http://doi.org/10.46480/esj.9.2.237

# Guía de Buenas Prácticas sobre Ciberseguridad en RouterOS mediante Contenedores

## Best Practices Guide for Cybersecurity in RouterOS Using Containers

Marlon Moposita-Tonato[1], y Alberto Arellano-Aucancela[2]

**RESUMEN**

**Contexto:** Este estudio presenta una guıa de buenas practicas para fortalecer la ciberseguridad en dispositivos MikroTik RouterOS que disponen de soporte para contenedores, considerando entornos de infraestructura limitada, como redes domesticas, laboratorios academicos y organizaciones de pequeña escala. **Metodologıa:** La metodologıa OSSTMM fue adoptada como marco de referencia y estructurada en cinco fases que comprenden la configuracion inicial del sistema, el analisis de servicios integrables, el despliegue de contenedores, la validacion tecnica del entorno y la consolidacion de resultados. **Resultados:** Se desarrollaron e implementaron cuatro contenedores especializados, construidos a partir de imagenes oficiales y adaptadas a la arquitectura ARM64. Alpine, Nmap, Rsyslog y FreeRADIUS fueron ejecutados sobre un RouterBoard RB5009, incorporando funcionalidades como escaneo de servicios, registro de eventos y autenticacion centralizada, sin comprometer la estabilidad del sistema. El comportamiento operativo fue monitoreado mediante herramientas integradas, lo que permitio registrar el consumo de recursos y el funcionamiento del entorno bajo distintas condiciones de carga. **Conclusiones:** Los resultados obtenidos evidencian que es viable incorporar capacidades de seguridad mediante el uso de Docker sobre RouterOS. Esta implementacion permite mejorar la madurez en ciberseguridad sin requerir infraestructura dedicada adicional.

**Palabras clave:** RouterOS, Contenedores Docker, OSSTMM, MikroTik, Madurez en ciberseguridad

**ABSTRACT**

**Context:** This study presents a best-practice guide to strengthen cybersecurity in MikroTik RouterOS devices that have support for containers, considering infrastructure-constrained environments such as home networks, academic laboratories, and small-scale organizations. **Methodology:** The OSSTMM methodology was adopted as the reference framework and structured into five phases that include the initial system configuration, analysis of integrable services, container deployment, technical validation of the environment, and consolidation of results. **Results:** Four specialized containers were developed and implemented, built from official images and adapted to the ARM64 architecture. Alpine, Nmap, Rsyslog, and FreeRADIUS were executed on a RouterBoard RB5009, incorporating functionalities such as service scanning, event logging, and centralized authentication, without compromising system stability. Operational behavior was monitored through integrated tools, allowing the registration of resource consumption and the functioning of the environment under different load conditions. **Conclusions:** The results obtained show that it is feasible to incorporate security capabilities through the use of Docker on RouterOS. This implementation allows advancing in cybersecurity maturity levels without requiring additional dedicated infrastructure.

**Keywords:** RouterOS, Docker containers, OSSTMM, MikroTik, Cybersecurity maturity

**Fecha de recepción:** Junio 13, 2025
**Fecha de aceptación:** Septiembre 30, 2025

## Introducción

RouterOS, es el sistema operativo desarrollado por MikroTik e integrado en sus dispositivos de red, se basa en el kernel de Linux (Mikrotik, 2025b). Su diseño facilita la gestión de interfaces y capacidades de enrutamiento, con una arquitectura modular que permite su implementación en redes educativas, residenciales y corporativas a nivel mundial (Prakosa et al., 2024).

Sin embargo, estas mismas propiedades exponen a RouterOS a riesgos operativos. (Wijayanto et al., 2022) identifican vulnerabilidades como ataques de fuerza bruta, redirección, inyección de código y denegación de servicio. Muchos de estos vectores se originan en plataformas en la nube, utilizadas como punto de entrada para comprometer equipos MikroTik. Por ello, recomienda mantener el sistema actualizado y aplicar medidas de seguridad adicionales que mitiguen estos riesgos.

La documentación oficial de (Mikrotik, 2025b) presenta el soporte para la ejecución de contenedores en RouterOS v7, orientado a la implementación de aplicaciones desarrolladas por terceros dentro del entorno del sistema operativo. Además, proporciona directrices técnicas para su habilitación, configuración y operación segura.

Según (Nkengereye et al., 2025), los contenedores constituyen una alternativa eficiente para desplegar aplicaciones distribuidas, dada su compatibilidad con arquitecturas de microservicios y su adaptabilidad a

[1]Pontificia Universidad Católica del Ecuador. E-mail: [mmmoposita@pucesa.edu.ec]
[2]Pontificia Universidad Católica del Ecuador. E-mail: [eaarellano@pucesa.edu.ec]



**Cómo citar:** Moposita-Tonato, M., & Arellano-Aucancela, A. (2025). Guía de Buenas Prácticas sobre Ciberseguridad en RouterOS mediante Contenedores. *Ecuadorian Science Journal, 9*(2), *21-30*, September–2025. DOI: https://doi.org/10.46480/esj.9.2.237

entornos heterogéneos, como el *edge computing*. Sin embargo, (Wang et al., 2024) advierten que fallos en el aislamiento de registros pueden provocar filtraciones de información y generar interferencias entre contenedores. Por ello, el análisis debe contemplar el sistema anfitrión, el contenedor y las aplicaciones desplegadas como un conjunto interdependiente.

(García Herrera and Cuenca Tapia, 2021) así como (AlHidaifi et al., 2024), destacan que la aplicación de lineamientos de seguridad estructurados permite fortalecer la protección de infraestructuras críticas, mitigar vulnerabilidades y preservar atributos esenciales como la confidencialidad, integridad y disponibilidad. Asimismo, contribuye a la recuperación ante fallos y respalda estrategias de resiliencia cibernética.

La presente investigación evalúa la implementación de contenedores Docker en equipos MikroTik con RouterOS v7 con el propósito elevar los niveles de madurez en ciberseguridad. Se incorporaron herramientas específicas para funciones como escaneo de servicios, registro de eventos y autenticación, utilizando imágenes Docker personalizadas construidas a partir de fuentes confiables. La validación se realiza en un entorno de laboratorio controlado, donde se analiza el funcionamiento, la compatibilidad operativa y el rendimiento del sistema. Como resultado, se presentan directrices de buenas prácticas para la configuración segura de RouterOS v7. Estas se fundamentan en el uso de contenedores Docker adaptados a requerimientos de seguridad propios de redes institucionales.

La validación se estructuró mediante un diseño cuasiexperimental de tipo longitudinal, con observaciones realizadas durante la implementación progresiva de los contenedores. Esta estrategia permitió evaluar el comportamiento del sistema durante la incorporación de servicios adicionales, así como el uso de recursos involucrado en su ejecución.

Se adopta como referencia metodológica *Open Source Security Testing Methodology Manual* (OSSTMM) (Herzog, 2010), adaptado al entorno experimental para estructurar las fases de configuración, despliegue de servicios y análisis de las medidas de seguridad implementadas. Según (**?**), este enfoque combina técnicas actuales con criterios estructurados que permiten una evaluación integral de los sistemas.

## Trabajos Relacionados

Donca et al. plantean que la incorporación de mecanismos de gestión de credenciales y autenticación en entornos *IoT* permite mitigar vulnerabilidades comunes, fortaleciendo la seguridad mediante pruebas de penetración y escaneo de vulnerabilidades (Donca et al., 2024) .

El Amin et al. argumentan que las organizaciones deben adoptar un enfoque proactivo ante las amenazas, priorizando estrategias preventivas en lugar de respuestas reactivas. Señalan que las capacidades para detectar, responder y recuperarse ante incidentes deben desarrollarse como parte integral de una estrategia orientada a mantener la seguridad operativa (El Amin et al., 2024).

Palate & Avila, en su estudio *Mitigación de vulnerabilidades en la red central de un ISP: Un caso de estudio*, evidencian el incremento de incidentes de ciberseguridad en la infraestructura de proveedores de servicios de Internet (ISP) y la necesidad de identificar vulnerabilidades que requieran protección inmediata. Destaca la función del *firewall* para filtrar paquetes de datos, analizar cabeceras y tomar decisiones de enrutamiento basadas en reglas establecidas. Se emplea una infraestructura MikroTik con RouterOS como caso de estudio, donde se implementaron reglas de seguridad en el *firewall* para mitigar ataques como denegación de servicio (DoS) y fuerza bruta. Como resultado, el consumo de CPU se redujo en un 50% durante los ataques, lo que permitió conservar la estabilidad y la disponibilidad de la red de comunicaciones (Palate and Avila, 2021).

Cerino Frías et al. analizan la seguridad de Docker en servidores Linux, centrándose en su comportamiento frente a amenazas en entornos de virtualización ligera. El estudio examina dos aspectos principales: los mecanismos internos de aislamiento del contenedor y la interacción de Docker con módulos de seguridad del kernel, como *SELinux* y *AppArmor*, orientados a reforzar la protección del sistema huésped (Cerino Frías et al., 2021).

Los estudios de (Flauzac et al., 2020) junto con los de (Alqaisi et al., 2023), comparan diversas soluciones de contenedores como *LXC, LXD, Singularity, Docker, Kata-containers y gVisor*. A pesar de sus diferencias metodológicas, los dos trabajos aportan criterios técnicos sobre aislamiento, almacenamiento y compatibilidad en entornos con recursos limitados. En particular, (Alqaisi et al., 2023) evalúan el rendimiento de tecnologías como Docker y Singularity en plataformas embebidas, con arquitectura ARM, evaluando su eficiencia operativa en condiciones de capacidad computacional restringida.

Kaiser et al. investigan la viabilidad del uso de contenedores en dispositivos IoT, gateways industriales y plataformas con arquitectura ARM, destacando su aplicabilidad en escenarios de bajo consumo. Sus resultados muestran que Docker puede mantener un aislamiento funcional adecuado con un uso moderado de CPU y memoria, siempre que se utilicen imágenes optimizadas y mecanismos de control de recursos. También documentan el soporte nativo para ARM en herramientas modernas de contenedorización, consolidando a Docker como una opción robusta y ampliamente adoptada (Kaiser et al., 2022).

Aunque ninguno de estos trabajos aborda directamente RouterOS, sus hallazgos respaldan el enfoque de esta investigación, al evidenciar la aplicabilidad de contenedores personalizados en arquitecturas ARM64 dentro de dispositivos embebidos como el RouterBoard.

## Metodología y Métodos

La presente investigación adopta la metodología *OSSTMM* como guía de análisis de ciberseguridad, mediante una secuencia estructurada de cinco fases, según (Palma and Carrillo, 2022), adaptadas a un entorno basado en RouterOS con soporte para contenedores. En la fase inicial, se aplicaron recomendaciones derivadas de estudios preliminares de (Palate and Avila, 2021) sobre configuraciones de seguridad en RouterOS. Estas recomendaciones se implementaron en un equipo MikroTik con arquitectura ARM64. El entorno de prueba replicó características comunes a entornos de producción.

A partir de este entorno, se llevó a cabo la enumeración de servicios activos y capacidades del sistema, lo que permitió identificar los requisitos para habilitar la funcionalidad de contenedores. Con base en este análisis, se procedió a desplegar cuatro contenedores especializados: Alpine base, Nmap, Rsyslog y FreeRADIUS. Cada uno fue configurado con herramientas orientadas a funciones específicas de seguridad. Esta integración se desarrolló para validar la compatibilidad operativa y aportar capacidades como escaneo, registro, autenticación y supervisión. La incorporación de estos servicios fue acompañada por ajustes específicos en las políticas de red y configuración del sistema, tal como lo señalan (Jabr et al., 2024).

Se desarrollaron pruebas cuasiexperimentales orientadas a validar la eficacia de las medidas de seguridad implementadas, considerando los criterios de integridad, confidencialidad y disponibilidad de los servicios desplegados, siguiendo el enfoque propuesto por (Jabr et al., 2024). Finalmente, se consolidaron los hallazgos técnicos obtenidos y se formularon recomendaciones orientadas a mejorar la capacidad de respuesta del sistema frente a escenarios operativos similares, como redes institucionales de pequeña escala o entornos educativos. Este proceso proporciona una base replicable y estructurada para futuras implementaciones de seguridad en dispositivos embebidos basados en contenedores.

La Figura 1 resume las fases metodológicas seguidas en el desarrollo de esta investigación.

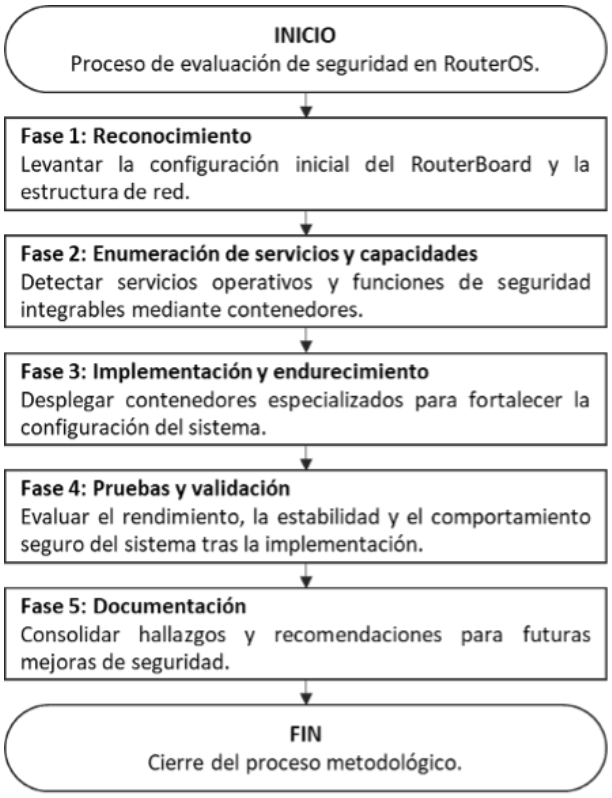


**Figura 1.** Fases metodológicas basadas en OSSTMM adaptadas a RouterOS.

## *Entorno Experimental y Configuración Inicial del Sistema*

El entorno de laboratorio representado en la Figura 2 se ejecutó en un MikroTik RouterBoard RB5009UG, con arquitectura ARM64 y sistema RouterOS v7.18.2. Está equipado con un procesador de cuatro núcleos (88F7040) con una frecuencia dinámica que varía entre 350 y 1400 MHz, 1 GB de memoria RAM y 1 GB de almacenamiento interno tipo NAND (Mikrotik, 2025a). Se configuró como nodo central de la red experimental con soporte habilitado para la ejecución de contenedores.

Se establecieron dos interfaces lógicas de tipo *bridge*: br_lan, orientada a la gestión del sistema, y br_docker, destinada al aislamiento de los contenedores. Cada *bridge* fue asociado a un rango de direcciones IP independiente, con reglas de *firewall* personalizadas y políticas de seguridad adaptadas a su función.

Para reforzar la postura defensiva del sistema, se aplicaron las recomendaciones de (Palate and Avila, 2021), incorporando reglas de *firewall* para el control de conexiones por estado, mitigación de ataques comunes, como los de tipo *SYN flood* e *ICMP Smurf*, y la desactivación de servicios innecesarios. Se restringió la administración remota únicamente a los servicios *SSH* y *Winbox*, y se implementaron filtros basados en direcciones IP y protocolos para restringir accesos no autorizados y proteger servicios críticos.

Además, el sistema anfitrión ejecutó la herramienta *The Dude*, integrada en RouterOS (Mikrotik, 2025b), para monitorear el rendimiento del dispositivo durante las fases de implementación de contenedores. El Listado 1 presenta el bloque de configuración correspondiente a /ip/firewall/filter de RouterOS, donde se definen reglas de control de tráfico de entrada y reenvío de paquetes.

```
1  # Permitir acceso administrativo por canales
       seguros
2  add chain=input protocol=tcp dst-port=22,8291
       action=accept comment="Acceso␣SSH␣y␣Winbox"
3  add chain=input protocol=udp dst-port=161 action=
       accept connection-state=new dst-address
       =10.11.12.1 comment="Acceso␣SNMP␣desde␣The␣
       Dude"
4
5  # Permitir comunicaci n UDP desde la LAN hacia
       servicios en contenedores
6  add chain=forward protocol=udp dst-port=514 src-
       address=10.10.10.0/24 dst-address=10.11.12.4
       action=accept comment="Syslog␣hacia␣
       contenedor␣Rsyslog"
7  add chain=forward protocol=udp dst-port=1812,1813
        src-address=10.10.10.0/24 dst-address
       =10.11.12.5 action=accept comment="RADIUS␣
       hacia␣contenedor␣FreeRADIUS"
8
9  # Permitir tr fico del contenedor Nmap
10 add chain=forward src-address=10.11.12.2 action=
       accept comment="Tr fico␣desde␣contenedor␣
       Nmap"
11
12 # Aplicar pol tica por defecto para denegar
       tr fico no autorizado
13 add chain=forward action=drop comment="Bloquear␣
       reenv o␣no␣autorizado"
14 add chain=input action=drop comment="Bloquear␣
       entrada␣no␣autorizada"
```

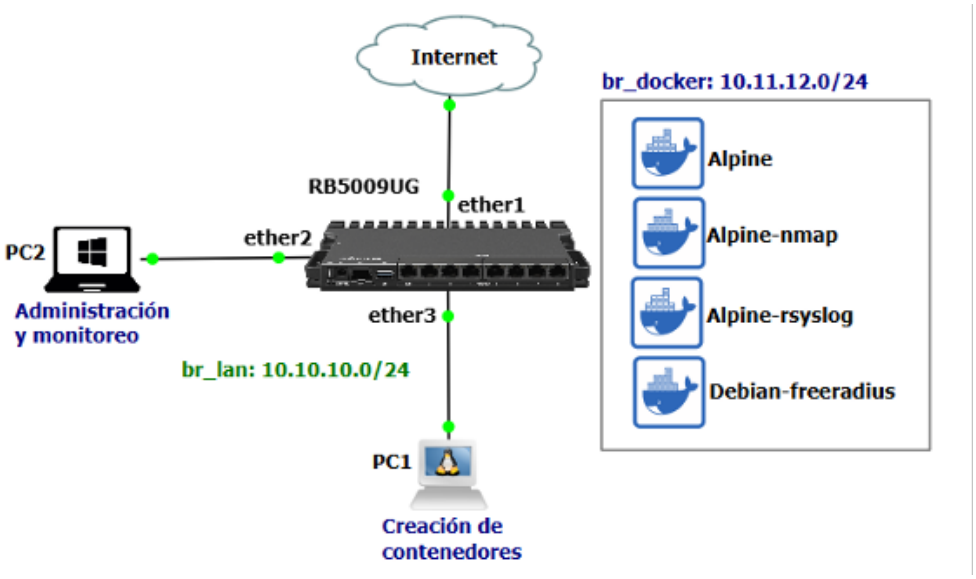


**Figura 2.** Diagrama de Red de Laboratorio.

Listing 1: Reglas de filtrado configuradas en MikroTik RouterOS

Este conjunto de reglas establece restricciones por dirección IP y puertos para cada contenedor, de modo que únicamente las direcciones IP internas interactúan con los servicios habilitados. Mediante el *bridge* br_docker se configura el aislamiento lógico de red para los contenedores, limitando tanto el acceso como la exposición a servicios sensibles.

### *Habilitación del Entorno de Contenedores en RouterOS*

Previo al despliegue de servicios, se habilitó la funcionalidad de contenedores con el comando /system/device-mode/update container=yes (Mikrotik, 2025b). Este cambio de modo requiere confirmación física local (presionar un botón o reconectar el equipo), lo que impide su ejecución remota y asegura que sea realizado por personal autorizado.

Luego, se instaló el paquete container correspondiente a la arquitectura ARM del sistema operativo, y se conectó una unidad USB externa formateada en *ext4*, destinada al entorno de ejecución de contenedores conforme a las recomendaciones oficiales (Mikrotik, 2025b). Este almacenamiento se destinó al entorno de ejecución, permitiendo preservar la integridad de la memoria interna y facilitar la administración modular de los servicios desplegados. La Figura 3 muestra la disposición de directorios definida en el almacenamiento externo configurado para contenedores.

La activación del entorno de contenedores en RouterOS implica riesgos de seguridad reconocidos por el propio fabricante (Mikrotik, 2025b). Una vez habilitada esta función, cualquier contenedor ejecutado puede ampliar la superficie de ataque del sistema. MikroTik advierte que la seguridad del dispositivo depende de la confiabilidad de los contenedores empleados, sin ofrecer aislamiento completo ni protección nativa frente a cargas de terceros. En respuesta a esta limitación, la presente investigación adopta un enfoque de desarrollo seguro, como el propuesto por (**?**), que contempla el uso exclusivo de imágenes oficiales, la reducción de servicios expuestos, la implementación de entornos mínimos y la aplicación de reglas de *firewall* para segmentación y control de tráfico.

### *Construcción y Despliegue de Contenedores*

Las imágenes Docker, compatibles con arquitectura ARM64, fueron construidas externamente sobre un sistema Debian. Se utilizaron imágenes oficiales como base, para garantizar el control sobre las dependencias y configuraciones. Las imágenes generadas se exportaron en formato .tar y almacenaron en la unidad USB del equipo MikroTik. Esta estrategia permite mantener una infraestructura controlada, alineada con prácticas seguras en la gestión de contenedores, en concordancia con los hallazgos de (Mills et al., 2023), quienes documentan la importancia de limitar el uso de componentes no gestionados para reducir la exposición a vulnerabilidades presentes en imágenes desactualizadas o mal mantenidas.

El Listado 2 presenta el proceso general para construir, exportar y cargar una imagen Docker destinada a ejecutarse en RouterOS:

```
# Construcci n externa en host Debian
FROM arm64v8/alpine:latest
RUN apk add --no-cache nmap libxslt tzdata
CMD ["/bin/sh"]

# Exportaci n y carga en RouterOS
$ docker buildx build --platform linux/arm64 -t
    alpine-nmap --load .
$ docker save alpine-nmap:latest > alpine-nmap.
    tar
$ scp alpine-nmap.tar admin@10.11.12.1:/usb1/
    docker/images/
```

Listing 2: Proceso de construcción y carga de una imagen Docker para RouterOS

El mismo procedimiento fue replicado a los demás contenedores, adaptando paquetes, configuraciones y puertos necesarios para cada servicio. Para garantizar la

```
usb1/
└─ docker/
    ├─ containers/  # Contenedores activos en tiempo de ejecución
    ├─ images/      # Imágenes Docker almacenadas en formato .tar
    ├─ mounts/      # Volúmenes persistentes utilizados por los contenedores
    └─ tmp/         # Archivos auxiliares o temporales generados durante la ejecución
```

**Figura 3.** Estructura de archivos utilizada en la ejecución de contenedores.

trazabilidad y la seguridad, se evitó el uso de imágenes de terceros y se documentó cada Dockerfile como parte del material complementario.

Luego de almacenar las imágenes en el dispositivo, se crearon los contenedores en RouterOS mediante comandos específicos. Los comandos especifican la interfaz virtual, el nombre del contenedor, las variables de entorno y los puntos de montaje, conforme a lo indicado por (Mikrotik, 2025b). El Listado 3 muestra la creación del contenedor Nmap como ejemplo representativo del procedimiento aplicado a los demás contenedores desplegados en RouterOS.

```
/container/add \
  file=usb1/docker/images/alpine-nmap.tar \
  interface=veth-alpine-nmap \
  root-dir=usb1/docker/containers/alpine-nmap \
  mounts=usb1/docker/mounts/alpine-nmap \
  envlist=alpine-nmap \
  hostname=alpine-nmap \
  domain-name=docker.local \
  dns=10.11.12.1 \
  cmd="tail -f /dev/null"
```

Listing 3: Creación del contenedor Nmap en RouterOS

La implementación de los servicios se realizó de forma progresiva, con el propósito de evaluar el consumo de recursos del sistema anfitrión. El monitoreo continuo, realizado mediante *The Dude*, permitió documentar el comportamiento del sistema operativo conforme se integraban los servicios. Esta estrategia permitió una observación sistemática orientada a identificar ajustes durante la fase de implementación y endurecimiento, y a validar el comportamiento operativo del sistema durante las pruebas de rendimiento y estabilidad.

*Contenedor Alpine*

Como punto de partida en la implementación progresiva, se desplegó un contenedor basado en la imagen alpine:latest, seleccionada por su compatibilidad con arquitectura ARM64 y su diseño reducido. Su bajo consumo de recursos lo convierte en una opción adecuada para comprobar la operatividad del entorno de contenedores en RouterOS, mediante tareas locales como la verificación de conectividad y la exploración del sistema de archivos desde línea de comandos (*CLI*).

La selección de esta imagen responde a criterios de seguridad y control, alineados con lo documentado por (Algarni et al., 2024), quienes advierten que incluso las imágenes oficiales pueden presentar vulnerabilidades persistentes si no se gestionan adecuadamente. Bajo este enfoque, Alpine proporciona un entorno confiable para iniciar la evaluación incremental de nuevos servicios lo que permite mantener una superficie de ataque mínima y controlada.

En la Figura 4 se muestra la consola con la verificación del sistema operativo del contenedor Alpine versión 3.20.3.

*Contenedor Alpine con Nmap*

Este contenedor extiende la imagen base alpine con la herramienta *Nmap*, habilitando capacidades de escaneo y auditoría de red ejecutadas localmente en RouterOS. Su ejecución es puntual y controlada, sin exposición de puertos hacia el exterior. Durante las pruebas, se registraron incrementos breves en el uso de CPU asociados a escaneos simultáneos, por lo que se optó por una ejecución secuencial de los procesos de enumeración. Esta decisión se fundamenta en estudios que evidencian el consumo intensivo de recursos en operaciones de escaneo automatizado y recomiendan una planificación cuidadosa en entornos embebidos con recursos limitados (Jabr et al., 2024).

En la Figura 5 se presenta un escaneo ejecutado exitosamente desde el contenedor, donde se evidencia la ejecución funcional del servicio.

*Contenedor Alpine con Rsyslog*

Este contenedor extiende la imagen alpine:latest, con la incorporación de *Rsyslog*, configurado para la recepción de registros a través del puerto UDP 514 desde equipos de red. Los mensajes se almacenan en volúmenes montados sobre una unidad USB externa. Como medida de seguridad, se implementaron reglas de *firewall* que restringen el acceso exclusivamente a emisores autorizados y se definieron políticas de retención que previenen la saturación del almacenamiento. Esta configuración se ajusta a principios de visibilidad operacional y trazabilidad en entornos perimetrales, de acuerdo con lineamientos vigentes que requieren registros estructurados para procesos de análisis o clasificación de eventos (Wijayanto et al., 2022).

En la Figura 6 se muestra el registro de eventos capturados, demostrando la correcta recepción de mensajes desde el sistema remoto.

*Contenedor Debian con FreeRADIUS*

Desplegado sobre debian:bookworm, este contenedor implementa un servidor *FreeRADIUS* con los puertos UDP 1812 y 1813 habilitados, orientado a la autenticación centralizada en redes que no cuentan con servidores dedicados. A diferencia de los contenedores previos basados en Alpine, la elección de Debian responde a la necesidad de una mayor compatibilidad con bibliotecas requeridas por FreeRADIUS y a su desempeño más estable en entornos de autenticación avanzada, como se ha documentado en entornos basados en *Ubuntu Server* (Ochoa Villanueva and Roman Gonzalez,

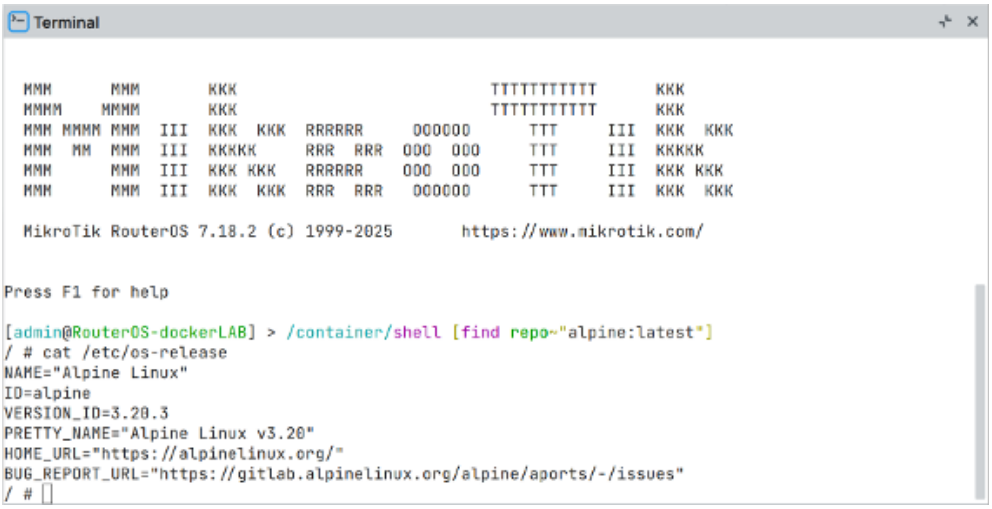


**Figura 4.** Contenedor Alpine base ejecutado en RouterOS.

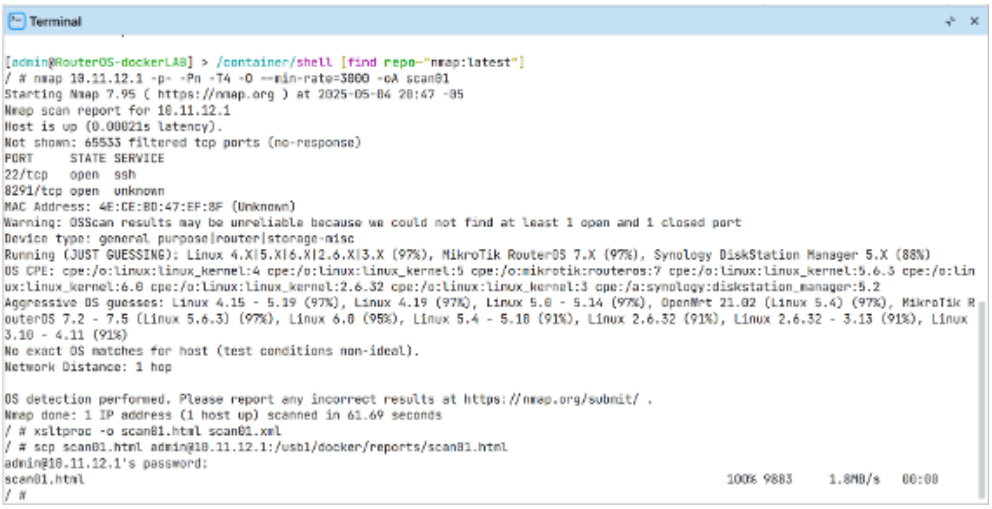


**Figura 5.** Escaneo de red ejecutado desde contenedor Nmap.

2023). Se definieron volúmenes persistentes para el almacenamiento de registros en la unidad externa, y se mantuvo una estructura modular que permite extender la funcionalidad mediante mecanismos como EAP (*Extensible Authentication Protocol*) o certificados digitales, en función de los requisitos del entorno. Las evaluaciones realizadas mostraron un consumo moderado de recursos y un comportamiento estable, adecuado para entornos embebidos de baja concurrencia, de acuerdo con buenas prácticas documentadas en esquemas de autenticación distribuida para redes embebidas.

La Figura 7 muestra un ejemplo de registro generado tras un inicio de sesión exitoso mediante FreeRADIUS.

La ejecución de los cuatro contenedores permitió comprobar su integración operativa en RouterOS. El monitoreo continuo mediante *The Dude* permitió registrar el comportamiento del sistema anfitrión en cuanto a consumo de CPU, memoria RAM y almacenamiento interno, reflejando variaciones en la carga de recursos según las funcionalidades de cada contenedor.

Esta implementación, orientada a funciones de escaneo, registro, autenticación y supervisión del entorno, demostró que es posible ampliar las capacidades funcionales del sistema manteniendo condiciones operativas estables. La sección siguiente expone los resultados obtenidos, acompañados de un análisis comparativo del uso de recursos por cada contenedor.

## Resultados

Tal como se detalló en la sección anterior, los contenedores Docker fueron desplegados de forma progresiva en un entorno controlado basado en RouterOS v7, lo que permitió evaluar de forma incremental el consumo de recursos del sistema anfitrión. Cada contenedor fue desplegado manteniendo activos los previamente ejecutados, permitiendo un análisis comparativo acumulativo de su comportamiento operativo.

El análisis del consumo de CPU, memoria y almacenamiento se llevó a cabo mediante dos herramientas de monitoreo: por un lado, los registros gráficos generados por *The Dude*, que ofrecen trazabilidad visual del comportamiento del sistema a lo largo del tiempo; por otro, las lecturas puntuales obtenidas mediante el comando /system/resources/print, que permiten obtener mediciones del estado del sistema en momentos definidos. Esta combinación de fuentes de información facilitó correlacionar los datos observados y

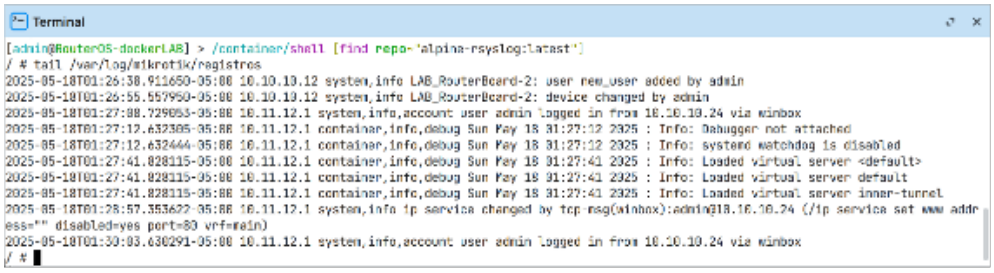


**Figura 6.** Recepción de registros en contenedor Rsyslog.


**Figura 7.** Registro de autenticación en contenedor FreeRADIUS.

realizar un seguimiento preciso de las variaciones asociadas a la incorporación progresiva de contenedores.

La Tabla 1 presenta los valores registrados bajo distintas condiciones operativas, lo que permite analizar el consumo acumulativo sobre el consumo de memoria, CPU y almacenamiento interno a medida que se incorporan los contenedores. La Figura 8, por su parte, muestra el comportamiento general del sistema con los cuatro contenedores en operación.

Aunque las gráficas de uso de disco corresponden al almacenamiento interno del RouterBoard RB5009UG, los contenedores fueron desplegados en una unidad USB externa. Esta estrategia permite aislar el entorno de ejecución de los servicios en contenedor, preservar la integridad del equipo y facilitar una gestión escalable y segura de los datos persistentes.

Los registros evidencian variaciones progresivas en el consumo de recursos conforme se incorporaron los contenedores, cuyas implicaciones se analizan en la sección de discusión.

## Discusión

La implementación progresiva de contenedores Docker en RouterOS v7 permitió evaluar efectos técnicos asociados a su ejecución en dispositivos embebidos con recursos limitados. Esta estrategia se desarrolló en función del soporte nativo que RouterOS brinda para arquitecturas ARM, ARM64 y AMD64, las cuales constituyen el conjunto de plataformas actualmente habilitadas para ejecutar instancias de contenedores de forma oficial.

El proceso de evaluación registró variaciones específicas en el consumo de CPU del sistema anfitrión. En condiciones de carga mínima, el uso se mantuvo por debajo del 5%. No obstante, servicios que implica procesamiento intensivo, como *Nmap*, generaron picos de hasta un 25%. La recurrencia de estos eventos sugiere una ejecución controlada de tareas de alta demanda, ya que un uso intensivo de CPU por parte de un contenedor reduciría los recursos disponibles para otros servicios y podría afectar la capacidad operativa del sistema principal.

Respecto al uso de memoria, se observó un aumento progresivo conforme se integraban nuevos contenedores, alcanzando un máximo aproximado de 630 MB en el escenario de mayor densidad funcional. Este comportamiento, es tolerable en dispositivos con al menos 1 GB de RAM, puede optimizarse mediante el uso del parámetro /container/config set ram-high=, disponible en RouterOS, para restringir el consumo según la demanda esperada del servicio. Establecer límites según la naturaleza del contenedor constituye una práctica recomendada para preservar la estabilidad del sistema.

Durante la ejecución simultánea de servicios de escaneo, autenticación y monitoreo, se observó que el procesador operaba en sus niveles superiores de frecuencia, característicos del modelo RB5009UG, y que el uso de CPU alcanzaba picos de hasta el 50%, lo que refleja el impacto operativo de tareas intensivas cuando se ejecutan en paralelo. La Tabla 2 resume los rangos de consumo de recursos por contenedor, junto con observaciones sobre su comportamiento operativo.

Desde la perspectiva de arquitectura de red, la implementación de reglas de *firewall* específicas permitió delimitar el tráfico hacia los servicios contenedorizados, reduciendo la exposición a amenazas externas. De esta forma, se configuró un entorno con principios de seguridad en profundidad, restringiendo accesos y aislando servicios críticos.

Adicionalmente, se evaluó el aporte funcional de cada contenedor en relación con dimensiones funcionales

**Tabla 1.** Consumo de recursos del sistema con ejecución de contenedores en RouterOS

| Condición operativa | Memoria utilizada (MB / Total) | Uso de CPU (%) | Frecuencia de CPU (MHz) | Uso de disco interno (MB / Total) |
|---|---|---|---|---|
| Línea base sin contenedores | 496.3 / 1024 | 1 | 350 | 983.6 / 1024 |
| Contenedor Alpine en modo CLI | 497.2 / 1024 | 1 | 350 | 983.6 / 1024 |
| Contenedor adicional Nmap | 497.9 / 1024 | 2 | 350 | 983.6 / 1024 |
| Contenedor adicional Rsyslog | 498.3 / 1024 | 4 | 350 | 983.6 / 1024 |
| Contenedor adicional FreeRADIUS | 571.5 / 1024 | 4 | 350 | 983.6 / 1024 |
| Contenedores activos: Alpine, Nmap, Rsyslog y FreeRADIUS simultáneos | 646.7 / 1024 | 25-50 | 1400 | 983.6 / 1024 |

**Fuente:** Captura directa del sistema RouterOS v7.18.2 mediante /system/resource/print

**Tabla 2.** Perfil de consumo de recursos por contenedor en RouterOS

| Contenedor | Consumo CPU (%) | Consumo Memoria (%) | Uso de Almacenamiento Interno (%) | Observaciones |
|---|---|---|---|---|
| Alpine | 1 | 1-2 | <1 | Ejecutado en CLI, con consumo estable entre el arranque y la operación |
| Alpine-nmap | 1–25 (por escaneo) | 1–6 (por escaneo) | <1 | Cada escaneo incrementa el uso de recursos. Se recomienda evitar ejecuciones simultáneas |
| Alpine-rsyslog | 1–3 | 1–3 | <1 | Consultas a registros incrementan temporalmente el uso de CPU y memoria |
| Debian-freeradius | 1–3 | 8–11 | <1 | Uso de memoria más alto al iniciar, asociado a la base Debian. Autenticación incrementa la carga |

**Fuente:** Elaboración propia a partir de resultados experimentales

vinculadas a la madurez en ciberseguridad. Esta evaluación se orientó a identificar la contribución de cada servicio a ejes como autenticación, trazabilidad, visibilidad del entorno y descubrimiento de activos. La Tabla 3 presenta esta clasificación funcional, determinada a partir de criterios observados durante las pruebas.

Herramientas como *FreeRADIUS* y *Rsyslog* ampliaron las funcionalidades operativas de RouterOS al incorporar autenticación externa y registro centralizado de eventos, respectivamente. A su vez, *Nmap* habilitó capacidades de detección activa de servicios en la red, útiles para la elaboración de inventarios y la verificación de superficies expuestas. Estas extensiones funcionales integran mecanismos adicionales de seguridad al sistema sin alterar su arquitectura base, lo que permiten mantener el control operativo en dispositivos con recursos limitados.

La incorporación de estos servicios mediante contenedores representa una solución técnica viable para entornos que demandan funciones de monitoreo, control o autenticación, sin recurrir a infraestructura externa. Mediante el uso de imágenes oficiales, el aislamiento lógico, la gestión externa del almacenamiento y políticas de acceso definidas, se estableció un entorno de ejecución alineado con prácticas de seguridad recomendadas para sistemas embebidos.

## Conclusiones

La presente investigación confirmó que la integración de contenedores Docker en RouterOS v7 es técnicamente factible y viable para extender funcionalidades de ciberseguridad en dispositivos MikroTik con capacidades embebidas. Considerando que estos entornos no tienen como objetivo reemplazar soluciones especializadas, esta implementación se propone como un mecanismo complementario de seguridad en escenarios con limitaciones de infraestructura, como redes domésticas, laboratorios educativos o nodos de pequeña escala.

Desde una perspectiva técnica, se comprobó la viabilidad de incorporar servicios de autenticación, monitoreo, escaneo y registro manteniendo la estabilidad operativa del dispositivo. La preparación del entorno, el aislamiento funcional mediante interfaces virtuales y el control riguroso del uso de recursos permitieron mantener condiciones estables de operación. Esta arquitectura demuestra que los principios

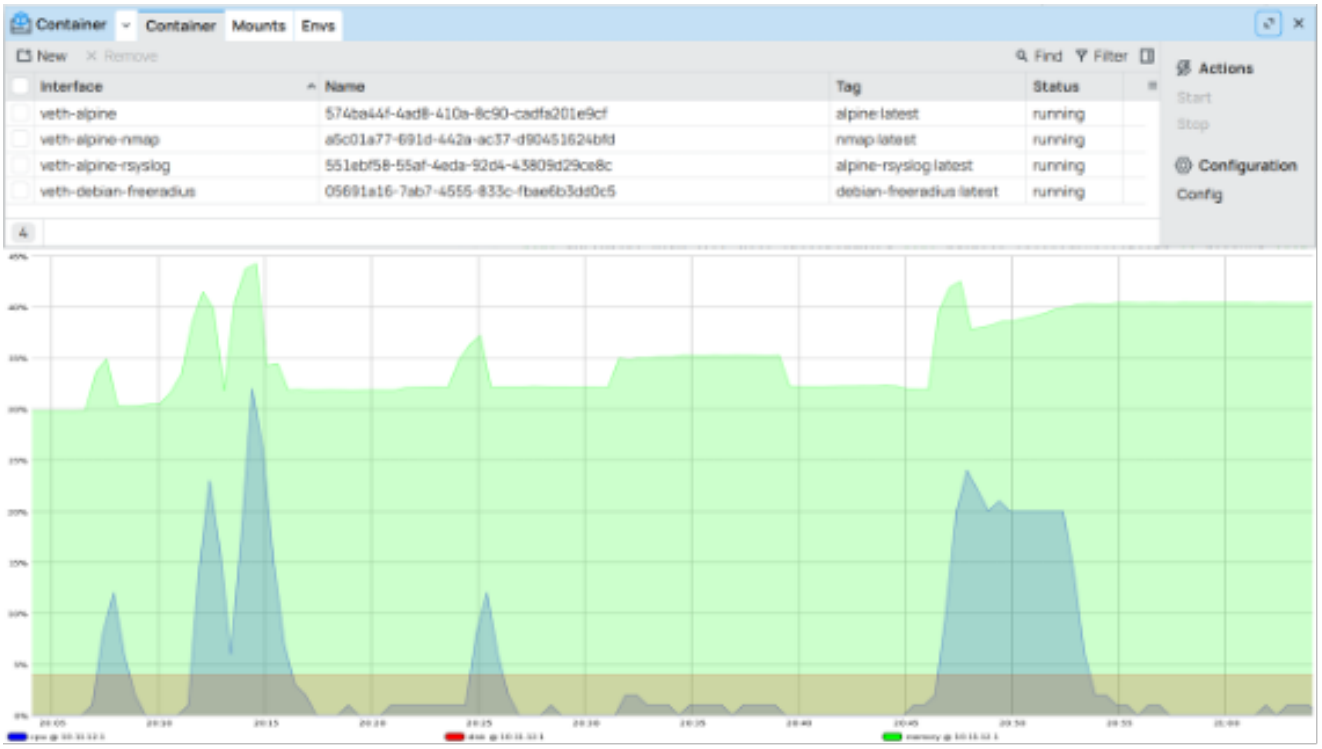


**Figura 8.** Monitoreo de recursos del sistema RouterOS durante ejecución de contenedores.

**Tabla 3.** Aporte funcional de contenedores a la seguridad en RouterOS

| Contenedor | Autenticación | Trazabilidad / Registro | Supervisión del entorno | Descubrimiento / Escaneo |
|---|---|---|---|---|
| Alpine | No | No | Parcial (CLI local) | No |
| Alpine-nmap | No | No | No | Sí |
| Alpine-rsyslog | No | Sí | Sí | No |
| Debian-freeradius | Sí | Parcial (logs internos) | Sí | No |

**Fuente:** Elaboración propia

de seguridad desde el diseño son aplicables en plataformas contenidas.

En términos funcionales, los servicios desplegados extendieron las capacidades defensivas del sistema sin modificar su arquitectura base. La posibilidad de auditar tráfico, registrar eventos relevantes y controlar el acceso desde contenedores aislados integró mecanismos adicionales de protección y redujo la dependencia de soluciones externas o de alto costo.

Se consolidó un modelo de despliegue replicable, fundamentado en el uso de imágenes oficiales, almacenamiento desacoplado y segmentación lógica de red, con capacidad de adaptación a entornos operativos de mayor complejidad. Este esquema de integración proporciona una base estructurada consistente para futuras implementaciones enfocadas a la automatización de servicios, la correlación de eventos y la integración de plataformas distribuidas de monitoreo.

En conjunto, los resultados respaldan el uso estratégico de contenedores como mecanismo de ampliación funcional sobre RouterOS, orientado a incrementar la madurez operativa en ciberseguridad mediante soluciones contenidas, escalables y técnicamente sostenibles.

## Agradecimientos

Los autores agradecen a sus respectivas instituciones por el apoyo brindado para la realización de esta investigación.

## Referencias

Algarni, A., Shah, I., Jehangiri, A. I., Ala'anzy, M. A., and Ahmad, Z. (2024). Predictive energy management for docker containers in cloud computing: A time series analysis approach. *IEEE Access*, 12:52524–52538.

AlHidaifi, S. M., Asghar, M. R., and Ansari, I. S. (2024). A survey on cyber resilience: Key strategies, research challenges, and future directions. *ACM Computing Surveys*, 56(8):1–48.

Alqaisi, O. I., Tosun, A. , and Korkmaz, T. (2023). Performance analysis of container technologies for computer vision applications on edge devices. *IEEE Access*, 12:41852–41869.

Cerino Frías, R., Magaña, J. J., Hernández Cadena, A., Garrido Vázquez, J. N., and Gómez Zea, J. M. (2021). Análisis de la seguridad de docker en servidores linux. *Innovación y Desarrollo Tecnológico: Revista Digital*, 13(2):617.

Donca, I.-C., Stan, O. P., Misaros, M., Stan, A., and Miclea, L. (2024). Comprehensive security for iot devices with kubernetes and raspberry pi cluster. *Electronics*, 13(1613).

El Amin, H., Samhat, A. E., Chamoun, M., Oueidat, L., and Feghali, A. (2024). An integrated approach to cyber risk management with cyber threat intelligence framework to secure critical infrastructure. *Journal of Cybersecurity and Privacy*, 4(2):357–381.

Flauzac, O., Mauhourat, F., and Nolot, F. (2020). A review of native container security for running applications. *Procedia Computer Science*, 175:157–164.

García Herrera, E. G. and Cuenca Tapia, J. P. (2021). Guía de implementación de buenas prácticas de seguridad en redes. caso de estudio infocentros mintel. *Dominio de las Ciencias*, 7(4):377–398.

Herzog, P. (2010). *The Open Source Security Testing Methodology Manual* (*OSSTMM 3*). ISECOM - Institute for Security and Open Methodologies. Recuperado de `https://www.isecom.org/OSSTMM.3.pdf`.

Jabr, I., Salman, Y., Shqair, M., and Hawash, A. (2024). Penetration testing and attack automation simulation: Deep reinforcement learning approach. *An-Najah University Journal for Research { A* (*Natural Sciences*), 39(1):7–14.

Kaiser, S., Haq, M. S., Tosun, A. , and Korkmaz, T. (2022). Container technologies for arm architecture: A comprehensive survey of the state of the art. *IEEE Access*, 10:84853–84872.

Mikrotik (2025a). Mikrotik routers and wireless|products: Rb5009ug+s+in. Recuperado 24 de mayo de 2025 de `https://mikrotik.com/product/rb5009ug_s_in`.

Mikrotik (2025b). Ros-200525-1501-900 [manual técnico]. Recuperado de `https://box.mikrotik.com/d/1a069dba20724f279e30/files/?p=%2FROS-200525-1501-900.pdf`.

Mills, A., White, J., and Legg, P. (2023). Longitudinal risk-based security assessment of docker software container images. *Computers & Security*, 135:103478.

Nkengereye, L., Lee, B. G., and Chung, W.-Y. (2025). Functionality-aware offloading technique for scheduling containerized edge applications in iot edge computing. *Journal of Cloud Computing: Advances, Systems and Applications*, 14(13).

Ochoa Villanueva, C. A. and Roman Gonzalez, A. (2023). Implementation of a radius server for access control through authentication in wireless networks. *International Journal of Advanced and Applied Sciences*, 10(3):183–188.

Palate, B. M. and Avila, D. (2021). Mitigación de vulnerabilidades en la red central de un isp: Un caso de estudio. *Ecuadorian Science Journal*, 5(2):68–82.

Palma, C. M. V. and Carrillo, J. M. (2022). Metodologías de testeo de redes de datos. *Revista Cientffica Sinapsis*, 21(1).

Prakosa, B. A., Afrianto, Y., Agustiyan, S., and Setiadi, I. H. (2024). Evaluating bandwidth management techniques on mikrotik routers: A multiple linear regression approach. *Ingénierie Des Systèmes d'Information*, 29(4):1561–1572.

Wang, K., Wu, S., Cui, Y., Huang, Z., Fan, H., and Jin, H. (2024). System log isolation for containers. *Frontiers of Computer Science*, 19(195106).

Wijayanto, A., Riadi, I., Prayudi, Y., and Sudinugraha, T. (2022). Network forensics against address resolution protocol spoofing attacks using trigger, acquire, analysis, report, action method. *Jurnal Ilmiah Teknologi Sistem Informasi*, 8(2):156–169.