVNC

Capítulo 6

Escenarios de ataque

Una vez desplegada la infraestructura y configurado el mecanismo de monitorización y creación de reglas mediante la interfaz web, se procede a ejecutar ataques IPv6 desde las máquinas Kali. Para poner a prueba el sistema, se ha utilizado la herramienta **Thc-Ipv6** [56], la cual proporciona una serie de utilidades que generan diferentes tipos de ataques IPv6. En los siguientes apartados se analizan los resultados obtenidos de varios ataques y los mecanismos que se proponen para mitigarlos. En la carpeta del repositorio de Github *attacks* se encuentran las capturas de *Wireshark* mostradas en cada apartado.

6.1. Denial6

Uno de los ataques que provocan una denegación de servicio en redes IPv6 es el implementado mediante la instrucción `atk6-denial6 [interfaz] [IPv6] [tipo]`. Esta herramienta genera distintos tipos de tráfico malintencionado en función del parámetro indicado, abarcando desde el envío de paquetes ICMPv6 que incorporan una cabecera de autenticación *AH*, hasta cabeceras *Hop-by-Hop* de gran tamaño que incluyen la opción *Router Alert*. En este caso, se ha elegido demostrar el funcionamiento del tipo 1, correspondiente a cabeceras de extensión de destino excesivamente largas con alertas de rutas y con múltiples opciones desconocidas.

En la figura 6.1 se muestra la generación de alertas de tipo *'IPv6 DSTOPTS unkown option'* en el sistema de monitorización creado. El ataque se realiza a través de la interfaz eth0 y la IP atacada es la `2001:db8:abcd:4000::1`, perteneciente a la interfaz eth2 del router CPD.

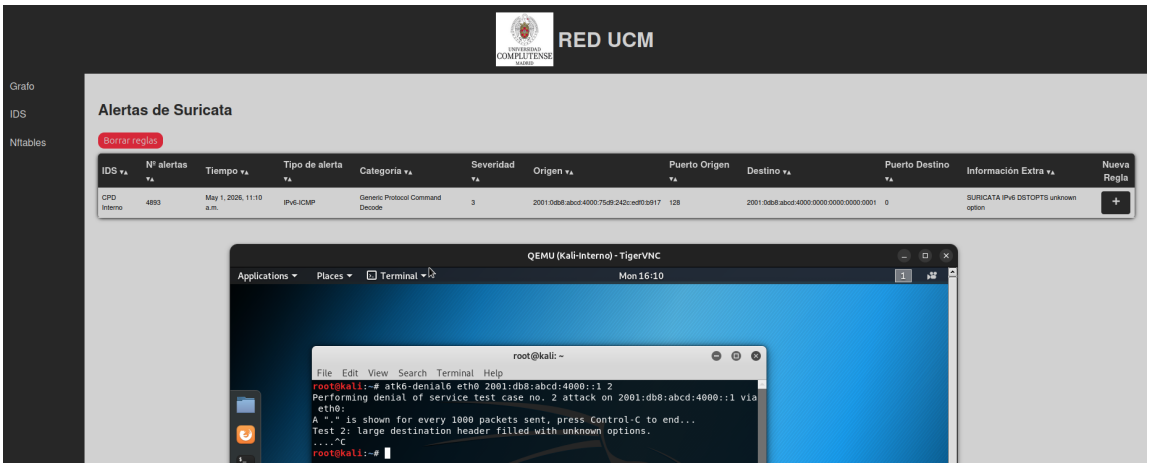


Figura 6.1: Captura de alertas del ataque *atk6-denial*, opción 1

Para poder observar cómo son los ataques que se producen hacia `2001:db8:abcd:4000::1`, GNS3 permite la captura de tráfico en cada uno de sus enlaces a través de *Wireshark*. Tras capturar el tráfico mientras se realiza el ataque, se observa que el paquete mostrado en las figuras 6.2 y 6.3 se genera mil veces por segundo.

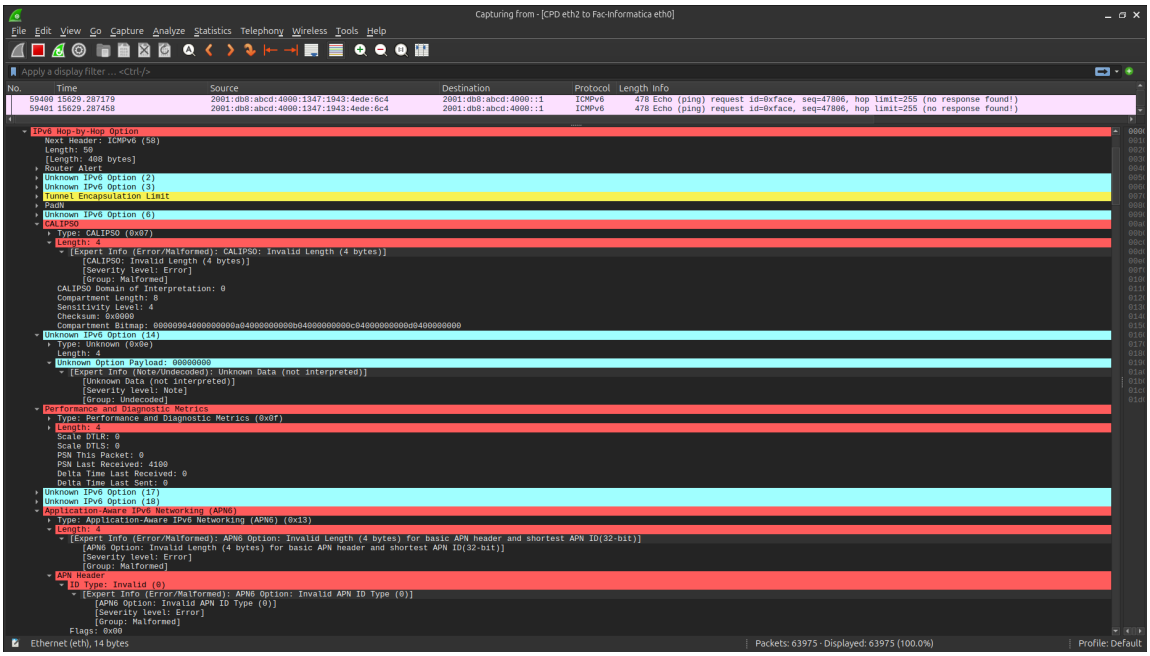


Figura 6.2: Captura de *Wireshark* del ataque *atk6-denial*, opción 1

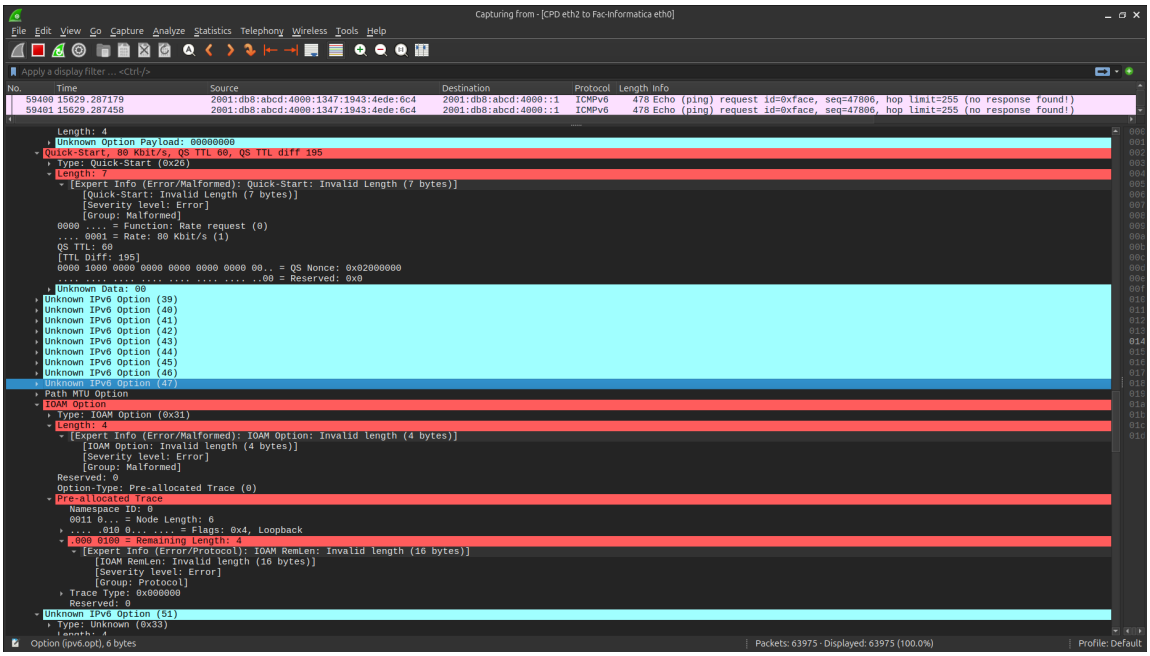


Figura 6.3: Captura de *Wireshark* del ataque *atk6-denial*, opción 1

Al analizar la captura, se observa que se está produciendo un ataque a través de la cabecera IPv6 *Hop-by-Hop* de un paquete ICMPv6 *Echo Request*. En el panel de *Wireshark* se puede observar cómo dicha cabecera aparece como *IPv6 Option* y *Error/Malformed*, destacando errores en opciones como *CALIPSO, Performance and Diagnostic Metrics (PDM), Application-Aware IPv6 Networking (APN6), Quick Start* e *IOAM Option*. A continuación, se explica qué significado y uso corresponde a cada cabecera.

- CALIPSO (*Common Architecture Label IPv6 Security Option*) [57] se trata de una cabecera opcional en la cabecera *Hop-by-Hop* que permite marcar ciertos paquetes como sensibles.
- PDM (*Performance and Diagnostic Metrics* [58] es una cabecera de *Destination Option Header* que realiza una medición del rendimiento de la red con el fin de diagnosticar problemas en ella.
- APN6 (*Application-Aware IPv6 Networking)* [59] es una cabecera que ofrece la información detallada sobre la red, destinado para los IDS e IPS.
- *Quick-Start* [60] es un mecanismo definido para IPv6 en el que, a través de un emisor, se solicita explícitamente una mayor velocidad de transmisión inicial.
- *IOAM option (In Situ Operations, Administration, and Maintenance)* [61] es un conjunto de opciones que recopilan información operativa y de monitorización directamente en los paquetes de datos mientras atraviesan la red.

Una forma de mitigar este ataque es limitar el número de paquetes permitidos que contengan cabeceras de fragmentación (*Fragmentation Header*), enrutamiento (*Routing Header*), límite de saltos (*Hop-by-hop Options Header*) u opciones de destino (*Destination Options Header*) a diez paquetes por segundo, lo que hace reducir el número de alertas alrededor de un 99 %. Esto se puede realizar con la instrucción:

```
nft add rule netdev dup_table6 block_ipv6_eth2 ip6 exthdr frag,
    route,hop,dst limit rate over 10/second drop
```

Debido a que el número de paquetes generados por la herramienta es de mil por segundo, todo el tráfico que no cumple esta limitación es eliminado por el router CPD, como se observa en la figura 6.4.

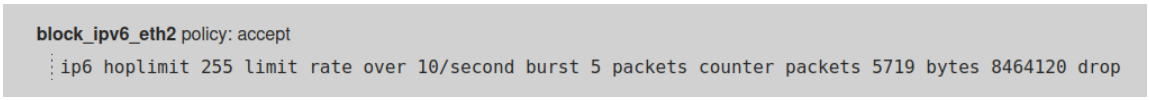


Figura 6.4: Demostración del bloqueo de paquetes en el ataque *atk6-denial*

6.2. *Flood solicitate6*

Entre los ataques evaluados con esta herramienta se encuentra `atk6-flood_solicitate6`, el cual genera una cantidad de tráfico masivo con distintas IPv6 origen de tipo *link-local*. El objetivo es saturar los mecanismos de resolución de redes, haciendo imposible que un tercero pueda conectarse y que el CPD sepa el origen del tráfico.

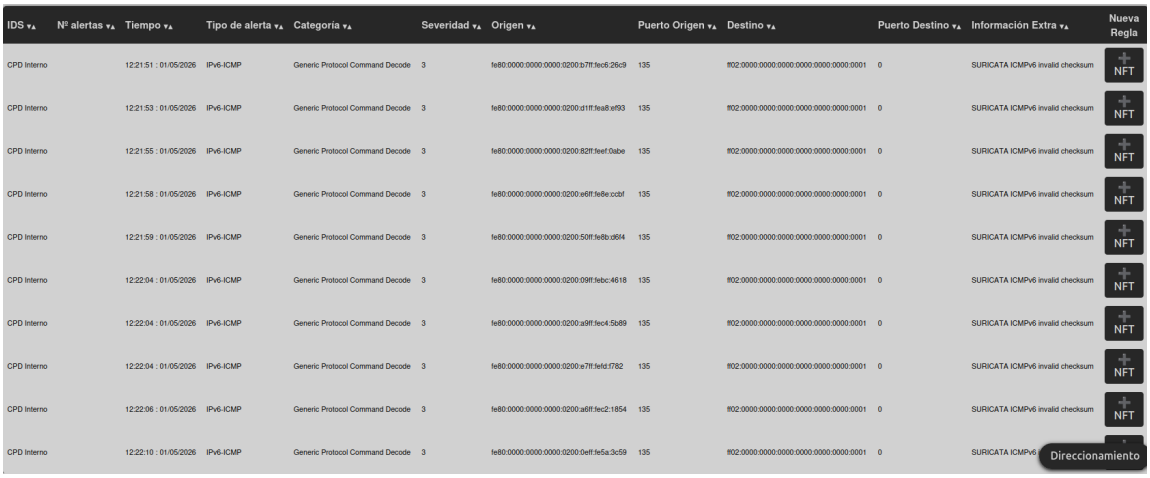


Figura 6.5: Captura de alertas del ataque *flood solicitate6*

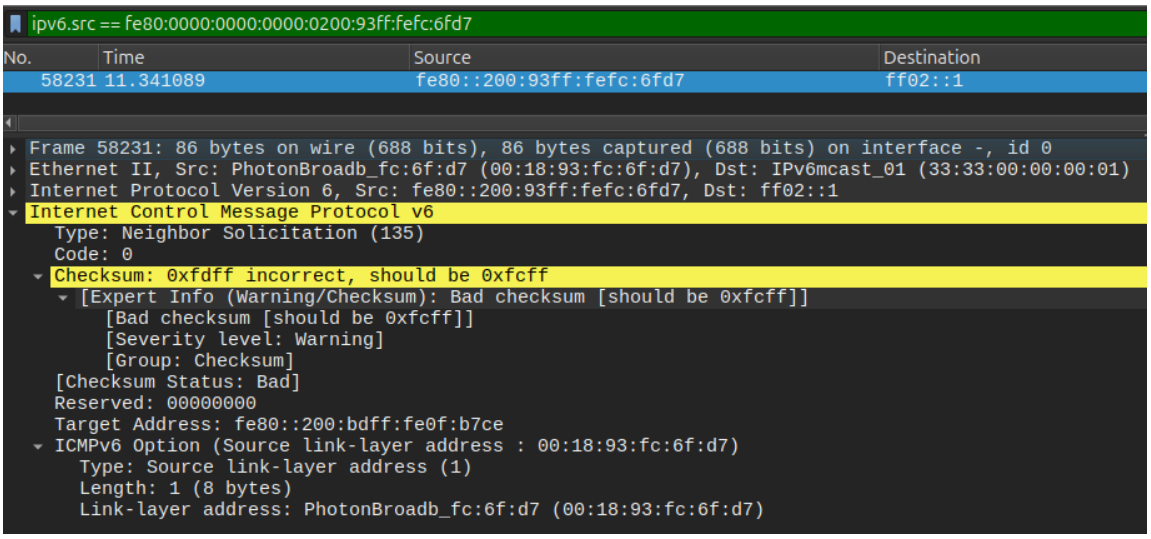


Figura 6.6: Captura de *Wireshark* del ataque mediante *flood solicitate6*

Dado que bloquear por completo los mensajes de tipo *Neighbour Solicitation* no es una opción viable, ya que impediría el funcionamiento de IPv6, la mitigación del ataque se ha realizado limitando el número de mensajes permitidos por segundo. Para ello, se han configurado dos reglas: la primera que permita únicamente un número controlado de solicitudes(10 por segundo y un pico máximo de 20 paquetes) y la segunda que descarte el exceso.

Regla 1: `nft add rule netdev dup_table6 block_ipv6_eth2 ip6 nexthdr ipv6-icmp icmpv6 type nd-neighbor-solicit limit rate 10/second burst 20 packets counter accept`

Regla 2: `nft add rule netdev dup_table6 block_ipv6_eth2 ip6 nexthdr ipv6-icmp icmpv6 type nd-neighbor-solicit counter drop`

6.3. *Exploit6*

El ataque `atk6-exploit6` también se realiza a partir de la interfaz eth0 hacia 2001:db8:abcd:4000::1. Permite explotar distintos tipos de vulnerabilidades de IPv6, consiguiendo el resultado mostrado en la figura 6.7.

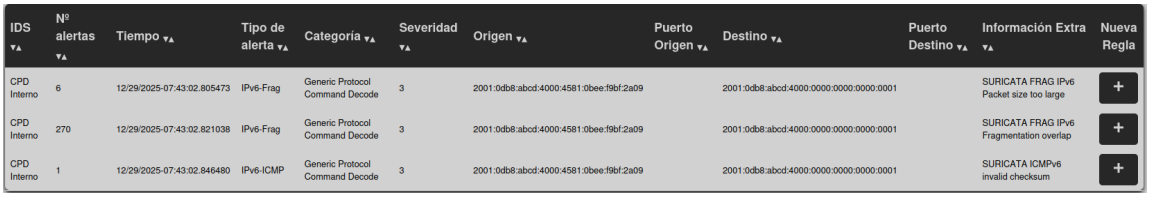


Figura 6.7: Captura de alertas del ataque *exploit6*

Al obtener distintos tipos de ataques, la captura de *Wireshark* en la figura 6.8 facilita el trabajo para obtener qué tipo de tráfico genera las alertas.

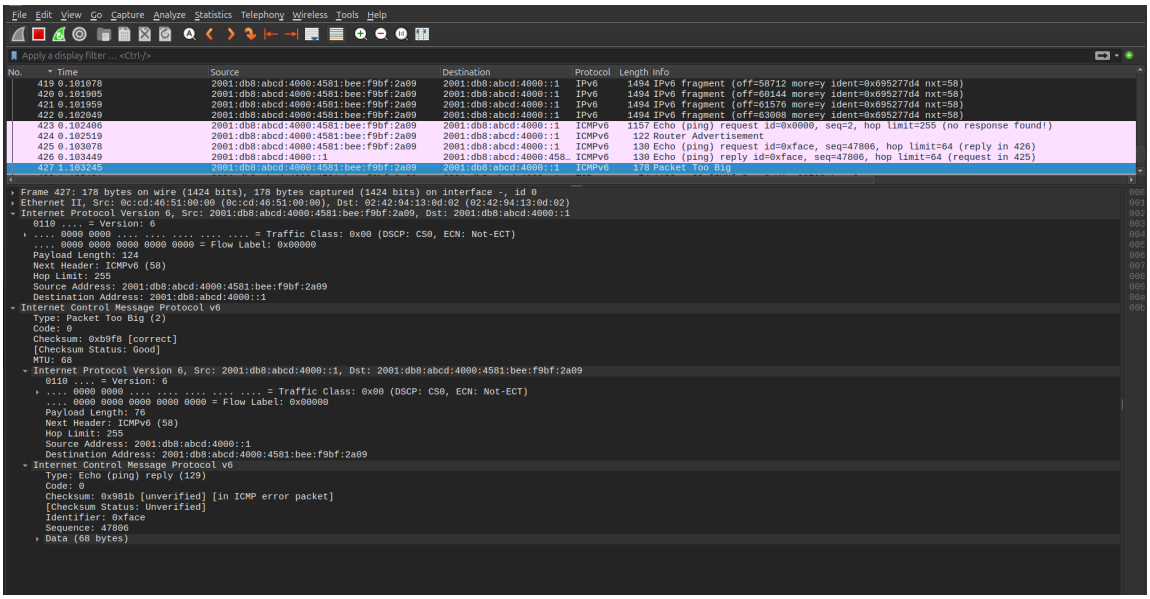


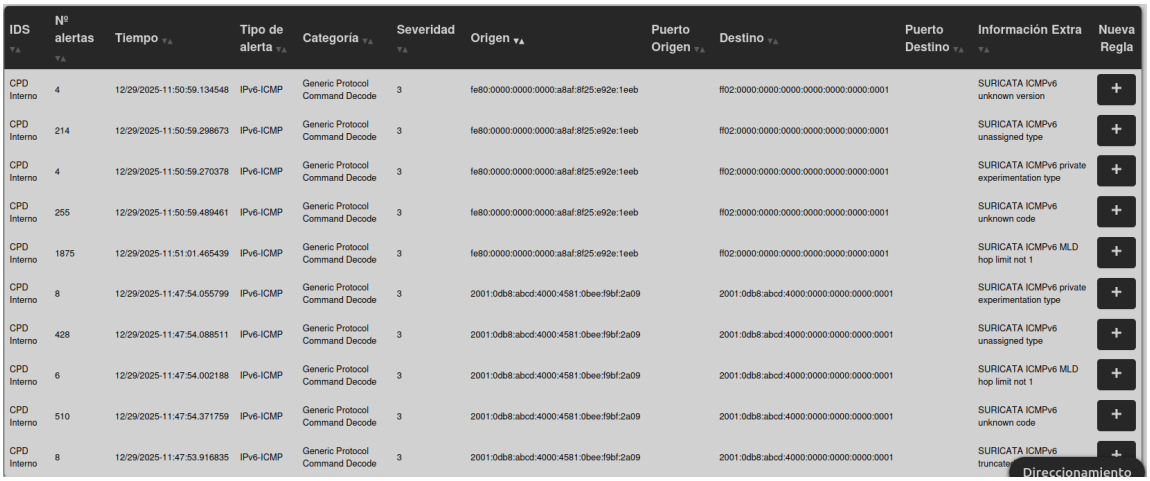
Figura 6.8: Captura de *Wireshark* del ataque *exploit6*

Para mitigar el ataque primero es necesario recordar que, como se ha explicado en el capítulo 3.1.5, `nftables` no realiza el reensamblado de paquetes a nivel de *hook ingress* al no permitir *conntrack*. Por lo tanto, en este sistema no es posible mitigar por completo la fragmentación superpuesta (*overlapping fragments*). Sin embargo, se puede limitar el número de mensajes de fragmentación mediante la instrucción:

```
nft add rule netdev dup_table6 block_ipv6_eth2 ip6 nexthdr ipv6-frag limit rate 50/second burst 100 packets counter accept
nft add rule netdev dup_table6 block_ipv6_eth2 ip6 nexthdr ipv6-frag counter drop
```

6.4. *Fuzz IPv6*

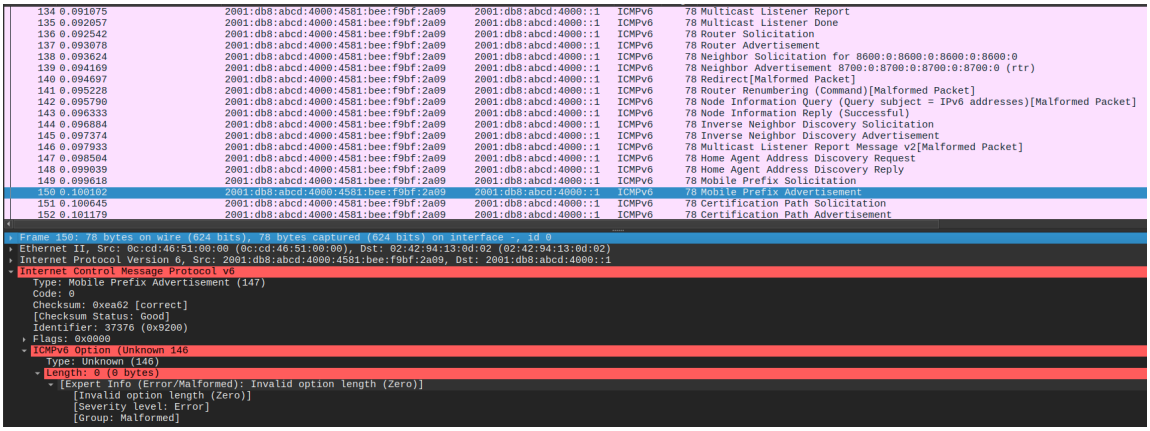
La herramienta `atk6-fuzz_ip6` genera paquetes ICMPv6 malformados con el objetivo de provocar comportamientos inesperados. Para comprobarlo, se han utilizado dos de las opciones incluidas en el comando: *ICMP6 echo request* y mensajes *Multicast*.



IDS	Nº alertas	Tiempo	Tipo de alerta	Categoría	Severidad	Origen	Puerto Origen	Destino	Puerto Destino	Información Extra	Nueva Regla
CPD Interno	4	12/29/2025-11:50:59.134548	IPv6-ICMP	Generic Protocol Command Decode	3	fe80:0000:0000:0000:a8af:8f25:e92e:1eeb		ff02:0000:0000:0000:0000:0000:0000:0001		SURICATA ICMPv6 unknown version	+
CPD Interno	214	12/29/2025-11:50:59.298673	IPv6-ICMP	Generic Protocol Command Decode	3	fe80:0000:0000:0000:a8af:8f25:e92e:1eeb		ff02:0000:0000:0000:0000:0000:0000:0001		SURICATA ICMPv6 unassigned type	+
CPD Interno	4	12/29/2025-11:50:59.270378	IPv6-ICMP	Generic Protocol Command Decode	3	fe80:0000:0000:0000:a8af:8f25:e92e:1eeb		ff02:0000:0000:0000:0000:0000:0000:0001		SURICATA ICMPv6 private experimentation type	+
CPD Interno	255	12/29/2025-11:50:59.489461	IPv6-ICMP	Generic Protocol Command Decode	3	fe80:0000:0000:0000:a8af:8f25:e92e:1eeb		ff02:0000:0000:0000:0000:0000:0000:0001		SURICATA ICMPv6 unknown code	+
CPD Interno	1875	12/29/2025-11:51:01.465439	IPv6-ICMP	Generic Protocol Command Decode	3	fe80:0000:0000:0000:a8af:8f25:e92e:1eeb		ff02:0000:0000:0000:0000:0000:0000:0001		SURICATA ICMPv6 MLD hop limit not 1	+
CPD Interno	8	12/29/2025-11:47:54.055799	IPv6-ICMP	Generic Protocol Command Decode	3	2001:0db8:abcd:4000:4581:0bee:f9bf:2a09		2001:0db8:abcd:4000:0000:0000:0000:0001		SURICATA ICMPv6 private experimentation type	+
CPD Interno	428	12/29/2025-11:47:54.088511	IPv6-ICMP	Generic Protocol Command Decode	3	2001:0db8:abcd:4000:4581:0bee:f9bf:2a09		2001:0db8:abcd:4000:0000:0000:0000:0001		SURICATA ICMPv6 unassigned type	+
CPD Interno	6	12/29/2025-11:47:54.002188	IPv6-ICMP	Generic Protocol Command Decode	3	2001:0db8:abcd:4000:4581:0bee:f9bf:2a09		2001:0db8:abcd:4000:0000:0000:0000:0001		SURICATA ICMPv6 MLD hop limit not 1	+
CPD Interno	510	12/29/2025-11:47:54.371759	IPv6-ICMP	Generic Protocol Command Decode	3	2001:0db8:abcd:4000:4581:0bee:f9bf:2a09		2001:0db8:abcd:4000:0000:0000:0000:0001		SURICATA ICMPv6 unknown code	+
CPD Interno	8	12/29/2025-11:47:53.916835	IPv6-ICMP	Generic Protocol Command Decode	3	2001:0db8:abcd:4000:4581:0bee:f9bf:2a09		2001:0db8:abcd:4000:0000:0000:0000:0001		SURICATA ICMPv6 truncated	+

Direccionamiento

Figura 6.9: Captura de alertas de los ataques *ICMP6 echo request* y *Multicast*



```
  134 0.091075  2001:db8:abcd:4000:4581:bee:f9bf:2a09  2001:db8:abcd:4000::1  ICMPv6  78 Multicast Listener Report
  135 0.092057  2001:db8:abcd:4000:4581:bee:f9bf:2a09  2001:db8:abcd:4000::1  ICMPv6  78 Multicast Listener Done
  136 0.092542  2001:db8:abcd:4000:4581:bee:f9bf:2a09  2001:db8:abcd:4000::1  ICMPv6  78 Router Solicitation
  137 0.093078  2001:db8:abcd:4000:4581:bee:f9bf:2a09  2001:db8:abcd:4000::1  ICMPv6  78 Router Advertisement
  138 0.093624  2001:db8:abcd:4000:4581:bee:f9bf:2a09  2001:db8:abcd:4000::1  ICMPv6  78 Neighbor Solicitation for 8600:0:8600:0:8600:0:8600:0
  139 0.094169  2001:db8:abcd:4000:4581:bee:f9bf:2a09  2001:db8:abcd:4000::1  ICMPv6  78 Neighbor Advertisement 8700:0:8700:0:8700:0:8700:0 (rtr)
  140 0.094697  2001:db8:abcd:4000:4581:bee:f9bf:2a09  2001:db8:abcd:4000::1  ICMPv6  78 Redirect[Malformed Packet]
  141 0.095228  2001:db8:abcd:4000:4581:bee:f9bf:2a09  2001:db8:abcd:4000::1  ICMPv6  78 Router Renumbering (Command)[Malformed Packet]
  142 0.095790  2001:db8:abcd:4000:4581:bee:f9bf:2a09  2001:db8:abcd:4000::1  ICMPv6  78 Node Information Query (Query subject = IPv6 addresses)[Malformed Packet]
  143 0.096333  2001:db8:abcd:4000:4581:bee:f9bf:2a09  2001:db8:abcd:4000::1  ICMPv6  78 Node Information Reply (Successful)
  144 0.096884  2001:db8:abcd:4000:4581:bee:f9bf:2a09  2001:db8:abcd:4000::1  ICMPv6  78 Inverse Neighbor Discovery Solicitation
  145 0.097374  2001:db8:abcd:4000:4581:bee:f9bf:2a09  2001:db8:abcd:4000::1  ICMPv6  78 Inverse Neighbor Discovery Advertisement
  146 0.097933  2001:db8:abcd:4000:4581:bee:f9bf:2a09  2001:db8:abcd:4000::1  ICMPv6  78 Multicast Listener Report Message v2[Malformed Packet]
  147 0.098504  2001:db8:abcd:4000:4581:bee:f9bf:2a09  2001:db8:abcd:4000::1  ICMPv6  78 Home Agent Address Discovery Request
  148 0.099039  2001:db8:abcd:4000:4581:bee:f9bf:2a09  2001:db8:abcd:4000::1  ICMPv6  78 Home Agent Address Discovery Reply
  149 0.099618  2001:db8:abcd:4000:4581:bee:f9bf:2a09  2001:db8:abcd:4000::1  ICMPv6  78 Mobile Prefix Solicitation
  150 0.100102  2001:db8:abcd:4000:4581:bee:f9bf:2a09  2001:db8:abcd:4000::1  ICMPv6  78 Mobile Prefix Advertisement
  151 0.100645  2001:db8:abcd:4000:4581:bee:f9bf:2a09  2001:db8:abcd:4000::1  ICMPv6  78 Certification Path Solicitation
  152 0.101179  2001:db8:abcd:4000:4581:bee:f9bf:2a09  2001:db8:abcd:4000::1  ICMPv6  78 Certification Path Advertisement
▸ Frame 150: 78 bytes on wire (624 bits), 78 bytes captured (624 bits) on interface -, id 0
▸ Ethernet II, Src: 0c:cd:46:51:00:00 (0c:cd:46:51:00:00), Dst: 02:42:94:13:0d:02 (02:42:94:13:0d:02)
▸ Internet Protocol Version 6, Src: 2001:db8:abcd:4000:4581:bee:f9bf:2a09, Dst: 2001:db8:abcd:4000::1
▾ Internet Control Message Protocol v6
    Type: Mobile Prefix Advertisement (147)
    Code: 0
    Checksum: 0xea62 [correct]
    [Checksum Status: Good]
    Identifier: 37376 (0x9200)
  ▸ Flags: 0x0000
  ▾ ICMPv6 Option (Unknown 146
      Type: Unknown (146)
    ▾ Length: 0 (0 bytes)
      ▾ [Expert Info (Error/Malformed): Invalid option length (Zero)]
          [Invalid option length (Zero)]
          [Severity level: Error]
          [Group: Malformed]
```

Figura 6.10: Captura de *Wireshark* del ataque *ICMP6 echo request*

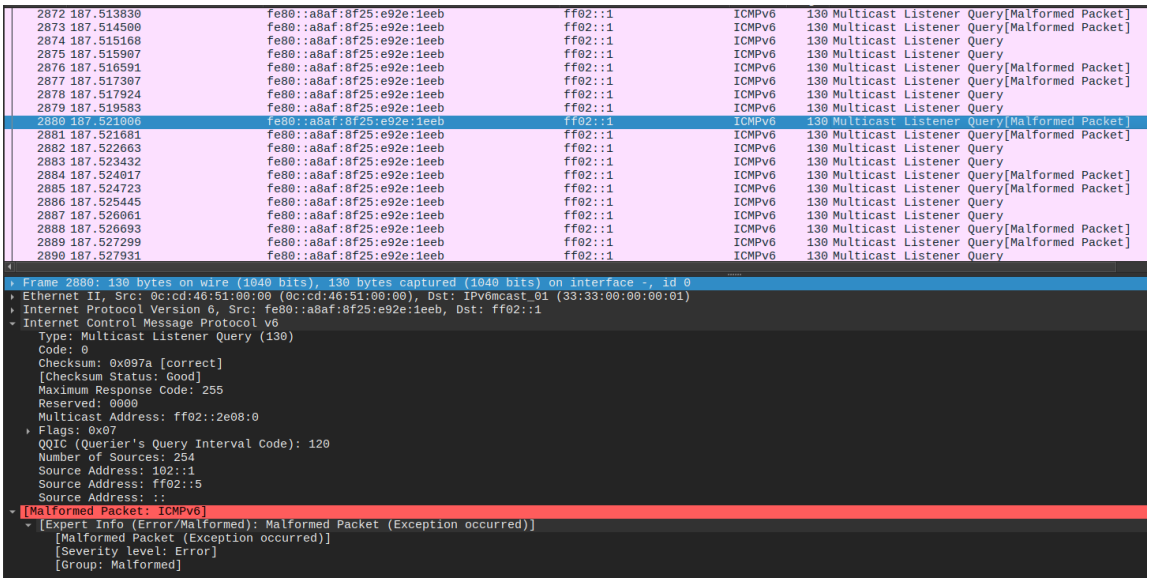


Figura 6.11: Captura de Wiresahrk del ataque *Multicast*

Observando el tráfico generado, se puede sacar como conclusión que la mitigación para estas dos opciones se pueden realizar mediante las siguientes instrucciones:

```
{
nft add rule netdev dup_table6 block_ipv6_eth2 ip6 nexthdr icmpv6
    icmpv6 type 128 limit rate 10/second burst 20 packets counter
    accept
nft add rule netdev dup_table6 block_ipv6_eth2 ip6 nexthdr icmpv6
    icmpv6 type 128 counter drop
nft add rule netdev dup_table6 block_ipv6_eth2 ip6 nexthdr icmpv6
    icmpv6 type 200-255 counter drop
nft add rule netdev dup_table6 block_ipv6_eth2 ip6 nexthdr icmpv6
    icmpv6 type {130,131,132} ip6 hoplimit != 1 counter drop
```

Gracias a estas instrucciones se realizan las siguientes acciones:

- Se limita ICMPv6 128 (Echo Request), permitiendo 10 pings por segundo con un pico máximo de 20 paquetes.
- Se bloquean ICMPv6 tipos 200 a 255, correspondientes a experimentales y reservados.
- Se bloquean ICMPv6 tipo 130 (*Multicast Listener Query*), 131 (*Multicast Listener Report*) y 132 (*Multicast Listener Done*) con *hop-limit* distinto de 1.

MLD (*Multicast Listener Discovery*) se utiliza para la gestión de multicast y, gracias al RFC 3810 [62], se sabe que estos paquetes deben tener el campo *hoplimit* a 1 y, en caso contrario, se deben eliminar. Debido a que estas acciones no son suficientes para bloquear los ataques en su totalidad, se propone realizar un bloqueo temporal (ejemplo en el capítulo 6.7) a la IP que está realizando los ataques.

6.5. *DoS new ip6*

Uno de los casos más curiosos al realizar pruebas es el `atk6-dos-new-ip6 -h eth0`. Este ataque bloquea la autoconfiguración IPv6 (*Stateless Address Autoconfiguration*, SLAAC) mediante el envío de paquetes ICMPv6 *Neighbor Advertisement (NA)* con la *flag* de *Override* activada, como se aprecia en la figura 6.12. Esto indica que el ataque está sobrescribiendo las entradas existentes de la caché de vecinos de los hosts. Un claro indicador de que se está realizando un ataque *spoofing* es que el mensaje NA se está realizando desde una dirección no específica (::), un comportamiento que no es válido en redes IPv6. A pesar de la gravedad del impacto, Suricata no detecta el ataque e impide a los puestos de trabajo obtener una dirección IPv6.

```
root@kali:~# atk6-dos-new-ip6 eth0
Started ICMP6 DAD Denial-of-Service (Press Control-C to end) ...
Spoofed packet for existing ip6 as ::
Spoofed packet for existing ip6 as fe80::42:f9ff:fee0:c00
Spoofed packet for existing ip6 as ::
Spoofed packet for existing ip6 as fe80::42:f9ff:fee0:c00
Spoofed packet for existing ip6 as ::
Spoofed packet for existing ip6 as ::
Spoofed packet for existing ip6 as fe80::42:d9ff:fec3:7602
Spoofed packet for existing ip6 as fe80::42:f9ff:fee0:c00
Spoofed packet for existing ip6 as 2001:db8:abcd:4000:1:2ff:fe00:0
Spoofed packet for existing ip6 as ::
Spoofed packet for existing ip6 as fe80::42:f9ff:fee0:c00
Spoofed packet for existing ip6 as ::
Spoofed packet for existing ip6 as 2001:db8:abcd:4000:1:2ff:fe00:0
Spoofed packet for existing ip6 as ::
Spoofed packet for existing ip6 as 2001:db8:abcd:4000:1:2ff:fe00:0
```

Figura 6.12: Instrucción de ataque DoS mediante *DoS new ip6*

fe80::42:94ff:fe13:d02	ff02::1	…	110	Router Advertisement from 02:42:94:13:0d:02
fe80::7903:fa10:d853:b6d5	ff02::16	…	110	Multicast Listener Report Message v2
fe80::7903:fa10:d853:b6d5	ff02::16	…	110	Multicast Listener Report Message v2
::	ff02::16	…	90	Multicast Listener Report Message v2
::	ff02::16	…	90	Multicast Listener Report Message v2
::	ff02::1:ffe0:c00	…	86	Neighbor Solicitation for fe80::42:f9ff:fee0:c00
fe80::42:f9ff:fee0:c00	ff02::1	…	86	Neighbor Advertisement fe80::42:f9ff:fee0:c00 (ovr) is at 0c:cd:26:f2:85:7a
fe80::42:f9ff:fee0:c00	ff02::1	…	86	Neighbor Advertisement fe80::42:f9ff:fee0:c00 (ovr) is at 0c:cd:26:f2:85:7a
fe80::42:d9ff:fec3:7601	ff02::2	…	70	Router Solicitation from 02:42:d9:c3:76:01
fe80::7903:fa10:d853:b6d5	ff02::1:ff00:0	…	86	Neighbor Solicitation for :: from 0c:cd:46:51:00:00
::	ff02::1	…	86	Neighbor Advertisement :: (ovr) is at 0c:cd:94:c6:af:d8
::	ff02::1	…	86	Neighbor Advertisement :: (ovr) is at 0c:cd:94:c6:af:d8
fe80::42:94ff:fe13:d02	ff02::1	…	110	Router Advertisement from 02:42:94:13:0d:02
::	ff02::16	…	90	Multicast Listener Report Message v2
fe80::7903:fa10:d853:b6d5	ff02::16	…	110	Multicast Listener Report Message v2
::	ff02::16	…	90	Multicast Listener Report Message v2
::	ff02::1:ff00:0	…	86	Neighbor Solicitation for 2001:db8:abcd:4000:1:2ff:fe00:0
2001:db8:abcd:4000:1:2ff:fe00:0	ff02::1	…	86	Neighbor Advertisement 2001:db8:abcd:4000:1:2ff:fe00:0 (ovr) is at 0c:cd:e0:db:12:b4
2001:db8:abcd:4000:1:2ff:fe00:0	ff02::1	…	86	Neighbor Advertisement 2001:db8:abcd:4000:1:2ff:fe00:0 (ovr) is at 0c:cd:e0:db:12:b4
::	ff02::16	…	90	Multicast Listener Report Message v2
::	ff02::16	…	90	Multicast Listener Report Message v2
fe80::7903:fa10:d853:b6d5	ff02::16	…	110	Multicast Listener Report Message v2

Figura 6.13: Captura de *Wireshark* de ataque mediante *DoS new ip6*

Este ataque no se puede impedir desde el Router CPD ya que el switch es el que redirige el tráfico al estar en la misma red. En cambio, gracias a las reglas *OpenFlow* explicadas en el capítulo 3.1.6, este ataque se puede mitigar realizando un *bridge* entre las interfaces que conectan los puertos de trabajo, en este caso *eth0* y *eth1* según el diagrama 4.3,y creando el flujo que impida el ataque

mediante la instrucción:

```
ovs-ofctl -O OpenFlow13 add-flow br0 "priority=100,ipv6,icmpv6_type
    =134,ipv6_src=0:0:0:0:0:0:0:0,actions=drop"
```

6.6. *Dump router6*

Para extraer información de routers IPv6 en una red local se puede utilizar la instrucción atk6-dump_router6 como se indica en la figura 6.14. Gracias a la escucha y captura de mensajes *IPv6 Router Advertisement* (RA) se puede identificar los routers activos. Esto no es un ataque, sino un escaneo de la red. Aún así, la información obtenida se puede usar para realizar un ataque o detectar routers no autorizados.

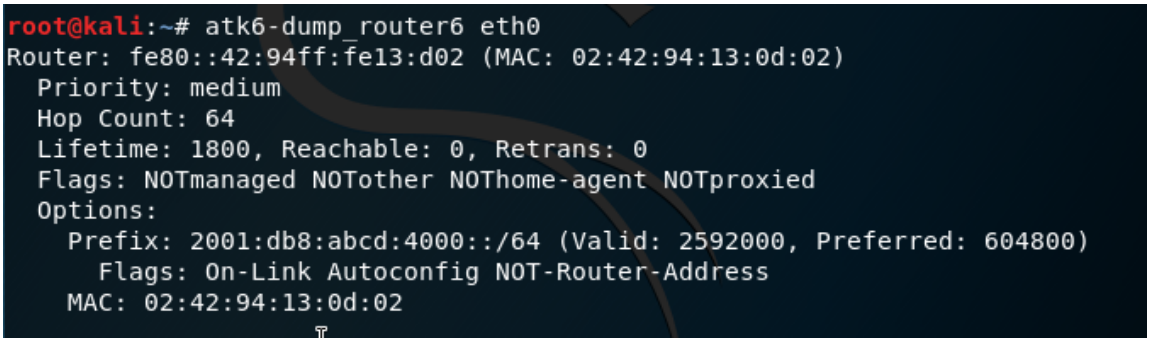


Figura 6.14: Ejecución de instrucción *dump router6*

Y, como se puede comprobar, en la figura 6.15, la MAC que se devuelve es la de la interfaz a la que está conectada el *switch* correspondiente, en este caso, eth2.

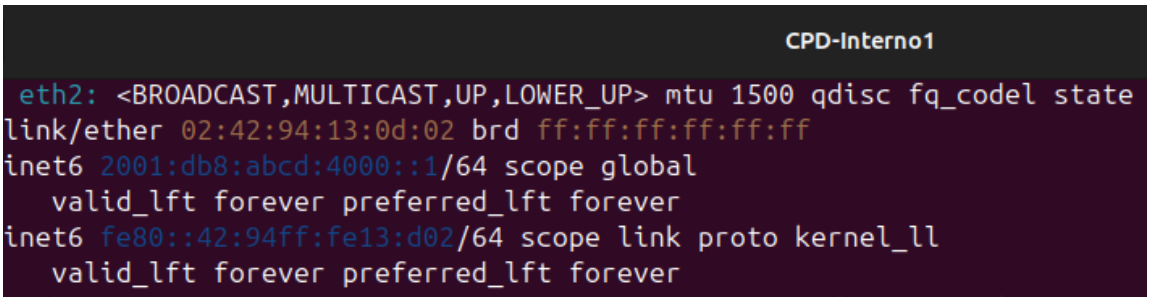


Figura 6.15: Demostración de que la MAC obtenida es la correcta

6.7. Bloqueo temporal

En el caso en el que los bloqueos anteriores no funcionen para un nuevo ataque o por el hecho de que un usuario esté realizando actividades sospechosas, se puede realizar un bloqueo temporal. Esto se realiza añadiendo reglas en la tabla cuyo *chain* contenga `_dynamic`. Un ejemplo de ello se puede observar en la figura 6.16, donde se bloquea todo tipo de comunicaciones a la máquina Kali interna.

```
block_ipv6_eth2_dynamic policy: accept
    ip6 saddr 2001:db8:abcd:4000:8f0a:ac28:1f59:40be counter packets 136109 bytes 63153400 drop
```

Figura 6.16: Bloqueo temporal a IP

Como también se explica en 5.2.5, se dan diez segundos desde que el ataque se detiene, hasta que la restricción hacia la IP desaparace de forma automática. Esto se puede observar en la figura 6.17.

```
root@CPD:/scripts# python3 counter.py
Monitor de reglas dinámicas (timeout: 10s)
----------------------------------------
+ Regla 10 creada
  → Regla 10: 38051 paquetes
  → Regla 10: 40948 paquetes
  → Regla 10: 43827 paquetes
  → Regla 10: 46528 paquetes
  → Regla 10: 49341 paquetes
  → Regla 10: 51842 paquetes
- Regla 10 eliminada (inactiva 10s)
```

Figura 6.17: Monitorización desde el CPD

6.8. Ataque Externo

Todos los ataques vistos hasta el momento han sido replicados desde el Kali-Interno, pero también se pueden realizar desde el exterior. En la figura 6.18 se observa un ejemplo con el ataque *atk6-denial* desde la máquina de Kali-Externa, por lo que la alerta la captura el IDS-CPD-Externo.

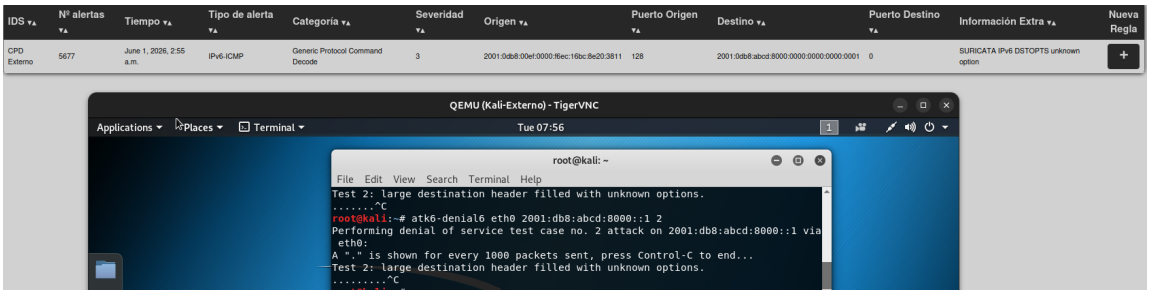


Figura 6.18: Captura de alertas por parte del IDS CPD-Externo

Capítulo 7

Comparativa con un entorno real: Fortinet

Una de las principales prioridades que se le ha dado a este trabajo es realizarlo de la manera más realista posible. Para demostrarlo, se realiza la comparación del sistema de mitigación creado con uno de los *firewalls* más utilizados a nivel mundial en el mundo empresarial: Fortinet. Suricata no compite con este tipo de proveedores, pero sí se alinea conceptualmente con lo que hacen a nivel funcional. Además de las configuraciones mostradas a continuación a través de las interfaces web que proporciona Fortinet, también se puede obtener la configuración y modificarla a partir de la línea de comandos, con una sintaxis muy similar a la utilizada en este proyecto con FRR.

Conectividad

Una vez que se han desplegado los FortiGates necesarios, lo primero que se debe hacer es dotar a los nodos de conectividad. Como se observa en la figura 7.1, se poseen distintas áreas dentro de la interfaz para configurar interfaces, rutas estáticas, políticas de rutas, OSPF, BGP, etc. También tiene un apartado de diagnóstico en la cual se puede elegir la interfaz desde donde realizar el filtrado de tráfico, eligiendo criterios como origen del tráfico, puerto, número de protocolo o cantidad de paquetes capturados, entre otros. Esta funcionalidad resulta equivalente a la acción de abrir *Wireshark* desde GNS3 y aplicar filtros en la barra de búsqueda.

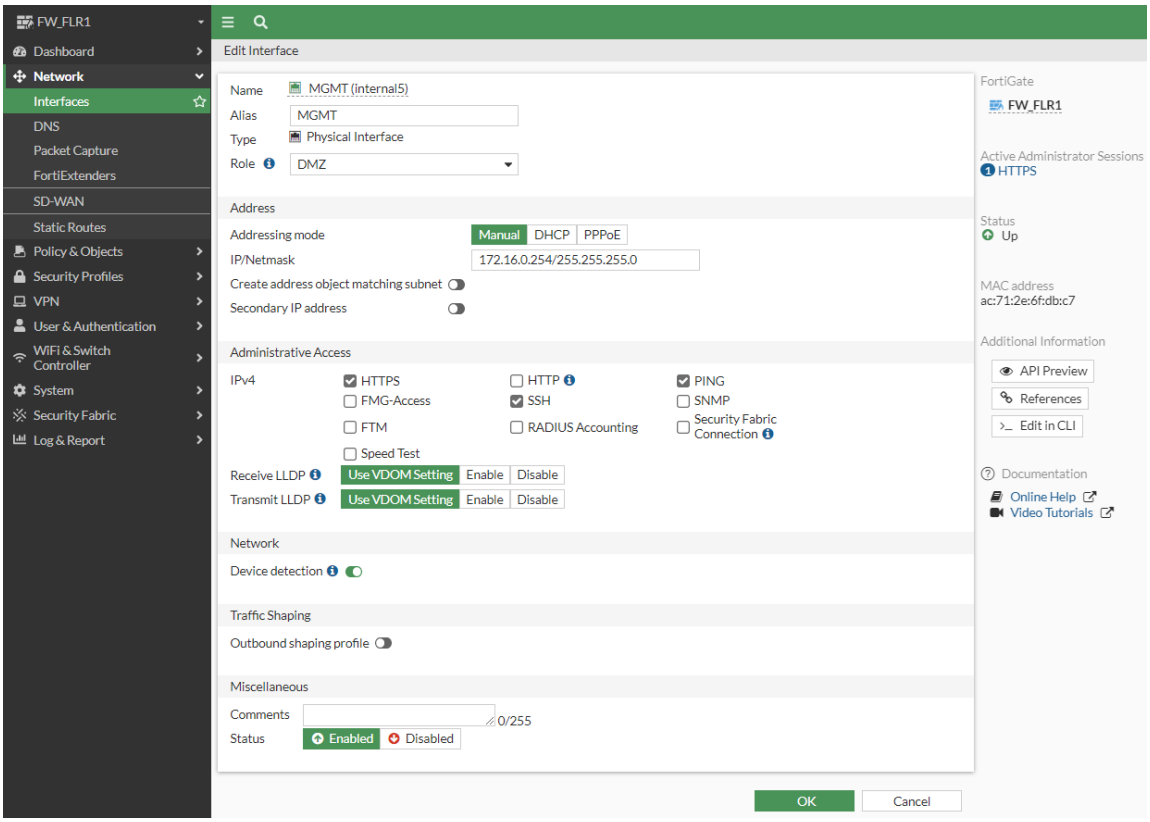


Figura 7.1: Configuración de Interfaces en FortiGate [63]

La similitud entre la configuración de BGP utilizada en Fortinet y los parámetros asignados a través de FRR se muestra en las figuras 7.2 y 7.3. En ellas se observa la posibilidad de definir opciones como el *router-id*, habilitar o desactivar vecinos BGP o las redes a anunciar. También se ofrecen diferentes opciones para la configuración de *timers* o la opción de activar distintos parámetros para la elección del `path` deseado.

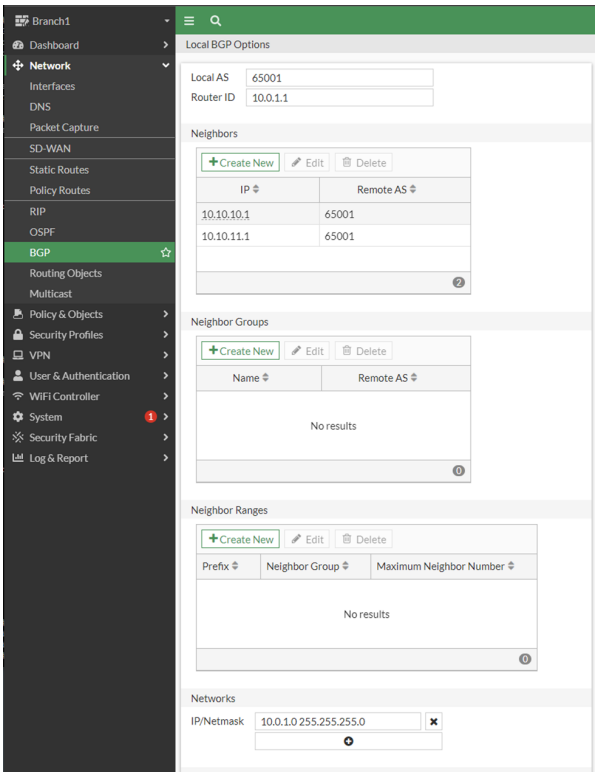


Figura 7.2: Configuración de BGP en FortiGate: configuración de vecinos y rutas a anunciar [64]

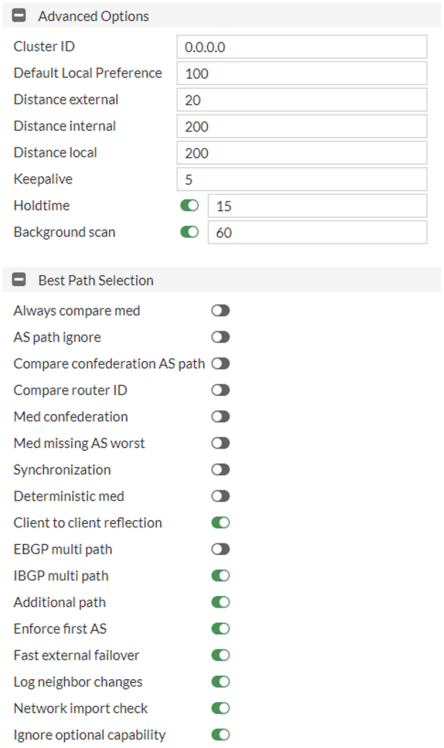


Figura 7.3: Configuración de BGP en FortiGate:*timers* y mejor *path* [64]

Creación de políticas

Al igual que en este trabajo, la interfaz de FortiGate tiene una pestaña para la generación de reglas, mostrada en la figura 7.4. Desde aquí se pueden configurar ciertos parámetros como direcciones y puertos origen/destino, la acción a realizar (*accept/deny*), así como otros atributos adicionales relacionados con el protocolo y el tratamiento del tráfico.

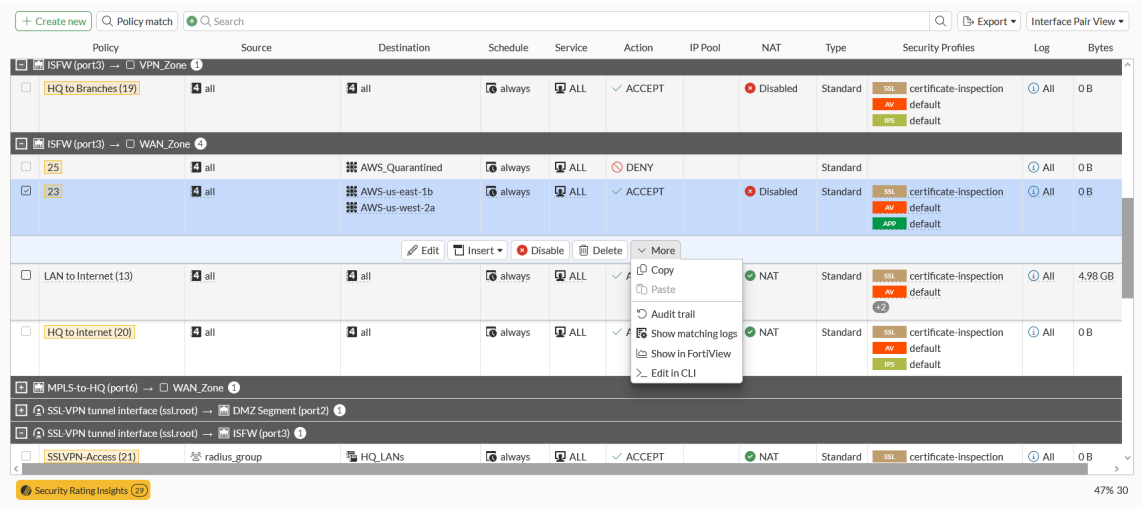


Figura 7.4: Ejemplo de políticas en Fortinet [65]

Prevención de Intrusos

Para crear un perfil de prevención de intrusos, Fortinet tiene un apartado en la sección “Perfiles de seguridad” llamado “Prevención de Intrusos”. De esta forma, se puede realizar la monitorización, el bloqueo o inhabilitar cierto tráfico. Dentro de las firmas de un sistema empresarial real, como son las comunicaciones de MAPFRE, donde he tenido la oportunidad de realizar las prácticas curriculares, se tienen firmas sobre IPv6. Entre ellas se encuentra la firma de Snort, sistema de prevención sobre el que está basado Suricata. Además, Fortinet tiene una herramienta [66] para realizar la traducción de reglas Snort al formato FortiOS.

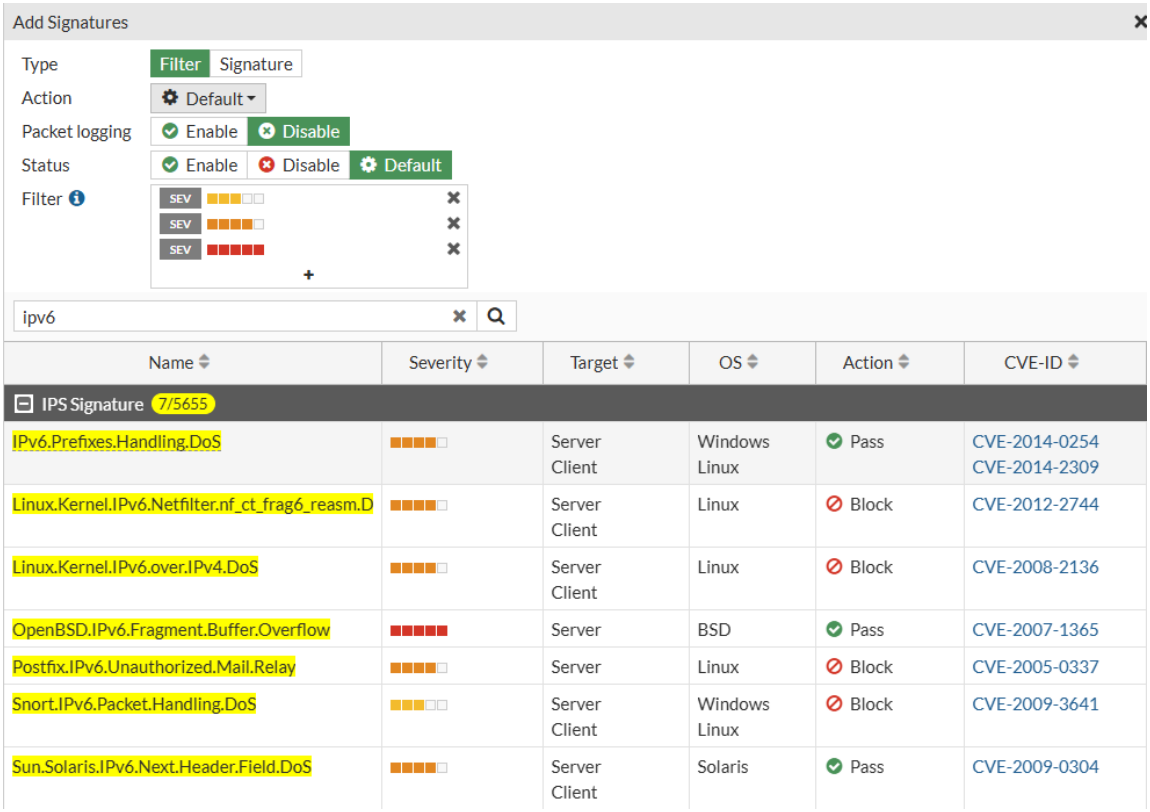


Figura 7.5: Firmas de IPv6 en FortiGate

Comprobación de la instalación

Al igual que en el formulario del dashboard con las reglas *nftables*, FortiGate también permite realizar un visualizado previo de las políticas o parámetros de configuración antes de realizar la instalación. Ejemplo de ello es la figura 7.6.

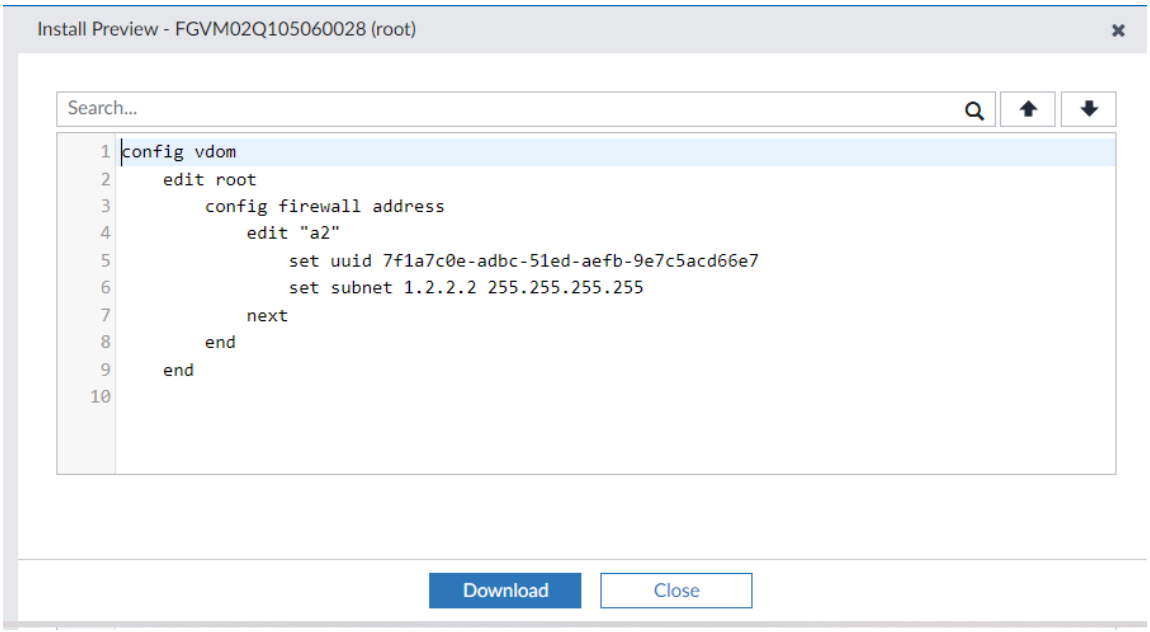


Figura 7.6: Ejemplo de *install preview* en FortiGate [67]

Capítulo 8

Conclusiones y trabajos futuros

Con este trabajo se ha querido realizar una prueba de concepto que demuestra la importancia de la seguridad en IPv6. Se han utilizado diversas tecnologías que han permitido la creación de un entorno similar al de la UCM y que permite probar ataques IPv6 sin causar problemas en la red. Uno de los principales problemas del CPD es la dificultad del despliegue de IPv6 en la UCM. En este trabajo se intenta lidiar con esa dificultad proponiendo una nueva arquitectura de red que utilice tanto OSPF como BGP para las comunicaciones con REDIMadrid y de forma interna.

Mediante las reglas nftables y la interfaz web, se ha propuesto un sistema efectivo de mitigación que obtiene en tiempo real los ataques que se están realizando y da las herramientas al usuario para hacer frente a ellos. Durante el proyecto, se han dado ejemplos de ataques mediante capturas de *Wireshark* y se han proporcionado opciones para mitigar los ataques. También se ha observado que no todos los ataques se pueden capturar mediante Suricata, sacando por tanto como conclusión que este sistema de detección frente a intrusos no es suficiente para un entorno real. Otros de los problemas a los que se enfrenta el CPD es el bloqueo de IP de forma temporal, ejemplificado en este trabajo. Lo que se ha realizado en este proyecto es la automatización parcial de esta tarea, permitiendo mantener bloqueado al usuario si sigue intentando realizar un ataque y desbloqueando el puesto de trabajo pasado un tiempo.

En los entornos reales se realizan la creación de nodos auxiliares como maquetas para la realización de pruebas de concepto antes de realizar cambios en la red. Debido al coste que supone esto, en este trabajo se incentiva el uso de GNS3 para realizar este tipo de pruebas de concepto. En varios casos, los ataques IPv6 eran lo suficientemente potentes para corromper la arquitectura o dejar sin conexión a algún nodo. GNS3 permite que, con un simple reinicio del nodo afectado, se puedan seguir probando distintas soluciones al problema.

Como conclusión final, se ha visto la posibilidad de incorporar Suricata como IDS principal y se ha demostrado, con los vectores de ataque proporcionados por *thc-ipv6*, que esta herramienta no es suficiente por sí sola para mantener una red segura de ataques. Pese a ello, se ha dado una solución usando *nftables* y *ovs-vsctl*, cuya complejidad a la hora de definir reglas y flujos demuestran la complejidad que existe para gestionar una red.

Como posible trabajo futuro se plantea realizar la automatización de la configuración de los switches mediante *ovs-vsctl* y la sustitución de los IDS de Suricata por nodos Fortinet en el propio GNS3, para lo cual se necesitaría la licencia de la compañía [68]. [1]

[1] *Declaración sobre el uso de IA: En el curso del desarrollo de este trabajo, el autor utilizó Copilot y ChatGPT para la implementación del dashboard. Tras el uso de estas herramientas, el autor revisó y modificó cuidadosamente el contenido y asume la responsabilidad total de los contenidos de la publicación.*

Capítulo 9

Conclusions and future work

This work aims to present a proof of concept that demonstrates the importance of security in IPv6. A range of technologies have been employed to create an environment comparable to that of UCM, enabling IPv6 attacks to be tested without causing disruptions to the production network. One of the main challenges faced by the data centre (CPD) is the difficulty of deploying IPv6 at UCM. This work seeks to address this issue by proposing a new network architecture that makes use of both OSPF and BGP for communications with REDIMadrid, as well as for internal communications.

By means of nftables rules and a web interface, an effective mitigation system has been proposed that is capable of detecting ongoing attacks in real time and providing users with the necessary tools to respond to them. Throughout the project, examples of attacks have been presented through Wireshark captures, and several mitigation options have been provided. It has also been observed that not all attacks can be detected by Suricata; therefore, it can be concluded that this intrusion detection system is not sufficient for a real-world environment. Another issue faced by the CPD is the temporary blocking of IP addresses, which has been illustrated in this work. The approach taken in this project involves the partial automation of this task, allowing an attacker to remain blocked if they continue attempting to carry out an attack, while automatically unblocking the workstation after a certain period of time.

In real-world environments, auxiliary nodes are established as mock-ups to conduct proofs of concept (PoC) before implementing changes in production networks. Due to the significant costs involved, this study aims to encourage the use of GNS3 for such testing. In several instances, the IPv6 attacks were potent enough to corrupt the architecture or cause connectivity loss in certain nodes. GNS3 facilitates continuity in testing various solutions by allowing a simple reboot of the affected node.

In final conclusion, the feasibility of incorporating Suricata as the primary Intrusion Detection System (IDS) was evaluated. It was demonstrated, through attack vectors provided by *thc-ipv6*, that this tool alone is insufficient to maintain a secure network. Nonetheless, a solution was provided using nftables and ovs-vsctl, whose complexity in defining rules and flows highlights the inherent difficulty of network management.

Regarding future work, the automation of switch configuration via *ovs-vsctl* is proposed, alongside the replacement of Suricata IDS nodes with Fortinet nodes within GNS3, which would require the corresponding corporate license [68]. [1]

[1] *Statement on the use of AI: During the development of this work, the author used Copilot and ChatGPT for the implementation of the dashboard. Following the use of these tools, the author carefully reviewed and modified the content and assumes full responsibility for the contents of this publication.*

Apéndice A

Creación del contenedor

A continuación, se explican los pasos seguidos para la creación de un nuevo contenedor Docker FRR que contenga las herramientas necesarias. Este material ha sido proporcionado por el profesor de este trabajo, Juan Carlos Fabero.

```
sudo debootstrap --variant=minbase --include=frr,iputils-ping,
   traceroute,vim-tiny,nano-tiny,iproute2,nftables,isc-dhcp-server,
   procps,ssh trixie /mnt http://ftp.debian.org/debian
```

Crear un archivo `zebra.sh` que se encargue de iniciar frr y copiarlo en `/usr/local/bin` dentro de la imagen. No olvidar darle permisos de ejecución.

```
#!/bin/bash -f
/usr/lib/frr/watchfrr zebra staticd &
/usr/sbin/sysctl -f
bash
```

Modificar el archivo `/etc/frr/daemons` para activar los demonios apropiados.

Crear `/var/run/frr` y hacer que pertenezca a `frr:frr` (es mejor hacerlo desde `chroot`).

Añadir a `/etc/sysctl.conf` las dos líneas siguientes para corregir el problema de frr con el encaminamiento en IPv6:

```
net.ipv6.conf.all.forwarding=1
net.ipv6.conf.default.forwarding=1
```

Crear el archivo de `dhcpd[6].leases`:

```
touch /var/lib/dhcp/dhcpd.leases
touch /var/lib/dhcp/dhcpd6.leases
```

Hacer `chroot /mnt` y ejecutar `apt-get clean` para eliminar los paquetes descargados. Borrar también los archivos en `/mnt/var/lib/apt/lists`. También se pueden eliminar páginas de manual y documentación adicional para ahorrar espacio.

Con la imagen montada, comprimir todos los archivos a un `.tar`, excluyendo el directorio `/proc` y algunos más contenidos en el archivo `exclude`:

```
proc/*
usr/lib/locale/*
usr/share/doc/*
usr/share/locale/*
usr/share/man/*
var/cache/apt/*
var/lib/apt/*
```

```
var/lib/dpkg/*
tmp/*

cd /mnt
tar cvf /tmp/debian.tar --exclude-from=path-to-exclude *
chown juan:juan /tmp/debian.tar
```

Crear la imagen del contenedor a partir del archivo `.tar`:

```
docker import /tmp/debian.tar jcfabero/debian-frr:tag
```

Con `tag` la fecha en formato ISO: `220113`, p.e.
Crear un alias con la etiqueta `latest`:

```
docker image tag jcfabero/debian-frr:22013 jcfabero/debian-frr:
    latest
```

Crear una nueva appliance en GNS3 a partir de la imagen Docker. Elegir consola de tipo `telnet` y marcar la casilla de inicio automático. Definir como comando de inicio `zebra.sh`.
Incluir los directorios `/etc/frr` y `/etc/dhcp/` en la lista de persistentes.

Bibliografía

[1] Wikipedia. IPv6 — wikipedia, la enciclopedia libre, 2025. [Internet; descargado 19-diciembre-2025]. URL: https://es.wikipedia.org/w/index.php?title=IPv6&oldid=171065433.

[2] IPv6 adoption statistics. https://www.google.com/intl/en/ipv6/statistics.html, 2025. Consultado en 2025.

[3] Sulyab Thottungal Valapu and John Heidemann. Towards a non-binary view of ipv6 adoption. In *Proceedings of the 2025 ACM Internet Measurement Conference*, page 727–745. ACM, October 2025. URL: http://dx.doi.org/10.1145/3730567.3764467, doi:10.1145/3730567.3764467.

[4] Docker, Inc. Docker. https://www.docker.com/, 2026. Accedido: 2026-01-02.

[5] QEMU Project. Qemu: Generic and open source machine emulator and virtualizer, 2025. URL: https://www.qemu.org/.

[6] GNS3 Technologies Inc. Gns3 - network software emulator, 2025. Consultado el 18 de diciembre de 2025. URL: https://www.gns3.com/.

[7] Dynamips.io, LLC. Dynamips — build real networking skills with labs, images, and workbooks, 2025. Consultado el 18 de diciembre de 2025. URL: https://dynamips.io/.

[8] Jiawei Shen, Jia Zhu, Hanghui Guo, Weijie Shi, Guoqing Ma, Yidan Liang, Jingjiang Liu, Hao Chen, and Shimin Di. Dynagen: Unifying temporal knowledge graph reasoning with dynamic subgraphs and generative regularization, 2025. URL: https://arxiv.org/abs/2512.12669, arXiv:2512.12669.

[9] OISF — Open Information Security Foundation. Suricata — high performance network ids, ips, and network security monitoring engine, 2025. Consultado el 21 de diciembre de 2025. URL: https://suricata.io/.

[10] Open Information Security Foundation. Suricata rules documentation, 2025. Consultado el 18 de diciembre de 2025. URL: https://docs.suricata.io/en/latest/rules/intro.html.

[11] Open Information Security Foundation. Suricata user guide — configuration, 2025. Consultado el 18 de diciembre de 2025. URL: https://docs.suricata.io/en/latest/configuration/index.html.

[12] Open Information Security Foundation. Suricata user guide — output, 2025. Consultado el 18 de diciembre de 2025. URL: https://docs.suricata.io/en/latest/output/index.html.

[13] Marco Chiesa, Roshan Sedar, Gianni Antichi, Michael Borokhovich, Andrzej Kamisiński, Georgios Nikolaidis, and Stefan Schmid. Fast reroute on programmable switches. *IEEE/ACM Transactions on Networking*, 29(2):637–650, 2021. doi:10.1109/TNET.2020.3045293.

[14] Paul Jakma and David Lamparter. Introduction to the quagga routing suite. *Network, IEEE*, 28:42–48, 03 2014. doi:10.1109/MNET.2014.6786612.

[15] John H. Hartman and John K. Ousterhout. The zebra striped network file system. *ACM Trans. Comput. Syst.*, 13(3):274–310, August 1995. doi:10.1145/210126.210131.

[16] FRRouting Project. vtysh — integrated cli shell for frrouting, 2025. Consultado el 21 de diciembre de 2025. URL: https://docs.frrouting.org/en/latest/vtysh.html.

[17] Netfilter Project. nftables — packet classification framework and firewall tool, 2025. Consultado el 21 de diciembre de 2025. URL: https://wiki.nftables.org/.

[18] Open vSwitch Project. Open vswitch — production quality, multilayer virtual switch, 2025. Consultado el 21 de diciembre de 2025. URL: https://www.openvswitch.org/.

[19] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, March 2008. doi:10.1145/1355734.1355746.

[20] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan J. Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Jonathan Stringer, Pravin Shelar, Keith Amidon, and Martín Casado. The design and implementation of open vswitch. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, NSDI'15, page 117–130, USA, 2015. USENIX Association.

[21] Fortinet Inc. Fortinet — cybersecurity solutions and network security, 2025. Consultado el 21 de diciembre de 2025. URL: https://www.fortinet.com/.

[22] Dr. Steve E. Deering and Bob Hinden. IP Version 6 Addressing Architecture. RFC 4291, February 2006. URL: https://www.rfc-editor.org/info/rfc4291, doi:10.17487/RFC4291.

[23] William A. Simpson, Dr. Thomas Narten, Erik Nordmark, and Hesham Soliman. Neighbor Discovery for IP version 6 (IPv6). RFC 4861, September 2007. URL: https://www.rfc-editor.org/info/rfc4861, doi:10.17487/RFC4861.

[24] Brian Haberman and Bob Hinden. Unique Local IPv6 Unicast Addresses. RFC 4193, October 2005. URL: https://www.rfc-editor.org/info/rfc4193, doi:10.17487/RFC4193.

[25] Brian Haberman and Dave Thaler. Unicast-Prefix-based IPv6 Multicast Addresses. RFC 3306, September 2002. URL: https://www.rfc-editor.org/info/rfc3306, doi:10.17487/RFC3306.

[26] John A. Hawkinson and Tony J. Bates. Guidelines for creation, selection, and registration of an Autonomous System (AS). RFC 1930, March 1996. URL: https://www.rfc-editor.org/info/rfc1930, doi:10.17487/RFC1930.

[27] John Moy. OSPF Version 2. RFC 2328, April 1998. URL: https://www.rfc-editor.org/info/rfc2328, doi:10.17487/RFC2328.

[28] Karthik Jaganathan, Kristin Lauter, and Larry Zhu. Elliptic Curve Cryptography (ECC) Support for Public Key Cryptography for Initial Authentication in Kerberos (PKINIT). RFC 5349, September 2008. URL: https://www.rfc-editor.org/info/rfc5349, doi:10.17487/RFC5349.

[29] Nischal Sheth, Lili Wang, and Zhaohui (Jeffrey) Zhang. OSPF Hybrid Broadcast and Point-to-Multipoint Interface Type. RFC 6845, January 2013. URL: https://www.rfc-editor.org/info/rfc6845, doi:10.17487/RFC6845.

[30] Yi Yang, Alvaro Retana, and Abhay Roy. Hiding Transit-Only Networks in OSPF. RFC 6860, January 2013. URL: https://www.rfc-editor.org/info/rfc6860, doi:10.17487/RFC6860.

[31] Acee Lindem and Jari Arkko. OSPFv3 Autoconfiguration. RFC 7503, April 2015. URL: https://www.rfc-editor.org/info/rfc7503, doi:10.17487/RFC7503.

[32] Yakov Rekhter, Susan Hares, and Tony Li. A Border Gateway Protocol 4 (BGP-4). RFC 4271, January 2006. URL: https://www.rfc-editor.org/info/rfc4271, doi:10.17487/RFC4271.

[33] Cedric Aoun and Elwyn B. Davies. Reasons to Move the Network Address Translator - Protocol Translator (NAT-PT) to Historic Status. RFC 4966, July 2007. URL: https://www.rfc-editor.org/info/rfc4966, doi:10.17487/RFC4966.

[34] Simon Perreault, Ikuhei Yamagata, Shin Miyakawa, Akira Nakagawa, and Hiroyuki Ashida. Common Requirements for Carrier-Grade NATs (CGNs). RFC 6888, April 2013. URL: https://www.rfc-editor.org/info/rfc6888, doi:10.17487/RFC6888.

[35] Robert E. Gilligan and Erik Nordmark. Basic Transition Mechanisms for IPv6 Hosts and Routers. RFC 4213, October 2005. URL: https://www.rfc-editor.org/info/rfc4213, doi:10.17487/RFC4213.

[36] Dan Wing and Andrew Yourtchenko. Happy Eyeballs: Success with Dual-Stack Hosts. RFC 6555, April 2012. URL: https://www.rfc-editor.org/info/rfc6555, doi:10.17487/RFC6555.

[37] Dr. Paolo Fasano, Dr. Ivano Guardini, Alain Durand, and Domenico Lento. IPv6 Tunnel Broker. RFC 3053, January 2001. URL: https://www.rfc-editor.org/info/rfc3053, doi:10.17487/RFC3053.

[38] Brian E. Carpenter and Keith Moore. Connection of IPv6 Domains via IPv4 Clouds. RFC 3056, February 2001. URL: https://www.rfc-editor.org/info/rfc3056, doi:10.17487/RFC3056.

[39] Ole Trøan and Brian E. Carpenter. Deprecating the Anycast Prefix for 6to4 Relay Routers. RFC 7526, May 2015. URL: https://www.rfc-editor.org/info/rfc7526, doi:10.17487/RFC7526.

[40] Mark Townsley and Ole Trøan. IPv6 Rapid Deployment on IPv4 Infrastructures (6rd) – Protocol Specification. RFC 5969, August 2010. URL: https://www.rfc-editor.org/info/rfc5969, doi:10.17487/RFC5969.

[41] Christian Huitema. Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs). RFC 4380, February 2006. URL: https://www.rfc-editor.org/info/rfc4380, doi:10.17487/RFC4380.

[42] Tim Gleeson, Dave Thaler, and Fred Templin. Intra-Site Automatic Tunnel Addressing Protocol (ISATAP). RFC 5214, March 2008. URL: https://www.rfc-editor.org/info/rfc5214, doi:10.17487/RFC5214.

[43] Alain Durand, Ralph Droms, Yiu Lee, and james woodyatt. Dual-Stack Lite Broadband Deployments Following IPv4 Exhaustion. RFC 6333, August 2011. URL: https://www.rfc-editor.org/info/rfc6333, doi:10.17487/RFC6333.

[44] RedIRIS. Política de peering de rediris, 2026. Accedido el 8 de enero de 2026. URL: https://www.rediris.es/lared/RedIRIS_Peering_Policy.pdf.

[45] Redimadrid - página oficial. https://www.redimadrid.es/quehacemos.html. Accedido: 06-01-2026.

[46] Servicios Informáticos, Universidad Complutense de Madrid. Red de comunicaciones — servicio de conexión a internet y servicios en red de la ucm, 2025. Consultado el 21 de diciembre de 2025. URL: https://ssii.ucm.es/red-de-comunicaciones.

[47] Tom Coffeen. *IPv6 Address Planning: Designing an Address Plan for the Future*. O'Reilly Media, 1st edition, 2014.

[48] Geoff Huston, Anne Lord, and Dr. Philip F. Smith. IPv6 Address Prefix Reserved for Documentation. RFC 3849, July 2004. URL: https://www.rfc-editor.org/info/rfc3849, doi:10.17487/RFC3849.

[49] GNS3 Marketplace. Ubuntu with gui appliance, 2026. Accedido el 13 de enero de 2026. URL: https://www.gns3.com/marketplace/appliances/ubuntu-with-gui.

[50] GNS3 Marketplace. Kali linux 2 appliance, 2025. Accedido el 13 de enero de 2026. URL: https://gns3.com/marketplace/appliances/kali-linux-2.

[51] Vince Fuller and Tony Li. Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan. RFC 4632, August 2006. URL: https://www.rfc-editor.org/info/rfc4632, doi:10.17487/RFC4632.

[52] Paul Shannon, Andrew Markiel, Owen Ozier, Nitin S. Baliga, J. T. Wang, Daniel Ramage, Nada Amin, Benno Schwikowski, and Trey Ideker. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome Research*, 13(11):2498–2504, 2003. doi:10.1101/gr.1239303.

[53] noVNC Project. novnc: Html5 vnc client. https://novnc.com/info.html, 2025. Accedido el 6 de enero de 2026.

[54] Django Software Foundation. Daphne: Http, http/2 y websocket server para asgi, 2025. Consultado el 26 de diciembre de 2025. URL: https://github.com/django/daphne.

[55] Django Software Foundation. Channels — routing, 2025. Consultado el 26 de diciembre de 2025. URL: https://channels.readthedocs.io/en/latest/topics/routing.html.

[56] Marc van Hauser Heuse. thc-ipv6, November 2020. URL: https://github.com/vanhauser-thc/thc-ipv6.

[57] Ran Atkinson and Michael StJohns. Common Architecture Label IPv6 Security Option (CALIPSO). RFC 5570, July 2009. URL: https://www.rfc-editor.org/info/rfc5570, doi:10.17487/RFC5570.

[58] Nalini Elkins, Rob Hamilton, and michael ackermann. IPv6 Performance and Diagnostic Metrics (PDM) Destination Option. RFC 8250, September 2017. URL: https://www.rfc-editor.org/info/rfc8250, doi:10.17487/RFC8250.

[59] Nan Geng, Zhenbin Li, Ran Pang, and Huiyue Zhang. A YANG Model for Application-aware Networking (APN). Internet-Draft draft-geng-rtgwg-apn-yang-00, Internet Engineering Task Force, November 2025. Work in Progress. URL: https://datatracker.ietf.org/doc/draft-geng-rtgwg-apn-yang/00/.

[60] Pasi Sarolahti, Amit Jain, Sally Floyd, and Mark Allman. Quick-Start for TCP and IP. RFC 4782, January 2007. URL: https://www.rfc-editor.org/info/rfc4782, doi:10.17487/RFC4782.

[61] Shwetha Bhandari and Frank Brockners. IPv6 Options for In Situ Operations, Administration, and Maintenance (IOAM). RFC 9486, September 2023. URL: https://www.rfc-editor.org/info/rfc9486, doi:10.17487/RFC9486.

[62] Luis Costa and Rolland Vida. Multicast Listener Discovery Version 2 (MLDv2) for IPv6. RFC 3810, June 2004. URL: https://www.rfc-editor.org/info/rfc3810, doi:10.17487/RFC3810.

[63] Fortinet. Configuring the management interface, 2023. URL: https://docs.fortinet.com/document/fortigate/7.0.0/ngfw-deployment/436050/configuring-the-management-interface.

[64] Fortinet. Configuring bgp, 2026. URL: https://docs.fortinet.com/document/fortigate/7.0.0/sd-wan-sd-branch-deployment-guide/208496/configuring-bgp.

[65] Fortinet. Firewall policy, 2026. URL: https://docs.fortinet.com/document/fortigate/7.6.5/administration-guide/656084/firewall-policy.

[66] Fortinet. Fortios ips snort signatures. https://github.com/fortinet/fortios-ips-snort, 2024. Accedido: enero 2026.

[67] Fortinet. Install preview support for partial install, 2026. URL: https://docs.fortinet.com/document/fortimanager/7.4.0/new-features/419272/install-preview-support-for-partial-install.

[68] GNS3 Marketplace. Fortigate appliance, 2026. Accedido el 6 de enero de 2026. URL: https://gns3.com/marketplace/appliances/fortigate.