Redis Security

This document provides an introduction to the topic of security from the point of view of Redis: the access control provided by Redis, code security concerns, attacks that can be triggered from the outside by selecting malicious inputs and other similar topics are covered.

For security related contacts please open an issue on GitHub, or when you feel it is really important that the security of the communication is preserved, use the GPG key at the end of this document.

Redis general security model

Redis is designed to be accessed by trusted clients inside trusted environments. This means that usually it is not a good idea to expose the Redis instance directly to the internet or, in general, to an environment where untrusted clients can directly access the Redis TCP port or UNIX socket.

For instance, in the common context of a web application implemented using Redis as a database, cache, or messaging system, the clients inside the front-end (web side) of the application will query Redis to generate pages or to perform operations requested or triggered by the web application user.

In this case, the web application mediates access between Redis and untrusted clients (the user browsers accessing the web application).

This is a specific example, but, in general, untrusted access to Redis should always be mediated by a layer implementing ACLs, validating user input, and deciding what operations to perform against the Redis instance.

Network security

Access to the Redis port should be denied to everybody but trusted clients in the network, so the servers running Redis should be directly accessible only by the computers implementing the application using Redis.

In the common case of a single computer directly exposed to the internet, such as a virtualized Linux instance (Linode, EC2, ...), the Redis port should be firewalled to prevent access from the outside. Clients will still be able to access Redis using the loopback interface.

Note that it is possible to bind Redis to a single interface by adding a line like the following to the **redis.conf** file:

```
bind 127.0.0.1
```

Failing to protect the Redis port from the outside can have a big security impact because of the nature of Redis. For instance, a single FLUSHALL command can be used by an external attacker to delete the whole data set.

Protected mode

Unfortunately many users fail to protect Redis instances from being accessed from external networks. Many instances are simply left exposed on the internet with public IPs. For this reasons since version 3.2.0, when Redis is executed with the default configuration (binding all the interfaces) and without any password in order to access it, it enters a special mode called **protected mode**. In this mode Redis only replies to queries from the loopback interfaces, and reply to other clients connecting from other addresses with an error, explaining what is happening and how to configure Redis properly.

We expect protected mode to seriously decrease the security issues caused by unprotected Redis instances executed without proper administration, however the system administrator can still ignore the error given by Redis and just disable protected mode or manually bind all the interfaces.

Authentication feature

While Redis does not try to implement Access Control, it provides a tiny layer of authentication that is optionally turned on editing the **redis.conf** file.

When the authorization layer is enabled, Redis will refuse any query by unauthenticated clients. A client can authenticate itself by sending the **AUTH** command followed by the password.

The password is set by the system administrator in clear text inside the redis.conf file. It should be long enough to prevent brute force attacks for two reasons:

- Redis is very fast at serving queries. Many passwords per second can be tested by an external client.
- The Redis password is stored inside the **redis.conf** file and inside the client configuration, so it does not need to be remembered by the system administrator, and thus it can be very long.

The goal of the authentication layer is to optionally provide a layer of redundancy. If firewalling or any other system implemented to protect Redis from external attackers fail, an external client will still not be able to access the Redis instance without knowledge of the authentication password.

The AUTH command, like every other Redis command, is sent unencrypted, so it does not protect against an attacker that has enough access to the network to perform

eavesdropping.

TLS support

Redis has optional support for TLS on all communication channels, including client connections, replication links and the Redis Cluster bus protocol.

Disabling of specific commands

It is possible to disable commands in Redis or to rename them into an unguessable name, so that normal clients are limited to a specified set of commands.

For instance, a virtualized server provider may offer a managed Redis instance service. In this context, normal users should probably not be able to call the Redis **CONFIG** command to alter the configuration of the instance, but the systems that provide and remove instances should be able to do so.

In this case, it is possible to either rename or completely shadow commands from the command table. This feature is available as a statement that can be used inside the redis.conf configuration file. For example:

```
rename-command CONFIG b840fc02d524045429941cc15f59e41cb7be6c52
```

In the above example, the **CONFIG** command was renamed into an unguessable name. It is also possible to completely disable it (or any other command) by renaming it to the empty string, like in the following example:

```
rename-command CONFIG ""
```

Attacks triggered by carefully selected inputs from external clients

There is a class of attacks that an attacker can trigger from the outside even without external access to the instance. An example of such attacks are the ability to insert data into Redis that triggers pathological (worst case) algorithm complexity on data structures implemented inside Redis internals.

For instance an attacker could supply, via a web form, a set of strings that are known to hash to the same bucket into a hash table in order to turn the O(1) expected time (the average time) to the O(N) worst case, consuming more CPU than expected, and ultimately causing a Denial of Service.

To prevent this specific attack, Redis uses a per-execution pseudo-random seed to the hash function.

Redis implements the SORT command using the qsort algorithm. Currently, the algorithm is not randomized, so it is possible to trigger a quadratic worst-case behavior by carefully selecting the right set of inputs.

String escaping and NoSQL injection

The Redis protocol has no concept of string escaping, so injection is impossible under normal circumstances using a normal client library. The protocol uses prefixed-length strings and is completely binary safe.

Lua scripts executed by the EVAL and EVALSHA commands follow the same rules, and thus those commands are also safe.

While it would be a very strange use case, the application should avoid composing the body of the Lua script using strings obtained from untrusted sources.

Code security

In a classical Redis setup, clients are allowed full access to the command set, but accessing the instance should never result in the ability to control the system where Redis is running.

Internally, Redis uses all the well known practices for writing secure code, to prevent buffer overflows, format bugs and other memory corruption issues. However, the ability to control the server configuration using the **CONFIG** command makes the client able to change the working dir of the program and the name of the dump file. This allows clients to write RDB Redis files at random paths, that is a security issue that may easily lead to the ability to compromise the system and/or run untrusted code as the same user as Redis is running.

Redis does not requires root privileges to run. It is recommended to run it as an unprivileged *redis* user that is only used for this purpose. The Redis authors are currently investigating the possibility of adding a new configuration parameter to prevent **CONFIG SET/GET dir** and other similar run-time configuration directives. This would prevent clients from forcing the server to write Redis dump files at arbitrary locations.

GPG key

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.4.13 (Darwin)

mQINBFJ7ouABEAC5HwiDmE+tRCsWyTaPLBFEGDHcWOLWzph5HdrRtB//UUlSVt9P
tTWZpDvZQvq/ujnS2i2c54V+9NcgVqsCEpA0uJ/U1sUZ3RVBGfGO/l+BIMBnM+B+
TzK825TxER57ILeT/2ZNSebZ+xHJf2Bgbun45pq3KaXUrRnuS8HWSysC+XyMoXET
nksApwMmFWEPZy62gbeayf1U/4yxP/YbHfwSaldpEILOKmsZaGp8PAtVYMVYHsie
gOUdS/jO0P3silagq39cPQLiTMSsyYouxaagbmtdbwINUX0cjtoeKddd4AK7PIww
7su/lhqHZ58ZJdlApCORhXPaDCVrXp/uxAQfT2HhEGCJDTpctGyKMFXQbLUhSuzf
```

```
IilRKJ4jqjcwy+h5lCfDJUvCNYfwyYApsMCs6OWGmHRd7QSFNSs335wAEbVPpO1n
oBJHtOLywZFPF+qAm3LPV4a0OeLyA260c05QZYO59itakjDCBdHwrwv3EU8Z8hPd
6pMNLZ/H1MNK/wWDVeSL8ZzVJabSPTfADXpc1NSwPPWSETS7JYWssdoK+lXMw5vK
q2mSxabL/y91sQ5uscEDzDyJxEPlToApyc5qOUiqQj/thlA6FYBlo1uuuKrpKU1I
e6AA3Gt3fJHXH9TlIcO6DoHvd5fS/o7/RxyFVxqbRqjUoSKQeBzXos3u+QARAQAB
tChTYWx2YXRvcmUgU2FuZmlsaXBwbyA8YW50aXJlekBnbWFpbC5jb20+iQI+BBMB
AgAoBQJSe6LgAhsDBQld/A8ABgsJCAcDAgYVCAIJCgsEFgIDAQIeAQIXgAAKCRAx
gTcoDlyI1riPD/oDDvyIVHtgHvdHqB8/GnF2EsaZgbNuwbiNZ+ilmqnjXzZpu5Su
kGPXAAo+v+rJVLSU2rjCUoL5PaoSlhznw5PL1xpBosN9QzfynWLvJE42T4i0uNU/
a7a1PQCluShnBchm4Xnb3ohNVthFF2MGFRT4OZ5VvK7UcRLYTZoGRlKRGKi9HWea
2xFvyUd9jSuGZG/MMuoslgEPxei09rhDrKxnDNQzQZQpamm/42MITh/1dzEC5ZRx
8hgh1J70/c+zEU7s6kVSGvmYtqbV49/YkqAbhENIeZQ+bCxcTpojEhfk6HoQkXoJ
oK5m21BkMlUEvf1oTX22c0tuOrAX8k0y1M5oismT2e3bqs2OfezNsSfK2gKbeASk
CyYivnbTjmOSPbkvtb27nDqXjb051q6m2A5d59KHfey8BZVuV9j35Ettx4nrS1Ni
S7QrHWRvqceRrIrqXJKopyetzJ6kYDlbP+EVN9NJ2kz/WG6ermltMJQoC0oMhwAG
dfrttG+QJ8PCOlaYiZLD2bjzkDfdfanE74EKYWt+cseenZUf0tsncltRbNdeGTQb
1/GHfwJ+nbA1uKhcHCQ2WrEeGiYpvwKv2/nxBWZ3gwaiAwsz/kI6DQlPZqJoMea9
8gDK2rQigMgbE88vIli4sNhc0yAtm3AbNgAO28NUhzIitB+av/xYxN/W/LkCDQRS
e6LgARAAtdfwe05ZQ0TZYAoeAQXxx2mil4XLzj6ycNjj2JCnFgpYxA8m6nf1gudr
C5V7HDlctp0i9i0wXbf07ubt4Szq4v3ihQCnPQKrZZWfRXxqg0/TOXFfkOdeIoXl
Fl+yC5lUaSTJSg21nxIr8pEq/oPbwpdnWdEGSL9wFanfDUNJExJdzxgyPzD6xubc
OIn2KviV9gbFzQfOIkgkl75V7gn/OA5g2SOLOIPzETLCvQYAGY9ppZrkUz+ji+aT
Tg7HBL6zySt1sCCjyBjFFgNF1RZY4ErtFj5bdBGKCuglyZou4o2ETfA8A5NNpu7x
zkls45UmqRTbmsTD2FU8Id77EaXxDz8nrmjz8f646J0rqn9pGnIg6Lc2PV8j7ACm
/xaTH03taIloOBkTs/Cl01XYeloM0KQwrML43TIm3xSE/AyGF9IGTQo3zmv8SnMO
F+Rv7+55QGlSkfIkXUNCUSm1+dJSBnUhVj/RAjxkekG2di+Jh/y8pkSUxPMDrYEa
OtDoiq2G/roXjVQcbOyOrWA2oB58IVuXO6RzMYi6k6BMpcbmQm0y+TcJqo64tREV
tjogZeIeYDu31eylwijwP67dtbWgiorrFLm2F7+povfXjsDBCQTYhjH4mZgV94ri
hYjP7X2YfLV3tvGyjsMhw3/qLlEyx/f/97gdAaosbpGlVjnhqicAEQEAAYkCJQQY
AQIADwUCUnui4AIbDAUJXfwPAAAKCRAxgTcoDlyI1kAND/sGnXTbMvfHd9AOzv7i
hDX15SSeMDBMWC+8jH/XZASQF/zuHk0jZNTJ01VAdpIxHIVb9dxRrZ3bl56BByyI
8m5DKJiIQWVai+pfjKj6C7p44My3KLodjEeR1oOODXXripGzqJTJNqpW5eCrCxTM
yz1rzO1H1wziJrRNc+ACjVBE3eqcxsZkDZhWN1m8StlX40YgmQmID1CC+kRlV+hg
LUlZLWQIFCGo2UJYoIL/xvUT3Sx4uKD4lpOjyApWzU40mGDaM5+SOsYYrT8rdwvk
nd/efspff64meT9PddX1hi7Cdqbq9woQRu6YhGoCtrHyi/kklGF3EZiw0zWehGAR
2pUeCTD28vsMfJ3ZL1mUGiwlFREUZAcjIlwWDG1RjZDJeZ0NV07KH1N1U8L8aFcu
+CObnlwiavZxOR2yKvwkqmu9c7iXi/R7SVcGQlNao5CWINdzCLHj6/6drPQfGoBS
K/w4JPe7fqmIonMR6O1Gmgkq3Bwl3rz6MWIBN6z+LuUF/b3ODY9rODsJGp21dl2q
xCedf//PAyFnxBNf5NSjyEoPQajKfplfVS3mG8USkS2pafyq6RK9M5wpBR9I1Smm
gon60uMJRIZbxUjQMPLOViGNXbPIilny3FdqbUgMieTBDxrJkE7mtkHfuYw8bERy
vI1sAEeV6ZM/uc4CDI3E2TxEbQ==
```

Key fingerprint

```
pub   4096R/0E5C88D6 2013-11-07 [expires: 2063-10-26]
      Key fingerprint = E5F3 DA80 35F0 2EC1 47F9  020F 3181 3728 0E5C
      uid                  Salvatore Sanfilippo <antirez@gmail.com>
      sub   4096R/3B34D15F 2013-11-07 [expires: 2063-10-26]
```
