

Tarea 3

Diego Franco
Vicente González

04 de Julio de 2025

Taller de redes y servicios

Profesor: Pablo Palacios
Sección 1

Índice

1. Introducción	4
2. Capítulo I. Introducción al protocolo RTSP	5
2.1. Introducción	5
2.1.1. Descripción del protocolo	5
2.1.2. Importancia y relevancia del protocolo	5
2.2. Historia del protocolo	6
2.2.1. Creación y desarrollo inicial	6
2.2.2. Evolución a lo largo del tiempo	6
2.3. Funcionamiento del protocolo	7
2.3.1. Descripción del funcionamiento	7
2.3.2. Detalles técnicos del protocolo	7
2.4. Aplicaciones, clientes y servicios	8
2.4.1. Descripción de las aplicaciones, clientes y servidores conocidos	8
2.5. Actualidad del protocolo	10
2.5.1. Uso actual del protocolo	10
2.5.2. Comunidades	10
2.6. Instalación del software	11
2.6.1. Instalación de Docker	11
2.6.2. Configuración servidor	17
2.6.3. Cliente	18
2.6.4. Conexión Cliente-Servidor	21
2.6.5. Verificación de la transmisión	24
3. Capítulo II. Análisis de tráfico	26
3.1. Detalle en el tráfico capturado	26
3.1.1. Paquete de Conexión al servidor	27
3.2. Suposiciones	28
4. Conclusión Capítulo II	29
5. Capítulo III. Modificación de paquetes con Scapy.	30
5.1. Introducción Capítulo III	30
5.2. Desarrollo	30
5.2.1. Configuración Scapy con Docker	30
5.2.2. Intercepción de tráfico	34
5.2.3. Modificación de tráfico	36
5.2.4. Hipótesis modificación de tráfico	39
5.3. Análisis modificación del tráfico con Scapy	40
5.3.1. Primer paquete modificado	41
5.3.2. Segundo Paquete modificado	42

5.3.3. Tercer Paquete modificado	42
6. Conclusión Capítulo III	43

1. Introducción

En este documento se abordará el protocolo RTSP (Real-Time Streaming Protocol), incluyendo aspectos técnicos y descriptivos como su origen, evolución a lo largo del tiempo, importancia actual y funcionamiento desde una perspectiva técnica. Además, se analizarán sus principales aplicaciones en servicios de streaming y se evaluará su vigencia: ¿para qué se utiliza hoy? ¿Sigue cumpliendo con su propósito original?

Asimismo, se explicará cómo implementar una arquitectura cliente-servidor utilizando RTSP en un entorno Linux, haciendo uso de contenedores Docker para aislar los procesos y evitar conflictos con el sistema anfitrión. Para ello, se utilizará `rtsp-simple-server` (actualmente conocido como MediaMTX) como servidor, y FFmpeg como cliente.

El protocolo RTSP fue desarrollado a mediados de los años 90 con el objetivo de proporcionar un sistema que permitiera controlar la transmisión de contenido multimedia entre clientes y servidores. Aunque ha recibido diversas actualizaciones para mejorar su rendimiento, su especificación original sigue siendo la más utilizada debido a su compatibilidad.

Por su parte, Docker es una plataforma de contenedores que permite empaquetar aplicaciones junto con todas sus dependencias, ejecutándolas en entornos aislados del sistema principal. Esto facilita el desarrollo, despliegue y prueba de aplicaciones de manera segura y controlada. El servidor MediaMTX, anteriormente conocido como `rtsp-simple-server`, se desplegará dentro de un contenedor Docker utilizando la imagen oficial proporcionada en su repositorio de GitHub. Este servidor permite publicar, leer, retransmitir y grabar contenido multimedia mediante el protocolo RTSP.

El cliente, basado en FFmpeg, también se ejecutará dentro de un contenedor. A diferencia del servidor, la imagen Docker del cliente será personalizada, ya que las imágenes disponibles públicamente no se ajustan completamente a los requerimientos prácticos de esta demostración. FFmpeg es un conjunto de bibliotecas y herramientas de línea de comandos que permiten procesar audio y video, incluyendo tareas como conversión, transmisión y análisis multimedia.

Finalmente, se realizará un análisis del tráfico de red generado durante la conexión entre el cliente FFmpeg y el servidor MediaMTX, utilizando la herramienta Wireshark, la cual permite inspeccionar y visualizar el flujo de paquetes en la red. Esto permitirá verificar la correcta transmisión de datos mediante el protocolo RTSP, así como observar en tiempo real los detalles de los protocolos de red que permiten la conexión, como el establecimiento de la conexión, intercambio de comandos, y transferencia del flujo multimedia.

2. Capítulo I. Introducción al protocolo RTSP

2.1. Introducción

2.1.1. Descripción del protocolo

El protocolo RTSP (Real-Time Streaming Protocol, o Protocolo de Transmisión en Tiempo Real) es un protocolo de control que opera en la capa de aplicación del modelo OSI. Está diseñado para gestionar la transmisión de contenidos multimedia en tiempo real, como audio y vídeo, en aplicaciones de entretenimiento, videovigilancia y comunicaciones interactivas. A diferencia de otros protocolos de transmisión, RTSP no transporta directamente los datos multimedia. Su función principal es permitir el control de la reproducción, con comandos como iniciar, pausar, reanudar y detener la transmisión, funcionando de forma similar a un control remoto. RTSP se utiliza comúnmente en conjunto con el protocolo RTP (Real-time Transport Protocol), que es el encargado de transportar los datos multimedia en tiempo real, asegurando una entrega adecuada y sincronizada del contenido.

2.1.2. Importancia y relevancia del protocolo

La importancia del protocolo RTSP radica en su capacidad para proporcionar un control interactivo y en tiempo real sobre la reproducción de contenido multimedia. Este tipo de control es esencial en sistemas donde el cliente necesita intervenir directamente en la gestión del flujo de datos, como en:

- Sistemas de videovigilancia con cámaras IP
- Servicios de streaming en redes locales o privadas
- Videoconferencias y comunicaciones en tiempo real
- Prototipos o herramientas de desarrollo, como ffmpeg o rtsp-simple-server

Gracias a su baja latencia, compatibilidad con herramientas abiertas y facilidad de implementación, RTSP sigue siendo una opción muy relevante frente a otros protocolos como HLS o RTMP, especialmente en aplicaciones donde la interactividad y el control en tiempo real son críticos.

2.2. Historia del protocolo

2.2.1. Creación y desarrollo inicial

El protocolo RTSP (Real-Time Streaming Protocol) fue desarrollado a mediados de los años 90 por tres principales colaboradores: RealNetworks, Netscape Communications y la Universidad de Columbia. Su objetivo principal era crear un mecanismo que permitiera controlar la transmisión de medios multimedia entre un cliente y un servidor, funcionando de manera similar a una videgrabadora remota, con comandos como reproducir, pausar o detener. RTSP fue estandarizado en 1998 por la IETF (Internet Engineering Task Force) como *RFC 2326*, y rápidamente ganó popularidad debido a su utilidad para reproducir audio y video directamente desde internet, sin necesidad de descargar previamente los archivos en el dispositivo del usuario. El protocolo se basó en conceptos y estándares existentes en la época, como HTTP, adoptando una estructura de comunicación similar (peticiones y respuestas en texto plano). Además, integró el uso del protocolo SDP (Session Description Protocol) para describir y gestionar sesiones de comunicación multimedia de manera flexible.

2.2.2. Evolución a lo largo del tiempo

A lo largo del tiempo, el protocolo RTSP ha experimentado diversas mejoras relacionadas con la seguridad, la eficiencia y la adaptación a nuevas necesidades tecnológicas. La versión 1.0, definida en la *RFC 2326*, fue la primera especificación estandarizada por la IETF en 1998, y no fue actualizada hasta casi dos décadas después, cuando en 2016 se publicó la versión 2.0 bajo la *RFC 7826*. RTSP 1.0 proporciona un conjunto de comandos para el control de transmisiones en tiempo real, tales como; PLAY, PAUSE, SETUP y TEARDOWN, entre otros. Está basado en HTTP/1.1, aunque con un estilo y funcionalidad más orientados a la gestión de sesiones multimedia. Además, es compatible con RTP/RTCP para el transporte de datos multimedia y con SDP (Session Description Protocol) para la descripción de las sesiones. La versión 2.0 representa una evolución significativa del protocolo, con una estructura más robusta y limpia, completamente basada en una sintaxis más coherente con HTTP. Entre sus principales mejoras destacan:

- Un manejo más eficiente de las sesiones de transmisión.
- Resolución de problemas relacionados con la traducción de direcciones de red (NAT).
- Eliminación de ambigüedades presentes en la versión anterior.
- Soporte para pipelining y multiplexación, que mejoran el rendimiento y reducen la latencia.

Sin embargo, a pesar de las mejoras introducidas en RTSP 2.0, muchos sistemas aún utilizan la versión original (1.0), especialmente en aplicaciones como la videovigilancia, debido a su bajo costo de implementación, amplia compatibilidad con dispositivos existentes y su simplicidad operativa.

2.3. Funcionamiento del protocolo

2.3.1. Descripción del funcionamiento

Como se mencionó anteriormente, el protocolo RTSP (Real-Time Streaming Protocol) facilita el control de transmisiones de medios en tiempo real (audio y video). Su funcionamiento se basa en un modelo de intercambio de solicitudes y respuestas entre un cliente y un servidor. A través de este mecanismo, el cliente puede enviar comandos para controlar la reproducción del contenido, como iniciar, pausar o detener la transmisión. Es importante destacar que RTSP no transporta directamente los datos multimedia. En su lugar, se encarga de transmitir las instrucciones del cliente al servidor. Los datos del audio o video son enviados a través de protocolos complementarios, principalmente RTP (Real-Time Transport Protocol).

2.3.2. Detalles técnicos del protocolo

A continuación, se describen los comandos principales utilizados en una sesión RTSP:

- **DESCRIBE:** El cliente solicita al servidor una descripción técnica del recurso multimedia. La respuesta, generalmente en formato SDP (Session Description Protocol), incluye información como codecs, duración, tipo de medio y puertos requeridos.
- **SETUP:** El cliente especifica cómo desea recibir el contenido (por ejemplo, si usará TCP o UDP, y qué puertos utilizará). El servidor responde con los parámetros de configuración aceptados y proporciona un identificador de sesión.
- **PLAY:** Se envía para iniciar la reproducción del contenido multimedia. A partir de este momento, el servidor comienza a transmitir los datos usando RTP, y el cliente los recibe y reproduce en tiempo real.
- **PAUSE / TEARDOWN:** Permiten al cliente pausar temporalmente la transmisión o finalizar la sesión, respectivamente.

Durante la fase SETUP, el protocolo RTSP permite al cliente definir el método de transporte a utilizar, siendo los más comunes TCP, que ofrece mayor confiabilidad y es ideal para redes con firewall o NAT, y UDP, que permite una transmisión más rápida pero con mayor riesgo de pérdida de paquetes en redes inestables. Es fundamental seleccionar correctamente los puertos asociados a cada protocolo, ya que una configuración inadecuada puede generar conflictos con firewalls o sistemas de traducción de direcciones (NAT), impidiendo la correcta entrega del contenido multimedia.

2.4. Aplicaciones, clientes y servicios

2.4.1. Descripción de las aplicaciones, clientes y servidores conocidos

1. Clientes RTSP.

Los clientes RTSP son aplicaciones o dispositivos que, mediante una conexión previa con el servidor, pueden solicitar y controlar flujos de contenido multimedia en tiempo real. Algunos disponen de una interfaz gráfica para facilitar al usuario la interacción con los medios, permitiéndole ejecutar instrucciones como las mencionadas anteriormente (iniciar, pausar, detener, reanudar, etc.). Entre los clientes más conocidos se encuentran:

- FFmpeg: Proyecto de software de código abierto que permite grabar, convertir, transmitir y reproducir archivos multimedia mediante las herramientas que proporciona. Su gran versatilidad lo convierte en una opción muy popular en la línea de comandos y está disponible para diversas plataformas. Es ampliamente utilizado para manipular archivos multimedia, prepararlos para su reproducción en distintos dispositivos o para su transmisión en streaming.
- VLC Media Player: Uno de los pioneros entre los reproductores de vídeo. Es un reproductor multimedia gratuito, de código abierto y multiplataforma. Es ampliamente reconocido por su capacidad para reproducir una gran variedad de formatos de contenido multimedia, incluyendo aquellos que requieren códecs adicionales para su decodificación.
- OBS Studio (Open Broadcaster Software): Software gratuito y de código abierto diseñado para la grabación y transmisión en vivo de vídeo. Su facilidad de uso lo convierte en una herramienta ideal para streamers, creadores de contenido y docentes que desean grabar clases o eventos y compartirlos en línea como material educativo.

2. Servidores RTSP.

Los servidores RTSP son responsables de emitir contenido multimedia en tiempo real. No se encargan directamente del transporte de los datos, sino que establecen la sesión de transmisión y controlan las acciones relacionadas con ella, como iniciar, pausar, detener o reanudar la transmisión según las instrucciones del cliente. El transporte de los datos se realiza generalmente con el apoyo de otros protocolos, como RTP (Protocolo de Transporte en Tiempo Real), que puede utilizar TCP o UDP, dependiendo del caso. La conexión entre cliente y servidor es crucial, ya que el cliente es quien emite las instrucciones para la reproducción del contenido multimedia. Entre los servidores más conocidos destacan:

- MediaMTX (anteriormente rtsp-simple-server): Servidor y proxy multimedia de código abierto que permite publicar, leer, retransmitir, grabar y reproducir flujos de contenido multimedia. Está diseñado como un enrutador multimedia de extremo a extremo para flujos RTSP.

- GStreamer: Framework multimedia de código abierto que permite la creación de aplicaciones para la gestión, edición, transmisión y procesamiento de contenido multimedia. Su arquitectura se basa en canalizaciones de elementos, cada uno con una función específica, que pueden combinarse para realizar tareas complejas de procesamiento multimedia.
- Wowza Streaming Engine: Software comercial robusto y escalable para la transmisión de contenido multimedia. Ofrece múltiples herramientas (SDKs) y APIs que permiten a los desarrolladores crear soluciones de streaming innovadoras y adaptables, acelerando el ciclo de desarrollo y satisfaciendo diversas necesidades del público objetivo.

3. Aplicaciones RTSP.

El protocolo RTSP se utiliza ampliamente en la actualidad para la transmisión y control de medios en tiempo real, siendo especialmente útil en entornos donde se requiere interacción con el contenido transmitido. Entre las aplicaciones más comunes se encuentran:

- Videovigilancia: RTSP es fundamental en sistemas de seguridad, ya que permite la transmisión en vivo de cámaras IP. Gracias a su capacidad de control, los usuarios pueden pausar, buscar o revisar partes específicas del contenido grabado, facilitando la supervisión y la revisión de incidentes.
- Streaming multimedia: Facilita la transmisión de contenido multimedia bajo demanda, como programas de televisión, películas, series, y más, a través de servicios en línea compatibles con el protocolo. Se ha utilizado en sitios web de cámaras en vivo (webcams), plataformas de educación en línea, y sistemas de radio por internet.
- Videoconferencias: RTSP puede ser empleado para establecer sesiones de videoconferencia con transmisión en tiempo real, garantizando sincronización y control sobre el flujo de datos. Aunque protocolos como WebRTC son más comunes en aplicaciones modernas de videoconferencia, RTSP aún se emplea en soluciones más específicas o integradas con sistemas de vigilancia y monitoreo.

2.5. Actualidad del protocolo

2.5.1. Uso actual del protocolo

Actualmente, el protocolo RTSP (Real-Time Streaming Protocol) sigue siendo una herramienta vigente y esencial para la transmisión de contenidos multimedia en tiempo real, especialmente en el sector de la videovigilancia, donde se requiere un control preciso sobre el contenido transmitido. Aunque con el tiempo han surgido nuevos protocolos orientados al entorno web que han desplazado parcialmente a RTSP, este continúa siendo una opción sólida gracias a su amplia compatibilidad con dispositivos, su bajo costo de implementación y su eficiencia en redes locales. En la actualidad, RTSP se utiliza principalmente en los siguientes contextos:

- Sistemas de videovigilancia y cámaras IP: Su capacidad de control, baja latencia y eficiencia lo convierten en un protocolo ideal para monitoreo en tiempo real en entornos domésticos, comerciales e industriales.
- Herramientas y aplicaciones de desarrollo: Soluciones como FFmpeg, VLC Media Player o rtsp-simple-server emplean RTSP para funciones como la captura, prueba o retransmisión de flujos de video, siendo una herramienta clave para desarrolladores, técnicos y entusiastas del video.

Sin embargo, a pesar de su robustez, RTSP no es completamente compatible con navegadores y reproductores multimedia modernos, los cuales están optimizados para protocolos de transmisión más adaptados al entorno web, como HLS (HTTP Live Streaming) o MPEG-DASH, utilizados por plataformas como YouTube, Netflix o Twitch. Esto ha limitado su uso en entornos de consumo masivo, aunque sigue siendo altamente valorado en aplicaciones técnicas, profesionales y de infraestructura.

2.5.2. Comunidades

Las comunidades en torno al protocolo RTSP están conformadas por equipos de desarrolladores, empresas y proyectos de código abierto que comparten un interés común en su funcionamiento. Estas comunidades colaboran activamente para mejorar, mantener y aplicar el protocolo en distintos entornos. Su trabajo se centra en compartir conocimientos, desarrollar nuevas herramientas, resolver errores y proponer mejoras que puedan optimizar el uso del protocolo. La mayoría de estas comunidades se desarrollan en plataformas de colaboración técnica como GitHub, Stack Overflow, YouTube, entre otras. Gracias a esta colaboración constante, han surgido numerosos proyectos respaldados por comunidades activas que investigan e implementan mejoras prácticas sobre el protocolo RTSP. Algunos ejemplos representativos son:

- Live555: Biblioteca escrita en C++ especializada en RTSP, RTP, HLS y otros protocolos de transmisión. Cuenta con una comunidad que contribuye a su mantenimiento mediante foros, actualizaciones y documentación técnica.

- MediaMTX: Uno de los proyectos más modernos y activos relacionados con RTSP. Su documentación se mantiene actualizada constantemente gracias al trabajo conjunto de usuarios que participan en su desarrollo. Es una opción sólida como servidor RTSP por su facilidad de uso y flexibilidad.
- Stack Overflow: Aunque no es un proyecto exclusivo de RTSP, esta plataforma es un punto de encuentro clave para desarrolladores que tienen dudas o enfrentan errores en la implementación del protocolo. Muchas soluciones y discusiones útiles pueden encontrarse allí.
- YouTube: En esta plataforma es posible acceder a una gran cantidad de contenido relacionado con RTSP, desde tutoriales hasta análisis técnicos, lo que resulta útil tanto para usuarios principiantes como para entusiastas avanzados que buscan innovar o entender a fondo el protocolo.

2.6. Instalación del software

Tanto el cliente como el servidor deberán instalar el sistema de contenedores Docker en sus respectivas máquinas Ubuntu. Esto permitirá ejecutar los procesos de servidor y cliente en entornos aislados, evitando conflictos con otros procesos o dependencias del sistema.

El servidor ejecutará un contenedor basado en la imagen predefinida MediaMTX, la cual habilita el protocolo RTSP a través del puerto TCP 8554, permitiendo que el cliente se conecte para la transmisión de datos multimedia.

Por su parte, el cliente ejecutará un contenedor basado en una imagen personalizada con ffmpeg, el cual será responsable de realizar la conexión con el servidor y transmitir el contenido. Este proceso se detallará en los siguientes apartados:

- Instalación de docker
- Configuración servidor
- Configuración cliente
- Conexión cliente-servidor

2.6.1. Instalación de Docker

Esta instalación es la misma para cliente y servidor.

Dado que ambos roles utilizarán contenedores Docker, se debe realizar la instalación en ambas máquinas siguiendo los mismos pasos:

1. Preparar el sistema

Antes de instalar el sistema de contenedores Docker, es necesario preparar el entorno instalando las dependencias requeridas para su correcto funcionamiento. Primero, se deben actualizar los paquetes disponibles en los repositorios del sistema con el siguiente comando:

```
sudo apt update
```

Una vez completada la actualización de los paquetes, se procede a instalar las siguientes dependencias:

```
sudo apt install \
ca-certificates \
curl \
gnupg \
lsb-release
```

Donde:

- **ca-certificates:** Proporciona certificados de seguridad para conexiones HTTPS.
- **curl:** Herramienta para realizar peticiones a URLs.
- **gnupg:** Encargado de verificar firmas GPG, necesarias para la autenticidad del repositorio.
- **lsb-release:** Permite identificar la versión de Ubuntu instalada.

```
ubuntu@ubuntu-VMware-Virtual-Platform:~$ sudo apt update
sudo apt install \
  ca-certificates \
  curl \
  gnupg \
  lsb-release
[sudo] contraseña para ubuntu:
Obj:1 http://cl.archive.ubuntu.com/ubuntu noble InRelease
Des:2 http://cl.archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Obj:3 http://cl.archive.ubuntu.com/ubuntu noble-backports InRelease
Des:4 https://download.docker.com/linux/ubuntu noble InRelease [48,8 kB]
Des:5 http://cl.archive.ubuntu.com/ubuntu noble-updates/main amd64 Packages [1.106 kB]
Des:6 http://cl.archive.ubuntu.com/ubuntu noble-updates/universe amd64 Packages [1.067 kB]
Obj:7 http://security.ubuntu.com/ubuntu noble-security InRelease
Des:8 https://download.docker.com/linux/ubuntu noble/stable amd64 Packages [25,0 kB]
Des:9 http://download.zerotier.com/debian/noble no
ble InRelease [20,5 kB]
Descargados 2.393 kB en 10s (229 kB/s)
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
Se pueden actualizar 75 paquetes. Ejecute «apt list --upgradable» para verlos.
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
ca-certificates ya está en su versión más reciente (20240203).
curl ya está en su versión más reciente (8.5.0-2ubuntu10.6).
gnupg ya está en su versión más reciente (2.4.4-2ubuntu17.2).
lsb-release ya está en su versión más reciente (12.0-2).
0 actualizados, 0 nuevos se instalarán, 0 para eliminar y 75 no actualizados.
```

Figura 1: Paso 1. Instalar paquetes necesarios.

2. Agregar la clave GPG oficial de Docker

Con las dependencias ya instaladas, se debe agregar la clave GPG oficial del repositorio

de Docker. Esta clave garantiza la autenticidad e integridad de los paquetes. Primero, se crea la carpeta donde se almacenará la clave:

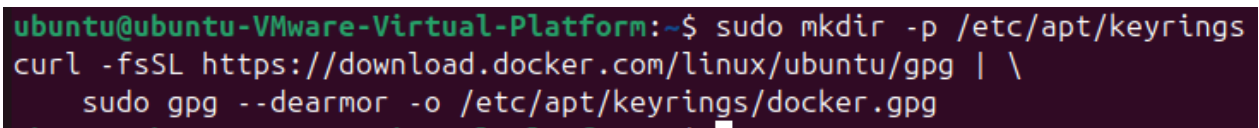
```
sudo mkdir -p /etc/apt/keyrings
```

Luego, se descarga y convierte la clave GPG desde el sitio oficial de Docker:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | \
sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
```

Donde:

- **curl:** descarga la clave GPG directamente desde el repositorio.
- **gpg --dearmor:** Convierte la clave de formato ASCII a binario y la guarda como “docker.gpg” en la carpeta creada.



```
ubuntu@ubuntu-VMware-Virtual-Platform:~$ sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | \
sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
```

Figura 2: Paso 2. Descargar y ajustar formato de la clave.

3. Agregar el repositorio oficial

Con la clave GPG ya instalada, se agrega el repositorio oficial de Docker a la configuración de APT:

```
echo \
"deb [arch=$(dpkg --print-architecture) \
signed-by=/etc/apt/keyrings/docker.gpg] \
https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

Donde:

- **arch=\$(dpkg --print-architecture):** Detecta la arquitectura del sistema.
- **signed-by=/etc/apt/keyrings/docker.gpg:** Indica a APT que confíe en el repositorio “https://download.docker.com/linux/ubuntu” solo si los paquetes están firmados con la clave especificada.
- **\$(lsb_release -cs) stable:** Obtiene el nombre clave de la versión de Ubuntu utilizada.

- **sudo tee /etc/apt/sources.list.d/docker.list >/dev/null:** El comando **sudo tee** le permite a **echo** realizar escritura de archivo, dado que **echo** por sí solo no tiene permisos para crear/modificar archivos dentro de la carpeta *apt*, que es donde se crea el archivo *docker.list* el cual contendrá la información que se obtuvo del comando previamente (arquitectura, restricción de clave GPG, dirección del repositorio y nombre clave de Ubuntu). Finalmente **>/dev/null** se encarga de que la información obtenida no sea impresa en la terminal.

Nota: Se utilizan comillas dobles en el comando echo porque incluye variables (\$()), y permite manejar correctamente los espacios y saltos de línea. Sin estas comillas, el comando no se ejecutará correctamente.

```
ubuntu@ubuntu-VMware-Virtual-Platform:~$ echo \
"deb [arch=$(dpkg --print-architecture) \
signed-by=/etc/apt/keyrings/docker.gpg] \
https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

Figura 3: Paso 3. Agregar el repositorio oficial de Docker.

4. Instalar Docker Engine

Una vez agregado el repositorio oficial de Docker al sistema se debe realizar la instalación de los componentes de Docker en sí:

```
sudo apt install docker-ce docker-ce-cli containerd.io \
docker-buildx-plugin docker-compose-plugin
```

Donde:

- **docker-ce:** Se especifica que se instalará Docker Community Edition.
- **docker-ce-cli:** La interfaz de línea de comandos de Docker.
- **containerd.io:** Servicio encargado de la gestión de los contenedores Docker.
- **docker-buildx-plugin:** Herramienta para la construcción de imágenes.
- **docker-compose-plugin:** Permite usar “docker compose” de manera nativa.

```
ubuntu@ubuntu-VMware-Virtual-Platform:~$ sudo apt install docker-ce docker-ce-cli containerd.io docker-buildx-plugin doc
ker-compose-plugin
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes adicionales:
  docker-ce-rootless-extras git git-man liberror-perl libslirp0 pigz slirp4netns
Paquetes sugeridos:
  cgroupfs-mount | cgroup-lite git-daemon-run | git-daemon-sysvinit git-doc git-email git-gui gitk gitweb git-cvs
  git-mediawiki git-svn
Se instalarán los siguientes paquetes NUEVOS:
  containerd.io docker-buildx-plugin docker-ce docker-ce-cli docker-ce-rootless-extras docker-compose-plugin git
  git-man liberror-perl libslirp0 pigz slirp4netns
0 actualizados, 12 nuevos se instalarán, 0 para eliminar y 68 no actualizados.
Se necesita descargar 125 MB de archivos.
Se utilizarán 464 MB de espacio de disco adicional después de esta operación.
¿Desea continuar? [S/n] s
Des:1 http://cl.archive.ubuntu.com/ubuntu noble/universe amd64 pigz amd64 2.8-1 [65,6 kB]
Des:2 http://cl.archive.ubuntu.com/ubuntu noble/main amd64 liberror-perl all 0.17029-2 [25,6 kB]
Des:3 http://cl.archive.ubuntu.com/ubuntu noble-updates/main amd64 git-man all 1:2.43.0-1ubuntu7.2 [1.100 kB]
Des:4 https://download.docker.com/linux/ubuntu noble/stable amd64 containerd.io amd64 1.7.27-1 [30,5 MB]
Des:5 http://cl.archive.ubuntu.com/ubuntu noble-updates/main amd64 git amd64 1:2.43.0-1ubuntu7.2 [3.679 kB]
Des:6 http://cl.archive.ubuntu.com/ubuntu noble/main amd64 libslirp0 amd64 4.7.0-1ubuntu3 [63,8 kB]
Des:7 http://cl.archive.ubuntu.com/ubuntu noble/universe amd64 slirp4netns amd64 1.2.1-1build2 [34,9 kB]
Des:8 https://download.docker.com/linux/ubuntu noble/stable amd64 docker-buildx-plugin amd64 0.23.0-1~ubuntu.24.04~noble
 [34,6 MB]
Des:9 https://download.docker.com/linux/ubuntu noble/stable amd64 docker-ce-cli amd64 5:28.1.1-1~ubuntu.24.04~noble [15,
8 MB]
Des:10 https://download.docker.com/linux/ubuntu noble/stable amd64 docker-ce amd64 5:28.1.1-1~ubuntu.24.04~noble [19,2 M
B]
Des:11 https://download.docker.com/linux/ubuntu noble/stable amd64 docker-ce-rootless-extras amd64 5:28.1.1-1~ubuntu.24.
04~noble [6.092 kB]
Des:12 https://download.docker.com/linux/ubuntu noble/stable amd64 docker-compose-plugin amd64 2.35.1-1~ubuntu.24.04~nob
```

Figura 4: Paso 4. Instalación de Docker. PARTE 1

```
Preparando para desempaquetar .../08-git-man_1%3a2.43.0-1ubuntu7.2_all.deb ...
Desempaquetando liberror-perl (0.17029-2) ...
Seleccionando el paquete git-man previamente no seleccionado.
Preparando para desempaquetar .../09-git_1%3a2.43.0-1ubuntu7.2_amd64.deb ...
Desempaquetando git-man (1:2.43.0-1ubuntu7.2) ...
Seleccionando el paquete git previamente no seleccionado.
Preparando para desempaquetar .../10-libslirp0_4.7.0-1ubuntu3_amd64.deb ...
Desempaquetando libslirp0:amd64 (4.7.0-1ubuntu3) ...
Seleccionando el paquete slirp4netns previamente no seleccionado.
Preparando para desempaquetar .../11-slirp4netns_1.2.1-1build2_amd64.deb ...
Desempaquetando slirp4netns (1.2.1-1build2) ...
Configurando liberror-perl (0.17029-2) ...
Configurando docker-buildx-plugin (0.23.0-1-ubuntu.24.04~noble) ...
Configurando containerd.io (1.7.27-1) ...
Created symlink /etc/systemd/system/multi-user.target.wants/containerd.service → /usr/lib/systemd/system/containerd.service.
Configurando docker-compose-plugin (2.35.1-1-ubuntu.24.04~noble) ...
Configurando docker-ce-cli (5:28.1.1-1-ubuntu.24.04~noble) ...
Configurando libslirp0:amd64 (4.7.0-1ubuntu3) ...
Configurando pigz (2.8-1) ...
Configurando git-man (1:2.43.0-1ubuntu7.2) ...
Configurando docker-ce-rootless-extras (5:28.1.1-1-ubuntu.24.04~noble) ...
Configurando slirp4netns (1.2.1-1build2) ...
Configurando docker-ce (5:28.1.1-1-ubuntu.24.04~noble) ...
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /usr/lib/systemd/system/docker.service.
Created symlink /etc/systemd/system/sockets.target.wants/docker.socket → /usr/lib/systemd/system/docker.socket.
Configurando git (1:2.43.0-1ubuntu7.2) ...
Procesando disparadores para man-db (2.12.0-4build2) ...
Procesando disparadores para libc-bin (2.39-0ubuntu8.4) ...
ubuntu@ubuntu-VMware-Virtual-Platform:~$
```

Figura 5: Paso 4. Instalación de Docker. PARTE 2

Nota: En las fotografías se muestra el inicio (Figura 4) del proceso y el final (Figura 5).

5. Verificar que Docker funciona

Para comprobar que Docker se ha instalado correctamente, se ejecuta el siguiente comando de prueba:

```
sudo docker run hello-world
```

Este comando descarga una imagen de prueba (hello-world) y la ejecuta en un contenedor. Si todo está configurado correctamente, se mostrará un mensaje de éxito en la terminal. En caso contrario, se notificará un error, indicando que algo falló durante la instalación.


```
ubuntu@ubuntu-VMware-Virtual-Platform:~$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
e6590344b1a5: Pull complete
Digest: sha256:dd01f97f252193ae3210da231b1dca0cffab4aadb3566692d6730bf93f123a48
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (amd64)
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
```

Figura 6: Paso 5. Verificación de Docker mediante la imagen *Hello-world*.

2.6.2. Configuración servidor

La máquina que actúe como servidor deberá ejecutar un contenedor a partir de la imagen predeterminada del servicio MediaMTX (anteriormente rtsp-simple-server). Esta imagen ya viene preconfigurada para utilizar el protocolo RTSP sobre TCP, empleando por defecto el puerto 8554, el cual es el estándar para este tipo de transmisiones.

1. Levantar el servidor

Para iniciar el servidor, se ejecuta el siguiente comando:

```
sudo docker run --rm -it --network=host \
bluenvir/mediamtx:latest
```

```
diego@diego-A320M-S2H:~$ sudo docker run --rm -it --network=host bluvieron/mediamtx:latest
[sudo] contraseña para diego:
2025/05/28 22:46:03 INF MediaMTX v1.12.2
2025/05/28 22:46:03 INF configuration loaded from /mediamtx.yml
2025/05/28 22:46:03 INF [RTSP] listener opened on :8554 (TCP), :8000 (UDP/RTP), :8001 (UDP/RTCP)
2025/05/28 22:46:03 INF [RTMP] listener opened on :1935
2025/05/28 22:46:03 INF [HLS] listener opened on :8888
2025/05/28 22:46:03 INF [WebRTC] listener opened on :8889 (HTTP), :8189 (ICE/UDP)
2025/05/28 22:46:03 INF [SRT] listener opened on :8890 (UDP)
```

Figura 7: Levantamiento del servidor

Donde:

- **-rm:** Indica que una vez se detenga el contenedor, se eliminará.
- **-it:** Permite interacción con el contenedor a través de la terminal.
- **-network=host:** Comparte la red del host con el contenedor, permitiendo que el puerto 8554 sea accesible directamente desde la IP del servidor.

2.6.3. Cliente

La máquina que cumpla el rol de cliente deberá construir una imagen personalizada de Docker que incluya el cliente FFmpeg, el cual será utilizado para transmitir contenido al servidor RTSP. Esta imagen también permitirá la interacción directa desde el contenedor.

1. Crear carpeta que almacenará la imagen personalizada

Primero, se debe crear la carpeta en la que se guardará la imagen que se creará para el cliente ffmpeg:

```
mkdir ffmpeg-custom
cd ffmpeg-custom
```

2. Crear archivo llamado Dockerfile

Dentro de la carpeta se crea un archivo vacío llamado Dockerfile, que contendrá las instrucciones de la construcción de la imagen:

```
touch Dockerfile
```

```
ubuntu@ubuntu-VMware-Virtual-Platform:~$ mkdir ffmpeg-custom
ubuntu@ubuntu-VMware-Virtual-Platform:~$ cd ffmpeg-custom
ubuntu@ubuntu-VMware-Virtual-Platform:~/ffmpeg-custom$ touch Dockerfile
```

Figura 8: Paso 1 y 2. Creación de la carpeta *ffmpeg-custom* y del archivo de configuración *Dockerfile*.

3. Acceder a Dockerfile

Se debe abrir el archivo para editarlo, esto se realiza desde dentro de la carpeta:

```
nano Dockerfile
```

4. Instrucciones para el Dockerfile

Dentro del archivo, se deben escribir las siguientes instrucciones:

```
FROM ubuntu:20.04

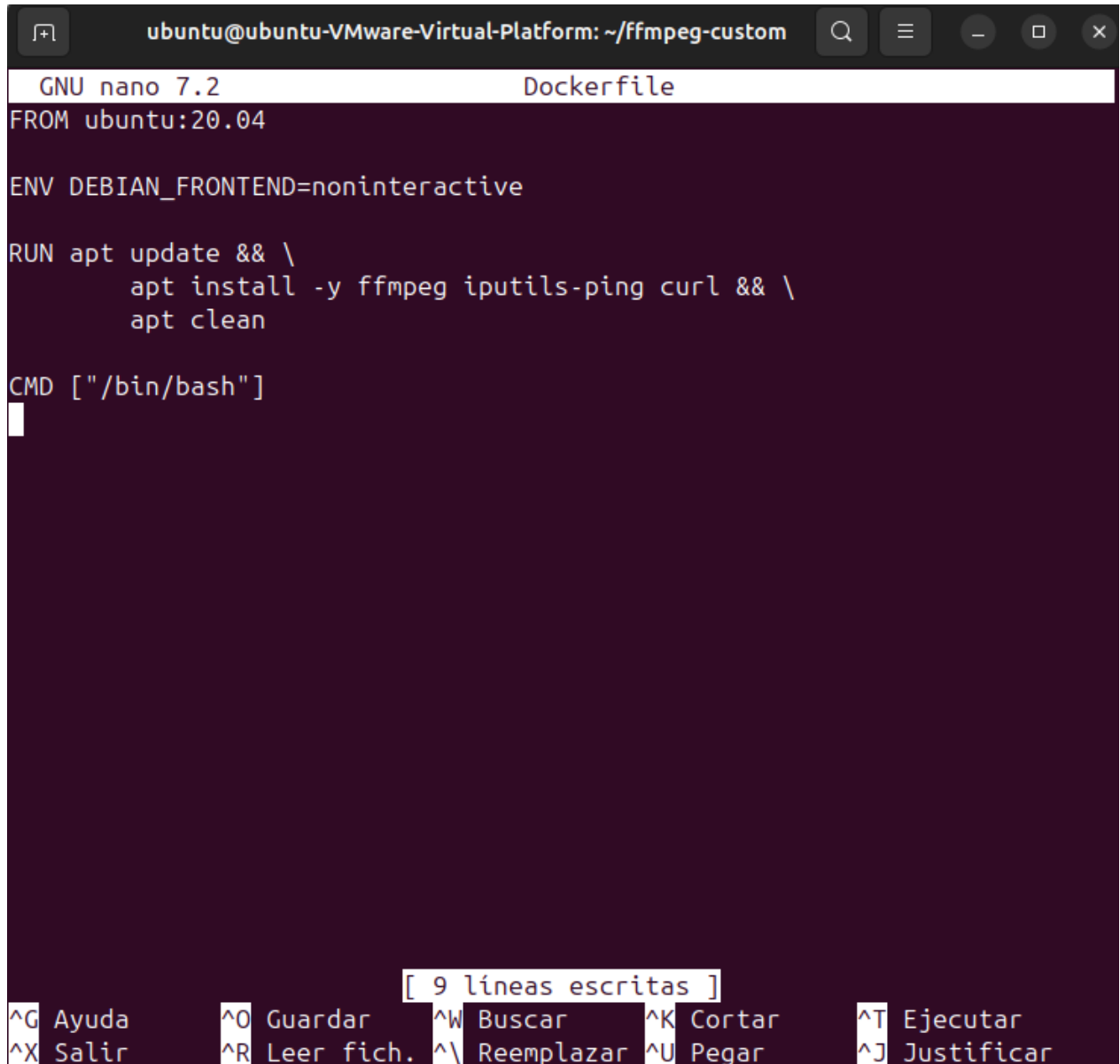
ENV DEBIAN_FRONTEND=noninteractive

RUN apt update && \
    apt install -y ffmpeg iputils-ping && \
    apt clean

CMD ["/bin/bash"]
```

Descripción:

- Se utiliza Ubuntu 20.04 como imagen base para la configuración.
- Se configura para no requerir interacción desde la terminal durante la instalación.
- Se instala el cliente ffmpeg y la herramienta ping.
- Se limpia la caché de la instalación, para reducir el tamaño de la imagen.
- Al ejecutar el contenedor, se abrirá una terminal bash.



```

ubuntu@ubuntu-VMware-Virtual-Platform: ~/ffmpeg-custom
GNU nano 7.2 Dockerfile
FROM ubuntu:20.04

ENV DEBIAN_FRONTEND=noninteractive

RUN apt update && \
    apt install -y ffmpeg iputils-ping curl && \
    apt clean

CMD ["/bin/bash"]

```

[9 líneas escritas]

^G Ayuda ^O Guardar ^W Buscar ^K Cortar ^T Ejecutar
 ^X Salir ^R Leer fich. ^\ Reemplazar ^U Pegar ^J Justificar

Figura 9: Paso 4. Conjunto de instrucciones dentro de *Dockerfile*.

5. Crear imagen personalizada

Con el Dockerfile listo, se crea la imagen personalizada bajo el siguiente comando:

```
docker build -t ffmpeg-custom .
```

Esto creará una imagen personalizada llamada “ffmpeg-custom” con las instrucciones descritas dentro del Dockerfile.

```
ubuntu@ubuntu-VMware-Virtual-Platform:~/ffmpeg-custom$ sudo docker build
-t ffmpeg-custom .
[+] Building 0.1s (6/6) FINISHED                                docker:default
=> [internal] load build definition from Dockerfile             0.0s
=> => transferring dockerfile: 190B                             0.0s
=> [internal] load metadata for docker.io/library/ubuntu:20.04 0.0s
=> [internal] load .dockerignore                               0.0s
=> => transferring context: 2B                                   0.0s
=> [1/2] FROM docker.io/library/ubuntu:20.04                  0.0s
=> CACHED [2/2] RUN apt update && apt install -y ffmpeg iputils- 0.0s
=> exporting to image                                           0.0s
=> => exporting layers                                          0.0s
=> => writing image sha256:9b888a13da25080a1f7563f6ebe62532f2db9e 0.0s
=> => naming to docker.io/library/ffmpeg-custom               0.0s
ubuntu@ubuntu-VMware-Virtual-Platform:~/ffmpeg-custom$
```

Figura 10: Paso 5. Creación de la imagen *ffmpeg-custom*.

2.6.4. Conexión Cliente-Servidor

Antes de establecer la conexión entre cliente y servidor, es importante comprender que el protocolo RTSP está diseñado principalmente para funcionar dentro de redes locales (LAN). Intentar implementarlo directamente a través de Internet implica configuraciones adicionales como apertura de puertos, reglas de firewall, traducción de direcciones (NAT), entre otros aspectos que dificultan innecesariamente esta práctica. Por ello, se optará por utilizar una VPN para simular una red local entre cliente y servidor, permitiendo una comunicación más simple y directa.

1. Conexión a la VPN ZeroTier

Se utilizará ZeroTier, una solución de red definida por software que permite interconectar dispositivos remotos como si estuvieran en la misma LAN. Tanto el cliente como el servidor deberán instalar y conectarse a la red virtual creada en ZeroTier. Para ello, se debe realizar la instalación del programa mediante el siguiente comando:

```
curl -s https://install.zerotier.com/ | sudo bash
```

Una vez instalado, cada dispositivo puede unirse o salirse de la red virtual ejecutando:

```
sudo zerotier-cli join/leave ID_VPN
```

Donde ID_VPN es un ID único de red que proporciona ZeroTier.

En la interfaz de administración dentro de la página de ZeroTier, el administrador debe aceptar a los usuarios.

Una vez aceptados, dentro del panel de administración se verá algo así:

Members

AUTHORIZATION

All ☒
Authorized ☐ (2)
Not authorized ☐ (0)

ACTIVITY

All ☒
Inactive ☐ (1)
Active ☐ (1)

Reset Filters

2 total members

2 filtered members

Refresh

<input type="checkbox"/>	Edit	Auth	Address	Name/Desc	Managed IPs	Last Seen	Version	Physical IP
<input type="checkbox"/>		<input checked="" type="checkbox"/>	C9F234F44C f6:61:d2:30:60:df	Diego Server	10.147.19.221	4 days	1.14.2	181.72.126.30
<input type="checkbox"/>		<input checked="" type="checkbox"/>	DC2BED6E68 f6:74:0b:e9:fa:fb	Vicente Cliente	10.147.19.187	1 minute	1.14.2	186.189.92.132

Figura 11: Cliente y servidor en la red simulada por ZeroTier

Una vez aceptados en la red, para comprobar que ambos se encuentran en la red local simulada y se pueden comunicar entre ambos, se realiza un ping a la dirección IPv4 asignada por ZeroTier del otro dispositivo.

ping <IPv4_asignada_por_ZeroTier>

```
ubuntu@ubuntu-VMware-Virtual-Platform:~$ ping 10.147.19.221
PING 10.147.19.221 (10.147.19.221) 56(84) bytes of data:
64 bytes from 10.147.19.221: icmp_seq=1 ttl=64 time=22.6 ms
64 bytes from 10.147.19.221: icmp_seq=2 ttl=64 time=17.8 ms
64 bytes from 10.147.19.221: icmp_seq=3 ttl=64 time=22.0 ms
64 bytes from 10.147.19.221: icmp_seq=4 ttl=64 time=16.7 ms
64 bytes from 10.147.19.221: icmp_seq=5 ttl=64 time=17.9 ms
64 bytes from 10.147.19.221: icmp_seq=6 ttl=64 time=17.5 ms
64 bytes from 10.147.19.221: icmp_seq=7 ttl=64 time=19.7 ms
^C
--- 10.147.19.221 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6009ms
rtt min/avg/max/mdev = 16.716/19.161/22.554/2.142 ms
```

Figura 12: Ping al servidor desde cliente.

Esto asegura que ambos equipos (cliente y servidor) están correctamente conectados a

través de la red privada virtual.

2. Crear contenedor en base a la imagen creada anteriormente

Desde el cliente, se debe ejecutar un contenedor Docker basado en la imagen personalizada **ffmpeg-custom**, montando un archivo de video para su posterior transmisión:

```
sudo docker run -it -rm \
--name ffmpeg-client \
-v /tmp/8sec.mp4:/video.mp4 \
ffmpeg-custom
```

Descripción:

- Se ejecuta un contenedor temporal llamado **ffmpeg-client** basado en la imagen **ffmpeg-custom** creada anteriormente.
- Se monta el archivo de video (**video.mp4**) dentro del contenedor desde la ruta **/tmp/**.



```
ubuntu@ubuntu-VMware-Virtual-Platform:~/ffmpeg-custom$ sudo docker run -i
t --rm \
--name ffmpeg-client \
-v /tmp/8sec.mp4:/video.mp4 \
ffmpeg-custom
root@77ff11ff5d2d:/#
```

Figura 13: Paso 2. Ejecutar contenedor temporal en base a la imagen *ffmpeg-custom*.

3. Enviar datos al servidor

Una vez creado el contenedor y accedido a este se debe realizar la conexión de cliente/servidor mediante el cliente **ffmpeg**:

```
ffmpeg -re -i video.mp4 -c copy -f rtsp \
rtsp://IP_SERVIDOR:8554/mystream
```

Donde:

- **-re:** Indica a **ffmpeg** que lea la entrada a la velocidad de reproducción nativa, simulando una transmisión en tiempo real.
- **-i video.mp4:** Especifica que archivo de entrada se transmite hacia el servidor. En este caso el archivo se llama **video.mp4**.
- **-c copy:** Copia los flujos de audio y video sin recodificarlos, lo que permite una transmisión más eficiente, siempre que el formato sea compatible con el formato de salida RTSP.

- **-f rtsp::** Establece el formato de salida como RTSP.
- **-rtsp://IP_SERVIDOR:8554/mystream:** Es la url que detalla hacia donde se enviará el contenido multimedia especificado, en este caso “IP_SERVIDOR” es (como especifica el nombre) la dirección IPv4 del servidor en la red. El puerto 8554 es el estándar utilizado comúnmente por el protocolo RTSP, y “/mystream” es la dirección del flujo que fue especificada en la configuración del servidor.

Al ejecutar este comando, ffmpeg mostrará en la terminal información técnica de la transmisión, como cantidad de cuadros enviados, duración, bitrate, y más métricas en tiempo real.

```

root@dba9f11ae728: /
  handler_name      : ISO Media file produced by Google Inc. Created on: 02/10/2024.
Output #0, rtsp, to 'rtsp://10.147.19.221:8554/mystream':
  Metadata:
    major_brand      : mp42
    minor_version    : 0
    compatible_brands: isommp42
    encoder          : Lavf58.29.100
  Stream #0:0(und): Video: h264 (Main) (avc1 / 0x31637661), yuv420p(tv, bt709), 640x360 [SAR 1:1 DAR 16:9], q=2-31, 21
3 kb/s, 30 fps, 30 tbr, 90k tbn, 30 tbc (default)
  Metadata:
    creation_time    : 2024-02-10T09:48:18.000000Z
    handler_name      : ISO Media file produced by Google Inc. Created on: 02/10/2024.
  Stream #0:1(und): Audio: aac (LC) (mp4a / 0x6134706D), 44100 Hz, stereo, fltp, 128 kb/s (default)
  Metadata:
    creation_time    : 2024-02-10T09:48:18.000000Z
    handler_name      : ISO Media file produced by Google Inc. Created on: 02/10/2024.
Stream mapping:
  Stream #0:0 -> #0:0 (copy)
  Stream #0:1 -> #0:1 (copy)
Press [q] to stop, [?] for help
frame= 23 fps=0.0 q=-1.0 size=N/A time=00:00:00.70 bitrate=N/A speed=1.frame= 38 fps= 37 q=-1.0 size=N/A time=00:00:
01.20 bitrate=N/A speed=1.frame= 53 fps= 35 q=-1.0 size=N/A time=00:00:01.70 bitrate=N/A speed=1.frame= 69 fps= 34 q
=-1.0 size=N/A time=00:00:02.23 bitrate=N/A speed=1.frame= 84 fps= 33 q=-1.0 size=N/A time=00:00:02.73 bitrate=N/A spe
ed=1.frame= 99 fps= 32 q=-1.0 size=N/A time=00:00:03.23 bitrate=N/A speed=1.frame= 114 fps= 32 q=-1.0 size=N/A time=0
0:00:03.73 bitrate=N/A speed=1.frame= 129 fps= 32 q=-1.0 size=N/A time=00:00:04.23 bitrate=N/A speed=1.frame= 145 fps=
32 q=-1.0 size=N/A time=00:00:04.76 bitrate=N/A speed=1.frame= 159 fps= 31 q=-1.0 size=N/A time=00:00:05.23 bitrate=N/
A speed=1.frame= 173 fps= 31 q=-1.0 size=N/A time=00:00:05.75 bitrate=N/A speed=1.frame= 187 fps= 31 q=-1.0 size=N/A t
ime=00:00:06.26 bitrate=N/A speed=1.frame= 205 fps= 31 q=-1.0 size=N/A time=00:00:06.76 bitrate=N/A speed=1.frame= 221
fps= 31 q=-1.0 size=N/A time=00:00:07.30 bitrate=N/A speed=1.frame= 235 fps= 31 q=-1.0 size=N/A time=00:00:07.77 bitra
te=N/A speed=1.frame= 235 fps= 31 q=-1.0 size=N/A time=00:00:07.82 bitrate=N/A speed=1.02x
video:204kB audio:123kB subtitle:0kB other streams:0kB global headers:0kB muxing overhead: unknown
root@dba9f11ae728: /

```

Figura 14: Paso 3. Envío del contenido multimedia al servidor.

2.6.5. Verificación de la transmisión

Para comprobar que la transmisión RTSP se realiza correctamente, se puede capturar y analizar el tráfico de red mediante Wireshark, filtrando por el protocolo RTSP. Esto permite observar el intercambio de paquetes entre cliente y servidor, asegurando que se estableció correctamente el flujo de datos.

No.	Time	Source	Destination	Protocol	Length	Info
70	1317.248391024	10.147.19.187	10.147.19.221	RTSP	157	OPTIONS rtsp://10.147.19.221:8554/mystream RTSP/1.0
72	1317.248596847	10.147.19.221	10.147.19.187	RTSP	194	Reply: RTSP/1.0 200 OK
76	1317.321883971	10.147.19.187	10.147.19.221	RTSP/S...	596	ANNOUNCE rtsp://10.147.19.221:8554/mystream RTSP/1.0[Malformed Packet]
78	1317.322368391	10.147.19.221	10.147.19.187	RTSP	113	Reply: RTSP/1.0 200 OK
79	1317.342133934	10.147.19.187	10.147.19.221	RTSP	234	SETUP rtsp://10.147.19.221:8554/mystream/streamid=0 RTSP/1.0
80	1317.342388596	10.147.19.221	10.147.19.187	RTSP	230	Reply: RTSP/1.0 200 OK
81	1317.362863085	10.147.19.187	10.147.19.221	RTSP	277	SETUP rtsp://10.147.19.221:8554/mystream/streamid=1 RTSP/1.0
82	1317.363020781	10.147.19.221	10.147.19.187	RTSP	230	Reply: RTSP/1.0 200 OK
83	1317.380855977	10.147.19.187	10.147.19.221	RTSP	218	RECORD rtsp://10.147.19.221:8554/mystream RTSP/1.0
88	1317.381360791	10.147.19.221	10.147.19.187	RTSP	156	Reply: RTSP/1.0 200 OK
489	1325.066633195	10.147.19.187	10.147.19.221	RTSP	201	TEARDOWN rtsp://10.147.19.221:8554/mystream RTSP/1.0
490	1325.066842720	10.147.19.221	10.147.19.187	RTSP	113	Reply: RTSP/1.0 200 OK

Figura 15: Visualización de la conexión en wireshark.

3. Capítulo II. Análisis de tráfico

Para llevar a cabo el análisis, se examinarán los diferentes paquetes que se generan durante la conexión entre cliente y servidor. Para ello, se utilizará Wireshark, una herramienta ampliamente utilizada en entornos educativos y profesionales para el análisis de protocolos de red. Esta aplicación permite capturar, filtrar y visualizar en tiempo real los paquetes de datos que circulan por una red, mostrando información detallada sobre cada uno de ellos, como las direcciones IP de origen y destino, los puertos utilizados, el tipo de protocolo, el tamaño del paquete y su contenido, tanto en formato hexadecimal como en texto legible.

Esta herramienta resulta especialmente útil para comprender el funcionamiento de la comunicación entre dispositivos dentro de una red, ya que permite observar paso a paso cómo se intercambian los mensajes. Además, incluye funciones de filtrado y seguimiento de flujos, lo que facilita centrarse en protocolos específicos o en conexiones determinadas.

En este análisis, se empleará Wireshark para capturar el tráfico generado al establecer una conexión entre un cliente y un servidor mediante el protocolo RTSP (Real Time Streaming Protocol). El objetivo será analizar en detalle cómo se transmiten los paquetes, identificar los mensajes de control propios de RTSP y comprender el flujo de comunicación durante una sesión de transmisión en tiempo real.

3.1. Detalle en el tráfico capturado

No.	Time	Source	Destination	Protocol	Length	Info
70	1317.248391024	10.147.19.187	10.147.19.221	RTSP	157	OPTIONS rtsp://10.147.19.221:8554/mystream RTSP/1.0
72	1317.248596847	10.147.19.221	10.147.19.187	RTSP	194	Reply: RTSP/1.0 200 OK
76	1317.321883971	10.147.19.187	10.147.19.221	RTSP/S...	596	ANNOUNCE rtsp://10.147.19.221:8554/mystream RTSP/1.0[Malformed Packet]
78	1317.322368391	10.147.19.221	10.147.19.187	RTSP	113	Reply: RTSP/1.0 200 OK
79	1317.342133934	10.147.19.187	10.147.19.221	RTSP	234	SETUP rtsp://10.147.19.221:8554/mystream/streamid=0 RTSP/1.0
80	1317.342388506	10.147.19.221	10.147.19.187	RTSP	230	Reply: RTSP/1.0 200 OK
81	1317.362803085	10.147.19.187	10.147.19.221	RTSP	277	SETUP rtsp://10.147.19.221:8554/mystream/streamid=1 RTSP/1.0
82	1317.363020781	10.147.19.221	10.147.19.187	RTSP	230	Reply: RTSP/1.0 200 OK
83	1317.380855977	10.147.19.187	10.147.19.221	RTSP	218	RECORD rtsp://10.147.19.221:8554/mystream RTSP/1.0
88	1317.381360791	10.147.19.221	10.147.19.187	RTSP	156	Reply: RTSP/1.0 200 OK
489	1325.066633195	10.147.19.187	10.147.19.221	RTSP	201	TEARDOWN rtsp://10.147.19.221:8554/mystream RTSP/1.0
490	1325.066842720	10.147.19.221	10.147.19.187	RTSP	113	Reply: RTSP/1.0 200 OK

Figura 16: Trafico entre servidor-cliente

Durante la captura de tráfico mostrada en la figura 15, se puede observar la secuencia de paquetes generados por el protocolo RTSP en el momento en que un cliente establece una conexión con un servidor. En esta sesión, el cliente envía un video de 8 segundos utilizando RTSP, y gracias a Wireshark, se pudieron identificar los distintos mensajes intercambiados. El proceso comenzó cuando el cliente envió un mensaje OPTIONS, utilizado para consultar qué métodos RTSP están disponibles en el servidor, como preguntando: “¿Qué puedo hacer contigo?”. El servidor respondió con una REPLY, indicando que acepta la solicitud y especificando los métodos habilitados.

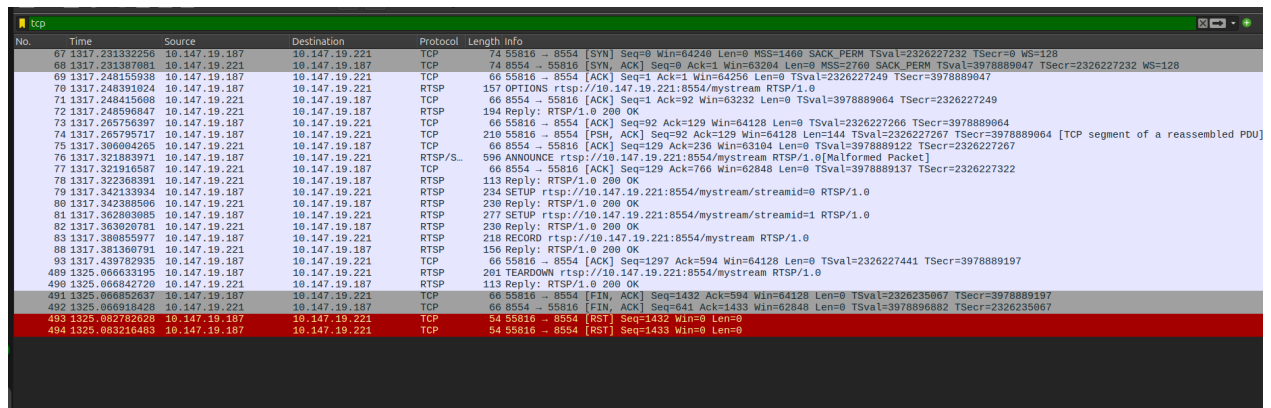
Posteriormente, el cliente envió un mensaje ANNOUNCE, donde comunicó información sobre el contenido que se transmitirá (en este caso, el video). Luego, llegó el mensaje SETUP, cuando el cliente solicitó establecer la configuración de la transmisión, incluyendo puertos y tipo de transporte. El servidor confirmó esta configuración con otra REPLY.

Una vez configurado todo, el cliente envió el mensaje RECORD, indicando el inicio de la transmisión del video. Durante los 8 segundos siguientes, los datos fluyeron conforme a lo pactado, hasta que finalmente se envió el mensaje TEARDOWN, que indica que la sesión ha terminado y que ya no es necesario mantener la conexión activa.

Esta secuencia de mensajes permitió establecer, configurar, ejecutar y cerrar de manera ordenada la sesión de transmisión, reflejando el funcionamiento completo del protocolo RTSP en una situación real.

3.1.1. Paquete de Conexión al servidor

Para analizar el paquete de conexión al servidor en Wireshark, es necesario aplicar un filtro por el protocolo TCP, ya que RTSP opera sobre este protocolo de transporte. Es decir, la conexión entre cliente y servidor es establecida realmente por TCP y no directamente por RTSP. Por esta razón, durante la captura en Wireshark se filtraron los paquetes usando el término "tcp", lo que permite visualizar únicamente aquellos paquetes relacionados con la conexión y transmisión de datos a través de TCP, facilitando así el análisis del inicio de la comunicación.



No.	Time	Source	Destination	Protocol	Length	Info
67	1317.231332556	10.147.19.187	10.147.19.221	TCP	74	55816 → 8554 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2326227232 TSecr=0 WS=128
68	1317.231387981	10.147.19.221	10.147.19.187	TCP	74	8554 → 55816 [SYN, ACK] Seq=0 Ack=1 Win=63204 Len=0 MSS=2760 SACK_PERM TSval=3978889047 TSecr=2326227232 WS=128
69	1317.248155938	10.147.19.187	10.147.19.221	TCP	66	55816 → 8554 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2326227249 TSecr=3978889047
70	1317.248391024	10.147.19.187	10.147.19.221	RTSP	157	OPTIONS rtsp://10.147.19.221:8554/mystream RTSP/1.0
71	1317.248415088	10.147.19.221	10.147.19.187	TCP	66	8554 → 55816 [ACK] Seq=1 Ack=92 Win=63232 Len=0 TSval=3978889064 TSecr=2326227249
72	1317.248590647	10.147.19.221	10.147.19.187	RTSP	194	Reply: RTSP/1.0 200 OK
73	1317.265756397	10.147.19.187	10.147.19.221	TCP	66	55816 → 8554 [ACK] Seq=92 Ack=129 Win=64128 Len=0 TSval=2326227266 TSecr=3978889064
74	1317.265795717	10.147.19.187	10.147.19.221	TCP	210	55816 → 8554 [PSH, ACK] Seq=92 Ack=129 Win=64128 Len=144 TSval=2326227267 TSecr=3978889064 [TCP segment of a reassembled PDU]
75	1317.306994265	10.147.19.221	10.147.19.187	TCP	66	8554 → 55816 [ACK] Seq=129 Ack=230 Win=63104 Len=0 TSval=3978889122 TSecr=2326227267
76	1317.321883971	10.147.19.187	10.147.19.221	RTSP/S.	596	ANNOUNCE rtsp://10.147.19.221:8554/mystream RTSP/1.0 [Malformed Packet]
77	1317.321916587	10.147.19.221	10.147.19.187	TCP	66	8554 → 55816 [ACK] Seq=129 Ack=766 Win=62848 Len=0 TSval=3978889137 TSecr=2326227322
78	1317.322368391	10.147.19.221	10.147.19.187	RTSP	113	Reply: RTSP/1.0 200 OK
79	1317.342133934	10.147.19.187	10.147.19.221	RTSP	234	SETUP rtsp://10.147.19.221:8554/mystream/streamid=0 RTSP/1.0
80	1317.342388506	10.147.19.221	10.147.19.187	RTSP	230	Reply: RTSP/1.0 200 OK
81	1317.362803985	10.147.19.187	10.147.19.221	RTSP	277	SETUP rtsp://10.147.19.221:8554/mystream/streamid=1 RTSP/1.0
82	1317.363826781	10.147.19.221	10.147.19.187	RTSP	230	Reply: RTSP/1.0 200 OK
83	1317.380855977	10.147.19.187	10.147.19.221	RTSP	218	RECORD rtsp://10.147.19.221:8554/mystream RTSP/1.0
88	1317.381360791	10.147.19.221	10.147.19.187	RTSP	156	Reply: RTSP/1.0 200 OK
93	1317.439782935	10.147.19.187	10.147.19.221	TCP	66	55816 → 8554 [ACK] Seq=1297 Ack=594 Win=64128 Len=0 TSval=2326227441 TSecr=3978889197
489	1325.066631195	10.147.19.187	10.147.19.221	RTSP	201	TEARDOWN rtsp://10.147.19.221:8554/mystream RTSP/1.0
490	1325.066842720	10.147.19.221	10.147.19.187	RTSP	113	Reply: RTSP/1.0 200 OK
491	1325.066852637	10.147.19.187	10.147.19.221	TCP	66	55816 → 8554 [FIN, ACK] Seq=1432 Ack=594 Win=64128 Len=0 TSval=2326235067 TSecr=3978889197
492	1325.066918428	10.147.19.221	10.147.19.187	TCP	66	8554 → 55816 [FIN, ACK] Seq=541 Ack=1433 Win=62848 Len=0 TSval=3978889582 TSecr=2326235067
493	1325.067056230	10.147.19.187	10.147.19.221	TCP	54	55816 → 8554 [RST] Seq=1432 Win=0 Len=0
494	1325.083216483	10.147.19.187	10.147.19.221	TCP	54	55816 → 8554 [RST] Seq=1433 Win=0 Len=0

Figura 17: Paquetes del filtro "tcp"

Como se puede observar en la figura 16, al aplicar el filtro "tcp" en Wireshark, aparece un listado de paquetes correspondientes al proceso de conexión gestionado por el protocolo TCP. Entre estos paquetes se identifican claramente los mensajes utilizados para establecer la conexión, como el SYN (synchronize) y el SYN-ACK (synchronize-acknowledge). Este intercambio forma parte del denominado three-way handshake, un proceso de tres pasos que permite a TCP crear una conexión confiable entre cliente y servidor antes de iniciar la transferencia de datos.

En este caso, dicho proceso es fundamental porque RTSP depende de TCP para transmitir sus mensajes. Por lo tanto, antes de observar cualquier paquete RTSP, primero deben aparecer estos paquetes correspondientes al establecimiento de la conexión TCP.

Además, al finalizar la comunicación, TCP también se encarga de cerrar la conexión de manera ordenada. Esto se refleja en los paquetes FIN (finish) y ACK (acknowledge), que indican que una de las partes desea terminar la sesión y la otra lo reconoce. Este mecanismo garantiza que todos los datos hayan sido transmitidos correctamente antes de cerrar la conexión.

Gracias a esta estructura del protocolo TCP, es posible mantener una transmisión segura y confiable, lo cual resulta especialmente importante cuando se trabaja con protocolos de control como RTSP, donde la integridad de los mensajes es clave para que la transmisión de video funcione correctamente.

3.2. Suposiciones

En caso de que se modifique el tráfico relacionado con el protocolo RTSP, podrían presentarse diversas repercusiones en el funcionamiento del software que implementa este protocolo. Una posible consecuencia es la pérdida del control sobre la sesión de transmisión, ya que RTSP utiliza mensajes específicos como SETUP, RECORD, ANNOUNCE y TEARDOWN para gestionar cada etapa de la comunicación entre el cliente y el servidor. Por ejemplo, si un mensaje SETUP es alterado o no llega correctamente, el servidor podría no configurar adecuadamente los parámetros de la transmisión, lo que impediría que el video se inicie o se reproduzca de forma correcta. De manera similar, si un mensaje RECORD se ve modificado, el servidor podría no comenzar a grabar o transmitir el contenido según lo esperado. Además, cualquier manipulación del tráfico durante la transmisión puede provocar errores como desincronización, comportamientos inesperados del reproductor o incluso bloqueos y cierres forzados del software. En resumen, cualquier alteración en los mensajes RTSP puede desestabilizar el funcionamiento lógico del sistema, afectando directamente la experiencia del usuario y la integridad del proceso de streaming.

4. Conclusión Capítulo II

La realización de esta tarea permitió comprender en profundidad el funcionamiento del protocolo RTSP (Real-Time Streaming Protocol), desde su origen y evolución hasta su implementación práctica en un entorno controlado. A través del uso de herramientas como Docker, MediaMTX y FFmpeg, se logró establecer una arquitectura cliente-servidor capaz de transmitir contenido multimedia en tiempo real, lo que permitió evidenciar el rol específico de cada comando RTSP en el ciclo de vida de una sesión de transmisión.

Asimismo, el uso de Wireshark para el análisis de tráfico de red fue clave para visualizar y verificar el correcto intercambio de mensajes, tanto a nivel del protocolo RTSP como de los protocolos subyacentes, como TCP. Esto no solo permitió comprobar técnicamente la integridad de la transmisión, sino también entender el valor de herramientas de inspección de red en contextos reales de monitoreo y diagnóstico.

Además, se pudo observar cómo una manipulación en el flujo de datos puede afectar directamente la estabilidad del sistema de streaming, generando consecuencias como fallas en la conexión, errores en la reproducción, o pérdida de control sobre la sesión. Esto refuerza la importancia de la seguridad, la sincronización y el diseño correcto en aplicaciones que dependen de transmisiones en tiempo real.

En resumen, esta experiencia no solo fortaleció el entendimiento del protocolo RTSP y su integración con otras herramientas tecnológicas, sino que también brindó una visión práctica y crítica sobre su funcionamiento, su utilidad en entornos reales como la videovigilancia y las transmisiones en vivo, y los desafíos que implica garantizar su correcto comportamiento en redes modernas.

5. Capítulo III. Modificación de paquetes con Scapy.

5.1. Introducción Capítulo III

Scapy es una herramienta desarrollada en Python que permite crear, enviar, capturar, analizar y modificar paquetes de red en múltiples protocolos, como IP, TCP, UDP, ICMP, entre otros. Está orientada a tareas como pruebas de penetración, análisis de tráfico (sniffing), fuzzing de protocolos y automatización de pruebas de seguridad. Es importante destacar que Scapy, por sí sola, no representa un riesgo, pero puede ser utilizada con fines maliciosos que comprometan la integridad o disponibilidad de una red.

En el contexto del protocolo RTSP (Real-Time Streaming Protocol), Scapy será utilizada para interceptar el tráfico entre cliente y servidor, identificando los paquetes que establecen y mantienen la conexión. A través de técnicas de fuzzing, se inyectarán paquetes con datos erróneos o malformados con el objetivo de probar la robustez del protocolo. Posteriormente, se realizarán modificaciones en tiempo real sobre los paquetes interceptados, con el fin de analizar el comportamiento del protocolo frente a dichas alteraciones y evaluar posibles vulnerabilidades.

5.2. Desarrollo

Toda la información sobre la instalación de las herramientas, se encuentra en el siguiente repositorio:

- <https://github.com/diegofranco1/Tarea3-Redes-Scapy>

5.2.1. Configuración Scapy con Docker

Primero, es necesario instalar y configurar correctamente la herramienta para garantizar que la actividad se desarrolle de forma adecuada y sin inconvenientes. Para ello, se implementará un contenedor Docker que integre Scapy, lo que permitirá un entorno de ejecución aislado, estable y sin interrupciones, facilitando así su uso durante todo el proceso de análisis.

1. Crear carpeta que almacenará la imagen personalizada

Primero se debe crear y acceder a la carpeta en la que se guardará la imagen que se creará para la herramienta Scapy:

```
mkdir scapy-custom
cd scapy-custom
```

2. Crear archivo llamado Dockerfile y acceder al mismo

Dentro de la carpeta se crea un archivo vacío llamado Dockerfile, que contendrá las instrucciones de la construcción de la imagen:

```
touch Dockerfile
nano Dockerfile
```

3. Instrucciones para el Dockerfile

Dentro del archivo, se deben escribir las siguientes instrucciones:

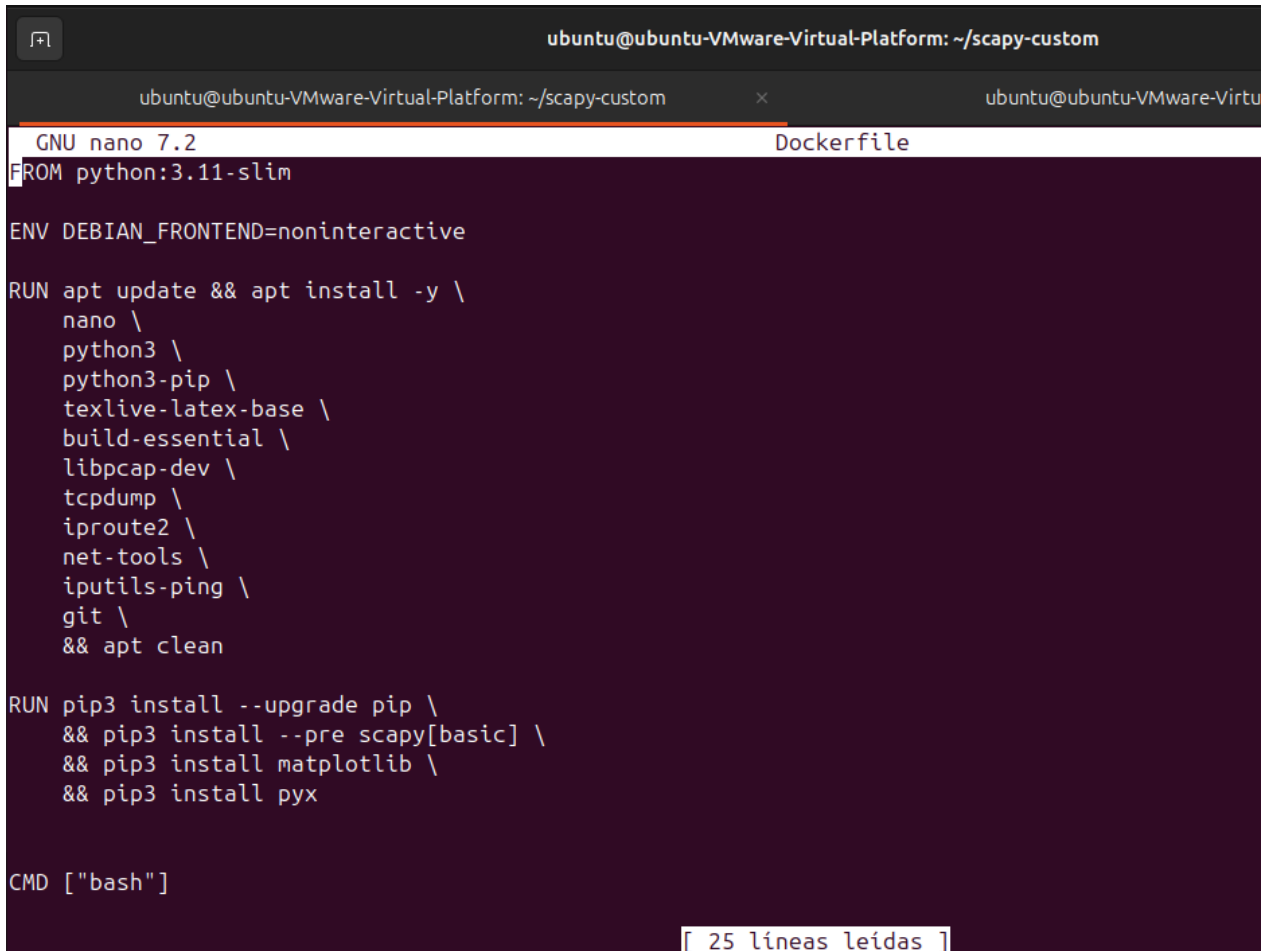
```
FROM python:3.11-slim

ENV DEBIAN_FRONTEND=noninteractive

RUN apt update && apt install -y \
    nano \
    python3 \
    python3-pip \
    texlive-latex-base \
    build-essential \
    libpcap-dev \
    tcpdump \
    iproute2 \
    net-tools \
    iputils-ping \
    git \
    && apt clean

RUN pip3 install --upgrade pip \
    && pip3 install --pre scapy[basic] \
    && pip3 install matplotlib \
    && pip3 install pyx

CMD ["bash"]
```



```

GNU nano 7.2 Dockerfile
FROM python:3.11-slim

ENV DEBIAN_FRONTEND=noninteractive

RUN apt update && apt install -y \
    nano \
    python3 \
    python3-pip \
    texlive-latex-base \
    build-essential \
    libpcap-dev \
    tcpdump \
    iproute2 \
    net-tools \
    iputils-ping \
    git \
    && apt clean

RUN pip3 install --upgrade pip \
    && pip3 install --pre scapy[basic] \
    && pip3 install matplotlib \
    && pip3 install pyx

CMD ["bash"]
[ 25 líneas leídas ]

```

Figura 18: Conjunto de instrucciones del Dockerfile

4. Crear imagen personalizada

Con el Dockerfile listo, se crea la imagen personalizada bajo el siguiente comando:

```
docker build -t scapy-t3 scapy-custom/
```

Esto creará una imagen personalizada llamada “scapy-t3” con las instrucciones descritas dentro del Dockerfile.


```
ubuntu@ubuntu-VMware-Virtual-Platform:~$ sudo docker build -t scapy-env scapy-custom/
[+] Building 170.2s (7/7) FINISHED                                docker:default
=> [internal] load build definition from Dockerfile                0.0s
=> => transferring dockerfile: 468B                               0.0s
=> [internal] load metadata for docker.io/library/python:3.11-slim 3.1s
=> [internal] load .dockerignore                                  0.0s
=> => transferring context: 2B                                     0.0s
=> [1/3] FROM docker.io/library/python:3.11-slim@sha256:9e1912aab0a30bbd9488eb79063f68f42a68ab0946cbe98fecf197fe 7.6s
=> => resolve docker.io/library/python:3.11-slim@sha256:9e1912aab0a30bbd9488eb79063f68f42a68ab0946cbe98fecf197fe 0.0s
=> => sha256:9e1912aab0a30bbd9488eb79063f68f42a68ab0946cbe98fecf197fe5b085506 9.13kB / 9.13kB 0.0s
=> => sha256:cfa2a40862158178855ab4f7cf6b9341646f826b0467a7b72bdeac68b03986bb 1.75kB / 1.75kB 0.0s
=> => sha256:be3324b8ee1a17161c5fa4a20f310d4af42cbb4f22a1e7a32a98ee9196a6defd 5.37kB / 5.37kB 0.0s
=> => sha256:dad67da3f26bce15939543965e09c4059533b025f707aad72ed3d3f3a09c66f8 28.23MB / 28.23MB 1.5s
=> => sha256:799440a7bae7c08a5fe9d9e5a1ccd72fc3cbf9d85fa4be450e12b8550175c620 3.51MB / 3.51MB 1.9s
=> => sha256:9596beeb5a6dc0950529870568799000e8d73fb678969ac2f485005cd5da1087 16.21MB / 16.21MB 2.2s
=> => sha256:15658014cd85cd0d8b913d50b4388228aebcf0437d43cfb37e8a5177e8b2bcf8 248B / 248B 2.0s
=> => extracting sha256:dad67da3f26bce15939543965e09c4059533b025f707aad72ed3d3f3a09c66f8 3.6s
=> => extracting sha256:799440a7bae7c08a5fe9d9e5a1ccd72fc3cbf9d85fa4be450e12b8550175c620 0.3s
=> => extracting sha256:9596beeb5a6dc0950529870568799000e8d73fb678969ac2f485005cd5da1087 1.7s
=> => extracting sha256:15658014cd85cd0d8b913d50b4388228aebcf0437d43cfb37e8a5177e8b2bcf8 0.0s
=> [2/3] RUN apt update && apt install -y python3 python3-pip texlive-latex-base build-essenti 121.2s
=> [3/3] RUN pip3 install --upgrade pip && pip3 install --pre scapy[basic] && pip3 install matplotlib 28.3s
=> exporting to image                                           9.7s
=> => exporting layers                                           9.7s
=> => writing image sha256:e4ba435bd5a7b00a57dbf406b903a45ff3d75926ba3cf0ebf9f2b1bf85c3284f 0.0s
=> => naming to docker.io/library/scapy-env                      0.0s
ubuntu@ubuntu-VMware-Virtual-Platform:~$
```

Figura 19: Creación del contenedor

5. Crear contenedor en base a la imagen personalizada

Desde la maquina que efectuará la modificacion de paquetes con scapy, se debe ejecutar un contenedor Docker basado en la imagen personalizada scapy-t3, permitiendo acceso a la red.

```
sudo docker run --rm -it \
--net=host \
--cap-add=NET_ADMIN \
--cap-add=NET_RAW \
--device /dev/net/tun \
--name scapy-tarea3 \
scapy-t3
```

```
ubuntu@ubuntu-VMware-Virtual-Platform:~$ sudo docker run --rm -it \
--net=host \
--cap-add=NET_ADMIN \
--cap-add=NET_RAW \
--device /dev/net/tun \
--name scapy-tarea3 \
scapy-t3
[sudo] contraseña para ubuntu:
root@ubuntu-VMware-Virtual-Platform:/#
```

Figura 20: Ejecución del contenedor

5.2.2. Intercepción de tráfico

Una vez que el contenedor ha sido creado y configurado correctamente para garantizar el funcionamiento óptimo de la herramienta Scapy, se procede a interceptar paquetes de red con el objetivo de verificar la presencia y actividad dentro de la conexión cliente-servidor. Para ello, se desarrollará un script en Python que permitirá capturar y visualizar en tiempo real los paquetes transmitidos, facilitando el análisis del tráfico RTSP y asegurando que la herramienta esté operando dentro del entorno de red deseado.

1. Crear script python

Dentro del contenedor, se debe crear un archivo **.py**, que contendrá el script python para utilizar de sniffer en la red. Para ello se crea el archivo **interceptar-rtsp.py** con el contenido del script

```
nano interceptar-rtsp.py
```

Script:

```
from scapy.all import *

INTERFAZ_VPN = "ztrf23gpht"
PUERTO_RTSP = 8554

def ver_paquetes(pkt):
    if pkt.haslayer(TCP) and pkt.haslayer(Raw):
        carga = pkt[Raw].load
        if b"RTSP" in carga or b"OPTIONS" in carga or b"DESCRIBE"
            in carga or b"SETUP" in carga or b"PLAY" in carga:
```

```

        print("\nPaquete RTSP detectado:")
        try:
            print(carga.decode(errors='ignore'))
        except:
            print("Paquete no decodificable")

print(f"Escuchando trafico RTSP TCP en interfaz '{INTERFAZ_VPN}'
puerto {PUERTO_RTSP}...")
sniff(
    iface=INTERFAZ_VPN,
    filter=f"tcp port {PUERTO_RTSP}",
    prn=ver_paquetes,
    store=0
)

```

Descripción: Este código se conecta a la interfaz de red donde se encuentran el cliente y el servidor (en este caso, la interfaz correspondiente a la red VPN) y al puerto en el que opera la conexión RTSP (configurado en el puerto 8554). Filtra únicamente los paquetes TCP dirigidos a dicho puerto, y si detecta que alguno contiene comandos del protocolo RTSP, imprime su contenido en la terminal. Esto permite monitorear en tiempo real la comunicación entre el cliente y el servidor RTSP, facilitando la visualización de los paquetes intercambiados entre cliente y servidor.

2. Ejecución del script

Una vez creado el script, se debe ejecutar para interceptar el tráfico RTSP en la red.

```
python3 interceptar-rtsp.py
```

```

root@ubuntu-VMware-Virtual-Platform:/# python3 interceptar-rtsp.py
Escuchando tráfico RTSP TCP en interfaz 'ztrf23gpht' puerto 8554...

Paquete RTSP detectado:
OPTIONS rtsp://10.147.19.221:8554/mystream RTSP/1.0
CSeq: 1
User-Agent: Lavf58.29.100

Paquete RTSP detectado:
RTSP/1.0 200 OK
CSeq: 1
Public: DESCRIBE, ANNOUNCE, SETUP, PLAY, RECORD, PAUSE, GET_PARAMETER, TEARDOWN
Server: gortsplib

Paquete RTSP detectado:
ANNOUNCE rtsp://10.147.19.221:8554/mystream RTSP/1.0
Content-Type: application/sdp
CSeq: 2
User-Agent: Lavf58.29.100
Content-Length: 530

```

Figura 21: Intercepción de tráfico RTSP

5.2.3. Modificación de tráfico

Una vez verificado que estamos en la red y que el sniffer detecta paquetes de tráfico cliente-servidor, se procede a realizar modificaciones de los mismos durante la conexión para así analizar su comportamiento.

1.

Crear script python

Nuevamente, dentro del contenedor se debe crear un archivo **.py** que contendrá el script que permitira la modificación del contenido de los paquetes. Para ello se crea el archivo **mod-t3.py** con el contenido del script

```
nano mod-t3.py
```

Script:

```

from scapy.all import *

IP_SERVIDOR = "10.147.19.221"
PUERTO_RTSP = 8554
INTERFAZ_VPN = "ztrf23gpht"

```

```

print(f"Escuchando paquetes RTSP TCP en interfaz '{INTERFAZ_VPN}'
para modificar y reenviar...")

modificados = 0
LIMITE = 3

def modificar_y_reenviar(pkt):
    global modificados

    if modificados >= LIMITE:
    return

    if pkt.haslayer(TCP) and pkt.haslayer(Raw):
        original = pkt[Raw].load.decode(errors='ignore')

        if original.startswith(("OPTIONS", "ANNOUNCE", "SETUP",
                                "DESCRIBE", "PLAY")):

            print("\n Paquete original interceptado:")
            print(original)

            modificado = re.sub(r"CSeq:\s*\d+", "CSeq: 9999",
                                original)
            modificado = re.sub(r"User-Agent:.*", "User-Agent:
EvilFaker/9.9", modificado)

            print("\n Paquete modificado:")
            print(modificado)

            paquete = IP(dst=IP_SERVIDOR)/TCP(dport=PUERTO_RTSP,
            sport=RandShort(), flags="PA")/Raw(load=modificado.encode())
            send(paquete)

            print(" Paquete reenviado con modificaciones.")
            modificados += 1

sniff(
    iface=INTERFAZ_VPN,
    filter=f"tcp port {PUERTO_RTSP}",
    prn=modificar_y_reenviar,
    store=0
)

```

Descripción: Este código tiene como objetivo interceptar paquetes 3 RTSP, modificar campos clave específicamente CSeq y User-Agent y reenviarlos al servidor con información alterada, simulando así el comportamiento de un cliente malicioso o erróneo. Para ello, el script configura la interfaz de red correspondiente (en este caso, la interfaz de la VPN), y establece un límite de modificaciones para evitar bucles o sobrecarga innecesaria en la red. El filtrado se aplica únicamente a paquetes TCP dirigidos al puerto RTSP, que contengan instrucciones comunes del protocolo. Una vez interceptados, los paquetes son modificados:

- El campo CSeq se reemplaza por un valor fuera de secuencia (9999).
- El campo User-Agent se falsifica para simular un cliente no legítimo (EvilFaker/9.9).

Estas modificaciones permiten analizar la respuesta del servidor ante tráfico malformado o no esperado. En particular, se espera que el servidor rechace las solicitudes, registre errores de sesión o finalice la conexión si detecta inconsistencias en la secuencia o el origen del cliente.

2. Ejecución del script

Una vez creado el script, se debe ejecutar para interceptar y modificar el tráfico RTSP en la red.

```
python3 mod-t3.py
```

```
root@ubuntu-VMware-Virtual-Platform:/# python3 mod-t3.py
Escuchando paquetes RTSP TCP en interfaz 'ztrf23gpht' para modificar y reenviar...

Paquete original interceptado:
OPTIONS rtsp://10.147.19.221:8554/mystream RTSP/1.0
CSeq: 1
User-Agent: Lavf58.29.100

Paquete modificado:
OPTIONS rtsp://10.147.19.221:8554/mystream RTSP/1.0
CSeq: 9999
User-Agent: EvilFaker/9.9
.
Sent 1 packets.
```

Figura 22: Modificación de paquetes con Scapy

5.2.4. Hipótesis modificación de tráfico

Previo al planteamiento de una hipótesis sobre el comportamiento del protocolo RTSP frente a la recepción de paquetes modificados, es fundamental comprender qué campos pueden ser alterados dentro de un paquete RTSP. Esta comprensión permite identificar los elementos clave del protocolo susceptibles de manipulación.

- **Método:** Comando RTSP como OPTIONS, DESCRIBE, SETUP, PLAY, etc.
- **Request URI:** Dirección del recurso solicitado, como rtsp://ip:puerto/stream.
- **Versión:** Versión del protocolo, usualmente RTSP/1.0.
- **Cseq:** Número de secuencia que identifica cada solicitud.
- **Transport:** Define el tipo de transporte (UDP, TCP) y puertos usados.
- **Session:** Identificador de sesión RTSP, requerido después del SETUP.
- **User-agent:** Identificación del cliente RTSP (ej. Lavf58.29.100).
- **Content-length:** Tamaño del cuerpo del mensaje (usado en mensajes con cuerpo, como ANNOUNCE).
- **Content-type:** Tipo de contenido del cuerpo, como application/sdp.
- **Authorization:** Cabecera usada si el servidor requiere autenticación.
- **Range:** Especifica el intervalo de tiempo para reproducir.
- **Scale:** Velocidad de reproducción (ej. Scale: 1.0 para velocidad normal).

Con esto en mente, nos enfocaremos en los campos **Cseq** y **User-agent**, que fueron los campos modificados en este caso.

1. Campo Cseq

El campo CSeq en RTSP representa el número de secuencia de cada solicitud y debe incrementarse en uno con cada mensaje enviado por el cliente. El servidor utiliza este valor para mantener la coherencia del flujo de comunicación. Si se modifica CSeq a un valor anómalo o fuera de secuencia como en este caso, donde se establece en 9999, se espera que el servidor:

- Rechace o ignore la solicitud si no coincide con el valor esperado.
- Registre un error de sincronización y potencialmente corte la conexión.
- O, si está configurado de manera permisiva y no depende estrictamente de CSeq, acepte la solicitud sin consecuencias visibles.

2. Campo User-agent

El campo User-Agent en RTSP identifica el software del cliente que realiza la solicitud; por ejemplo, Lavf58.29.100 corresponde al cliente FFmpeg. Aunque este campo es opcional según la especificación, algunos servidores lo utilizan para recopilar estadísticas o aplicar filtros específicos. En este caso, al modificar su valor a EvilFaker/9.9, se espera que el servidor:

- Registre al cliente con un identificador falso en los logs.
- O bien, rechace la solicitud si dispone de un sistema de filtrado basado en el contenido del User-Agent.

En conclusión, al modificar estos campos, se espera que el servidor adopte un comportamiento distinto al previsto, como rechazar la solicitud, finalizar la sesión RTSP o impedir el inicio correcto del stream, especialmente si su configuración está orientada a la seguridad o integridad de la sesión.

5.3. Análisis modificación del tráfico con Scapy

Para realizar el análisis del tráfico, se utilizó la herramienta Wireshark, con el objetivo de examinar los paquetes modificados y determinar el tipo de respuesta que enviaba el servidor. A partir de esta revisión, se obtuvo el siguiente resultado.

No.	Time	Source	Destination	Protocol	Length	Info
12	88.266406237	10.147.19.187	10.147.19.221	RTSP	157	OPTIONS rtsp://10.147.19.221:8554/mystream RTSP/1.0
14	88.26669827	10.147.19.221	10.147.19.187	RTSP	194	Reply: RTSP/1.0 200 OK
18	88.342286125	10.147.19.187	10.147.19.221	RTSP/S...	596	ANNOUNCE rtsp://10.147.19.221:8554/mystream RTSP/1.0 [Malformed Packet]
20	88.343519570	10.147.19.221	10.147.19.187	RTSP	113	Reply: RTSP/1.0 200 OK
21	88.368771774	10.147.19.187	10.147.19.221	RTSP	234	SETUP rtsp://10.147.19.221:8554/mystream/streamid=0 RTSP/1.0
22	88.368907415	10.147.19.221	10.147.19.187	RTSP	238	Reply: RTSP/1.0 200 OK
23	88.382833677	10.147.19.187	10.147.19.221	RTSP	277	SETUP rtsp://10.147.19.221:8554/mystream/streamid=1 RTSP/1.0
24	88.392214353	10.147.19.221	10.147.19.187	RTSP	238	Reply: RTSP/1.0 200 OK
25	88.417579699	10.147.19.187	10.147.19.221	RTSP	218	RECORD rtsp://10.147.19.221:8554/mystream RTSP/1.0
30	88.418118664	10.147.19.221	10.147.19.187	RTSP	156	Reply: RTSP/1.0 200 OK
433	87.937906455	10.147.19.187	10.147.19.221	RTSP	201	TEARDOWN rtsp://10.147.19.221:8554/mystream RTSP/1.0
435	87.938173314	10.147.19.221	10.147.19.187	RTSP	113	Reply: RTSP/1.0 200 OK
446	215.884826348	10.147.19.187	10.147.19.221	RTSP	157	OPTIONS rtsp://10.147.19.221:8554/mystream RTSP/1.0
448	215.885139510	10.147.19.221	10.147.19.187	RTSP	194	Reply: RTSP/1.0 200 OK
453	215.852178521	10.147.19.187	10.147.19.221	RTSP	147	OPTIONS rtsp://10.147.19.221:8554/mystream RTSP/1.0
458	215.880884974	10.147.19.187	10.147.19.221	RTSP/S...	596	ANNOUNCE rtsp://10.147.19.221:8554/mystream RTSP/1.0 [Malformed Packet]
460	215.881308311	10.147.19.221	10.147.19.187	RTSP	113	Reply: RTSP/1.0 200 OK
461	215.887818799	10.147.19.187	10.147.19.221	RTSP	147	OPTIONS rtsp://10.147.19.221:8554/mystream RTSP/1.0
463	215.901832342	10.147.19.187	10.147.19.221	RTSP	234	SETUP rtsp://10.147.19.221:8554/mystream/streamid=0 RTSP/1.0
464	215.902061837	10.147.19.221	10.147.19.187	RTSP	238	Reply: RTSP/1.0 200 OK
465	215.920355037	10.147.19.187	10.147.19.221	RTSP	277	SETUP rtsp://10.147.19.221:8554/mystream/streamid=1 RTSP/1.0
466	215.920525447	10.147.19.221	10.147.19.187	RTSP	238	Reply: RTSP/1.0 200 OK
467	215.935953949	10.147.19.187	10.147.19.221	RTSP	218	RECORD rtsp://10.147.19.221:8554/mystream RTSP/1.0
472	215.936478169	10.147.19.221	10.147.19.187	RTSP	156	Reply: RTSP/1.0 200 OK
873	223.613223158	10.147.19.187	10.147.19.221	RTSP	201	TEARDOWN rtsp://10.147.19.221:8554/mystream RTSP/1.0
875	223.613397759	10.147.19.221	10.147.19.187	RTSP	113	Reply: RTSP/1.0 200 OK

Figura 23: Paquetes generados

A continuación se presentan todos los paquetes generados durante el proceso de modificación. Es importante destacar que únicamente tres de ellos fueron alterados. En lo que sigue, se analizarán en detalle estos paquetes modificados, describiendo qué ocurrió en cada caso y comparando su contenido con el esperado.

5.3.1. Primer paquete modificado

No.	Time	Source	Destination	Protocol	Length	Info
12	80.266480237	10.147.19.187	10.147.19.221	RTSP	157	OPTIONS rtsp://10.147.19.221:8554/mystream RTSP/1.0
14	80.26669827	10.147.19.221	10.147.19.187	RTSP	194	Reply: RTSP/1.0 200 OK
18	80.332206125	10.147.19.187	10.147.19.221	RTSP/S	596	ANNOUNCE rtsp://10.147.19.221:8554/mystream RTSP/1.0 [Malformed Packet]
20	80.343519570	10.147.19.221	10.147.19.187	RTSP	113	Reply: RTSP/1.0 200 OK
21	80.368771774	10.147.19.187	10.147.19.221	RTSP	234	SETUP rtsp://10.147.19.221:8554/mystream/streamid=0 RTSP/1.0
22	80.369007415	10.147.19.221	10.147.19.187	RTSP	230	Reply: RTSP/1.0 200 OK
23	80.392833677	10.147.19.187	10.147.19.221	RTSP	277	SETUP rtsp://10.147.19.221:8554/mystream/streamid=1 RTSP/1.0
24	80.392214353	10.147.19.221	10.147.19.187	RTSP	230	Reply: RTSP/1.0 200 OK
25	80.417579699	10.147.19.187	10.147.19.221	RTSP	218	RECORD rtsp://10.147.19.221:8554/mystream RTSP/1.0
26	80.418111005	10.147.19.221	10.147.19.187	RTSP	153	Reply: RTSP/1.0 200 OK
433	87.937986455	10.147.19.187	10.147.19.221	RTSP	261	TEARDOWN rtsp://10.147.19.221:8554/mystream RTSP/1.0


```

* Frame 18: 596 bytes on wire (4768 bits), 596 bytes captured (4768 bits) on interface ztrf23gphrt, id 0
* Ethernet II, Src: f6:74:0b:e9:fa:fb (f6:74:0b:e9:fa:fb), Dst: f6:61:d2:30:60:df (f6:61:d2:30:60:df)
* Internet Protocol Version 4, Src: 10.147.19.187, Dst: 10.147.19.221
* Transmission Control Protocol, Src Port: 49712, Dst Port: 8554, Seq: 236, Ack: 129, Len: 530
* [2 Reassembled TCP Segments (674 bytes): #10(144), #10(530)]
* Real Time Streaming Protocol
  * Request: ANNOUNCE rtsp://10.147.19.221:8554/mystream RTSP/1.0\r\n
    Method: ANNOUNCE
    URL: rtsp://10.147.19.221:8554/mystream
    Content-type: application/sdp
    CSeq: 2\r\n
    User-Agent: Lavf58.29.100\r\n
    Content-length: 530
    \r\n
  * Session Description Protocol

```

Figura 24: Primer paquete modificado

En la imagen se muestra el primer paquete que se intentó modificar, correspondiente a una solicitud ANNOUNCE del protocolo RTSP. Sin embargo, como se puede observar, el paquete fue clasificado por Wireshark como malformed packet (paquete malformado). Esto indica que el paquete no cumplía con la estructura esperada por el protocolo, lo que impidió su correcta interpretación.

Este comportamiento puede deberse a que, durante el proceso de modificación con Scapy, el paquete fue alterado de forma que rompió su formato interno o sus encabezados, impidiendo su correcta reconstrucción por parte del receptor o incluso por herramientas de análisis como Wireshark. Otra posibilidad es que el servidor o el sistema operativo empleen mecanismos de verificación o integridad sobre ciertos paquetes RTSP, como el ANNOUNCE, lo que hace que al detectar una modificación no válida, el paquete se rechace o se envíe incompleto.

En este caso, es probable que el paquete fuera transmitido antes de completar su modificación o que el sistema impidiera su reenvío correcto, resultando en una captura que muestra un paquete parcialmente armado o inválido. Esto sugiere que el protocolo RTSP —o las implementaciones involucradas— podría contar con medidas que dificultan la manipulación directa de ciertos mensajes, especialmente aquellos clave como ANNOUNCE.

5.3.2. Segundo Paquete modificado

No.	Time	Source	Destination	Protocol	Length	Info
30	80.418118064	10.147.19.221	10.147.19.187	RTSP	156	Reply: RTSP/1.0 200 OK
433	87.93796455	10.147.19.187	10.147.19.221	RTSP	261	TEARDOWN rtsp://10.147.19.221:8554/mystream RTSP/1.0
435	87.938173314	10.147.19.221	10.147.19.187	RTSP	113	Reply: RTSP/1.0 200 OK
446	215.804826348	10.147.19.187	10.147.19.221	RTSP	157	OPTIONS rtsp://10.147.19.221:8554/mystream RTSP/1.0
448	215.805139510	10.147.19.221	10.147.19.187	RTSP	194	Reply: RTSP/1.0 200 OK
453	215.852178521	10.147.19.187	10.147.19.221	RTSP	147	OPTIONS rtsp://10.147.19.221:8554/mystream RTSP/1.0
458	215.880849474	10.147.19.187	10.147.19.221	RTSP/S.	596	ANNOUNCE rtsp://10.147.19.221:8554/mystream RTSP/1.0 [Malformed Packet]
460	215.881308311	10.147.19.221	10.147.19.187	RTSP	113	Reply: RTSP/1.0 200 OK
461	215.887818799	10.147.19.187	10.147.19.221	RTSP	147	OPTIONS rtsp://10.147.19.221:8554/mystream RTSP/1.0
463	215.901832342	10.147.19.187	10.147.19.221	RTSP	234	SETUP rtsp://10.147.19.221:8554/mystream/streamid=0 RTSP/1.0
464	215.902861837	10.147.19.221	10.147.19.187	RTSP	238	Reply: RTSP/1.0 200 OK

```

Frame 453: 147 bytes on wire (1176 bits), 147 bytes captured (1176 bits) on interface ztrf23gghit, id 0
  Ethernet II, Src: fe:74:0b:e9:fa:fb (fe:74:0b:e9:fa:fb), Dst: fe:61:d2:30:60:df (fe:61:d2:30:60:df)
  Internet Protocol Version 4, Src: 10.147.19.187, Dst: 10.147.19.221
  Transmission Control Protocol, Src Port: 63127, Dst Port: 8554, Seq: 1, Ack: 1, Len: 93
  Real Time Streaming Protocol
    Request: OPTIONS rtsp://10.147.19.221:8554/mystream RTSP/1.0\r\n
      Method: OPTIONS
      URL: rtsp://10.147.19.221:8554/mystream
      CSeq: 9999\r\n
      User-Agent: EvilFaker/9.9\r\n
    \r\n
  
```

Figura 25: Primer paquete modificado

El segundo paquete modificado corresponde a una solicitud OPTIONS del protocolo RTSP. A diferencia del caso anterior, en este paquete la modificación se realizó correctamente. Como se observa en la imagen, el campo User-Agent fue alterado y muestra el valor personalizado EvilFaker/9.9, lo que indica que el paquete fue enviado con éxito y reconocido por el servidor, sin ser descartado ni marcado como malformado.

Este resultado sugiere que las solicitudes OPTIONS en RTSP no están sujetas a validaciones tan estrictas como otras solicitudes más críticas —como ANNOUNCE— y, por lo tanto, permiten cierto grado de manipulación sin comprometer la estructura del paquete ni su aceptación por el servidor. En consecuencia, se confirma que la modificación de este segundo paquete fue exitosa y el contenido alterado fue transmitido e interpretado correctamente.

5.3.3. Tercer Paquete modificado

No.	Time	Source	Destination	Protocol	Length	Info
30	80.418118064	10.147.19.221	10.147.19.187	RTSP	156	Reply: RTSP/1.0 200 OK
433	87.93796455	10.147.19.187	10.147.19.221	RTSP	261	TEARDOWN rtsp://10.147.19.221:8554/mystream RTSP/1.0
435	87.938173314	10.147.19.221	10.147.19.187	RTSP	113	Reply: RTSP/1.0 200 OK
446	215.804826348	10.147.19.187	10.147.19.221	RTSP	157	OPTIONS rtsp://10.147.19.221:8554/mystream RTSP/1.0
448	215.805139510	10.147.19.221	10.147.19.187	RTSP	194	Reply: RTSP/1.0 200 OK
453	215.852178521	10.147.19.187	10.147.19.221	RTSP	147	OPTIONS rtsp://10.147.19.221:8554/mystream RTSP/1.0
458	215.880849474	10.147.19.187	10.147.19.221	RTSP/S.	596	ANNOUNCE rtsp://10.147.19.221:8554/mystream RTSP/1.0 [Malformed Packet]
460	215.881308311	10.147.19.221	10.147.19.187	RTSP	113	Reply: RTSP/1.0 200 OK
461	215.887818799	10.147.19.187	10.147.19.221	RTSP	147	OPTIONS rtsp://10.147.19.221:8554/mystream RTSP/1.0
463	215.901832342	10.147.19.187	10.147.19.221	RTSP	234	SETUP rtsp://10.147.19.221:8554/mystream/streamid=0 RTSP/1.0
464	215.902861837	10.147.19.221	10.147.19.187	RTSP	238	Reply: RTSP/1.0 200 OK

```

Frame 458: 596 bytes on wire (4768 bits), 596 bytes captured (4768 bits) on interface ztrf23gghit, id 0
  Ethernet II, Src: fe:74:0b:e9:fa:fb (fe:74:0b:e9:fa:fb), Dst: fe:61:d2:30:60:df (fe:61:d2:30:60:df)
  Internet Protocol Version 4, Src: 10.147.19.187, Dst: 10.147.19.221
  Transmission Control Protocol, Src Port: 52499, Dst Port: 8554, Seq: 236, Ack: 129, Len: 530
  [2 Reassembled TCP Segments (674 bytes): #452(144), #458(530)]
  Real Time Streaming Protocol
    Request: ANNOUNCE rtsp://10.147.19.221:8554/mystream RTSP/1.0\r\n
      Method: ANNOUNCE
      URL: rtsp://10.147.19.221:8554/mystream
      Content-type: application/sdp
      CSeq: 2\r\n
      User-Agent: Lavf58.29.100\r\n
      Content-length: 530
    \r\n
  Session Description Protocol
  
```

Figura 26: Tercer paquete modificado

El tercer paquete corresponde nuevamente a una solicitud ANNOUNCE del protocolo RTSP. Al igual que en el primer caso, Wireshark lo identifica como un malformed packet, es decir,

un paquete malformado. Esto indica que el paquete no logró conservar una estructura válida después del intento de modificación y, por lo tanto, no pudo ser procesado correctamente. Tal como se discutió previamente, este tipo de solicitudes (ANNOUNCE) parecen estar sujetas a validaciones más estrictas dentro del protocolo o en su implementación por parte del servidor. Es posible que, durante el intento de modificación con herramientas como Scapy, el paquete haya sido transmitido de manera incompleta o se haya invalidado algún campo clave que impidió su correcta reconstrucción. También es probable que exista algún mecanismo de protección que impida modificar exitosamente este tipo de mensajes críticos, lo que explicaría por qué el paquete fue enviado con errores de formato antes de completar la alteración. En consecuencia, se concluye que este tercer intento de modificación también fue fallido, lo que refuerza la idea de que las solicitudes ANNOUNCE no son fácilmente manipulables sin desencadenar errores de integridad o estructura.

6. Conclusión Capitulo III

Durante el desarrollo de este capítulo, pudimos explorar de forma práctica cómo se puede interceptar y modificar el tráfico de red utilizando la herramienta Scapy. Esta experiencia fue muy útil para entender no solo cómo funcionan los paquetes en una red RTSP, sino también cómo ciertos cambios en campos clave pueden afectar directamente la comunicación entre cliente y servidor.

Uno de los puntos más interesantes fue comprobar que no todos los paquetes reaccionan igual a las modificaciones: mientras algunos, como los mensajes OPTIONS, se aceptaron sin problema, otros como ANNOUNCE fueron rechazados por estar malformados. Esto refuerza la idea de que el protocolo RTSP, o al menos su implementación, tiene ciertos mecanismos de control que evitan manipulaciones maliciosas.

También fue clave trabajar dentro de un entorno aislado con Docker, ya que nos permitió hacer todas las pruebas sin poner en riesgo otras redes o dispositivos. En resumen, este capítulo no solo nos enseñó a usar Scapy, sino que nos dio una mejor comprensión de cómo funciona la seguridad y la integridad de los datos en redes reales.

Referencias

- [1] Repositorio Github, *Tarea3-Redes-Scapy*, 2025. Available online: <https://github.com/diegofranco1/Tarea3-Redes-Scapy>
- [2] Reolink Blog, *What is RTSP?*, 2021. Available online: <https://reolink.com/blog/what-is-rtsp/>
- [3] Kaltura Blog, *RTSP Streaming: What Is It And How Does It Work?*, 2021. Available online: <https://corp.kaltura.com/blog/rtsp-streaming/>

- [4] Schulzrinne, H., Rao, A., & Lanphier, R. (1998). *RFC 2326: Real Time Streaming Protocol (RTSP)*. Internet Engineering Task Force (IETF). Available online: <https://datatracker.ietf.org/doc/html/rfc2326>
- [5] FFmpeg Developers, *FFmpeg GitHub Repository*. Available online: <https://github.com/FFmpeg/FFmpeg>
- [6] FFmpeg, *Official Documentation*. Available online: <https://ffmpeg.org/ffmpeg.html>
- [7] aler9, *rtsp-simple-server (MediaMTX)*. Available online: <https://github.com/aler9/rtsp-simple-server>
- [8] IONOS, *Cómo instalar Docker en Ubuntu 20.04*. Available online: <https://www.ionos.com/es-us/digitalguide/servidores/configuracion/docker-ubuntu-2004/>
- [9] ZeroTier, *ZeroTier Official Website*. Available online: <https://www.zerotier.com/>