

How to Secure Your Redis Installation on Ubuntu 18.04

Posted June 27, 2018 6.3k REDIS SECURITY UBUNTU UBUNTU 18.04



By: Mark Drake

Introduction

Redis is an in-memory, NoSQL, key-value cache and store that can also be persisted to disk. It was designed for use by *trusted clients* in a *trusted environment*, with no robust security features of its own. To underscore that point, here's a quote from the [official Redis website](#):

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.

...

In general, Redis is not optimized for maximum security but for maximum performance and simplicity.

Despite this, Redis does have a few basic security features built into it. These include the ability to create a unencrypted password and the freedom to rename and disable commands. Notably, it lacks a true access control system.

These features cannot, on their own, ensure the security of your Redis Installation. However, configuring them is still a big step up from leaving your database completely unsecured.

In this tutorial, you'll read how to configure the few security features Redis has, and make a few changes to your system's configuration which will boost the security posture of a standalone Redis installation on an Ubuntu server.

Note that this guide does not address situations where the Redis server and the client applications are on different hosts or in different data centers. Installations where Redis traffic has to traverse an insecure or untrusted network require an entirely different set of configurations, such as setting up an [VPN](#) between the Redis machines, in addition to the ones given here.

Prerequisites

For this tutorial, you'll need:

- An Ubuntu 18.04 server. This server should also have a non-root user with `sudo` privileges as well as a firewall set up with UFW, which you can configure by following our [Initial Server Setup Guide for Ubuntu 18.04](#).
- Redis installed and configured on your server. You can set this up by following [this guide for Ubuntu 18.04 servers](#).

Step 1 — Verifying that Redis is Running

First, SSH in to your server with your non-root user.

To check that Redis is working, open up a Redis command line with the `redis-cli` command:

```
$ redis-cli
```

Note: If you already set a password for Redis, you have to authenticate with the `auth` command after connecting:

```
127.0.0.1:6379> auth your_redis_password
```

Output

```
OK
```

If you have not set a password for Redis, you can read how to do so in Step 4 of this tutorial.

Test the connection with the `ping` command:

```
127.0.0.1:6379> ping
```

If Redis is working correctly, you will see the following:

Output

```
PONG
```

Following this, exit the Redis command line:

```
127.0.0.1:6379> quit
```

Now that you've confirmed that Redis is running and working correctly, you can move on to the most important step for enhancing your server's security: configuring a firewall.

Step 2 — Securing the Server with UFW

Redis is just an application that's running on your server. Because it has only a few fundamental security features of its own, the first step to truly securing it is to secure the server it's running on. In the case of a public-facing server like your Ubuntu 18.04 server, configuring a firewall as described in the [Initial Server Setup Guide for Ubuntu 18.04](#) is that first step. **Follow that link and set up your firewall now if you haven't already done so.**

If you aren't sure whether you set up a firewall or if it's active, you can check this by running the following:

```
$ sudo ufw status
```

If you followed the Initial Server Setup Guide for Ubuntu 18.04, you will see the following output:

Output

```
Status: active
```

To	Action	From
--	-----	----
OpenSSH	ALLOW	Anywhere
OpenSSH (v6)	ALLOW	Anywhere (v6)

If you've implemented the firewall rules using that guide, then you do not need to add an extra rule for Redis because, by default, UFW drops all incoming traffic unless it is explicitly allowed. Since a default standalone installation of the Redis server is listening only on the loopback interface (`127.0.0.1` , or **localhost**), there should be no concern for incoming traffic on its default port.

For more information on how to add rules, please see this [guide on common UFW rules and commands](#).

Step 3 — Binding to localhost

By default, Redis is only accessible from **localhost**. However, if you followed a different tutorial to configure Redis than the one given in the prerequisites section, you might have updated the configuration file to allow connections from anywhere. This is not as secure as binding to **localhost**.

Open the Redis configuration file for editing:

```
$ sudo nano /etc/redis/redis.conf
```

SCROLL TO TOP

Locate this line and make sure it is uncommented (remove the `#` if it exists):

```
/etc/redis/redis.conf
```

```
bind 127.0.0.1
```

Save and close the file when finished (press `CTRL + X`, `Y`, then `ENTER`).

Then, restart the service to ensure that `systemd` reads your changes:

```
$ sudo systemctl restart redis
```

To check that this change has gone into effect, run the following `netstat` command:

```
$ sudo netstat -ltn | grep redis
```

Output

```
tcp        0      0 127.0.0.1:6379          0.0.0.0:*               LISTEN      2855/redis-server 1
```

This output shows that the `redis-server` program is bound to **localhost** (`127.0.0.1`), reflecting the change you just made to the configuration file. If you see another IP address in that column (`0.0.0.0`, for example), then you should double check that you uncommented the correct line and restart the Redis service again.

Now that your Redis installation is only listening in on **localhost**, it will be more difficult for malicious actors to make requests or gain access to your server. However, Redis isn't currently set to require users to authenticate themselves before making changes to its configuration or the data it holds. To remedy this, Redis allows you to require users to authenticate with a password before making changes via the Redis client (`redis-cli`).

Step 4 — Configuring a Redis Password

Configuring a Redis password enables one of its two built-in security features — the `auth` command, which requires clients to authenticate to access the database. The password is configured directly in Redis's configuration file, `/etc/redis/redis.conf`, so open that file again with your preferred editor:

```
$ sudo nano /etc/redis/redis.conf
```

Scroll to the `SECURITY` section and look for a commented directive that reads:

```
/etc/redis/redis.conf
```

```
# requirepass foobared
```

[SCROLL TO TOP](#)

Uncomment it by removing the #, and change foobared to a secure password.

Note: Above the `requirepass` directive in the `redis.conf` file, there is a commented warning:

```
/etc/redis/redis.conf
```

```
# Warning: since Redis is pretty fast an outside user can try up to
# 150k passwords per second against a good box. This means that you should
# use a very strong password otherwise it will be very easy to break.
#
```

Thus, it's important that you specify a very strong and very long value as your password. Rather than make up a password yourself, you can use the `openssl` command to generate a random one, as in the following example. The pipe to the second `openssl` command will remove any line breaks that are output by the first command:

```
$ openssl rand 60 | openssl base64 -A
```

Your output should look something like:

Output

```
RBOJ9cCNoGCKhlEBwQLHri1g+atWgn4Xn4HwNUbtzoVxAYxkiYBi7aufl4MILv1nxBqR4L6NNzI0X6cE
```

After copying and pasting the output of that command as the new value for `requirepass`, it should read:

```
/etc/redis/redis.conf
```

```
requirepass RBOJ9cCNoGCKhlEBwQLHri1g+atWgn4Xn4HwNUbtzoVxAYxkiYBi7aufl4MILv1nxBqR4L6NNzI0X6cE
```

After setting the password, save the file, and restart Redis:

```
$ sudo systemctl restart redis.service
```

To test that the password works, access the Redis command line:

```
$ redis-cli
```

The following shows a sequence of commands used to test whether the Redis password works. The first command tries to set a key to a value before authentication:

```
127.0.0.1:6379> set key1 10
```

That won't work because you didn't authenticate, so Redis returns an error:

Output

```
(error) NOAUTH Authentication required.
```

The next command authenticates with the password specified in the Redis configuration file:

```
127.0.0.1:6379> auth your_redis_password
```

Redis acknowledges:

Output

```
OK
```

After that, running the previous command again will succeed:

```
127.0.0.1:6379> set key1 10
```

Output

```
OK
```

`get key1` queries Redis for the value of the new key.

```
127.0.0.1:6379> get key1
```

Output

```
"10"
```

After confirming that you're able to run commands in the Redis client after authenticating, you can exit the `redis-cli`:

```
127.0.0.1:6379> quit
```

Next, we'll look at renaming Redis commands which, if entered by mistake or by a malicious actor, could cause serious damage to your machine.

Step 5 — Renaming Dangerous Commands

The other security feature built into Redis involves renaming or completely disabling certain commands that are considered dangerous.

When run by mistake or by unauthorized users, such commands can be used to reconfigure, destroy, or otherwise wipe your data. Like the authentication password, renaming or disabling commands is configured in the same `SECURITY` section of the `/etc/redis/redis.conf` file.

Some of the commands that are considered dangerous include: **FLUSHDB**, **FLUSHALL**, **KEYS**, **PEXPIRE**, **DEL**, **CONFIG**, **SHUTDOWN**, **BGREWRITEAOF**, **BGSAVE**, **SAVE**, **SPOP**, **SREM**, **RENAME**, and **DEBUG**. This is not a comprehensive list, but renaming or disabling all of the commands in that list is a good starting point for enhancing your Redis server's security.

Whether you should disable or rename a command depends on your specific needs or those of your site. If you know you will never use a command that could be abused, then you may disable it. Otherwise, it could be in your best interest to rename it.

To enable or disable Redis commands, open the configuration file once more:

```
$ sudo nano /etc/redis/redis.conf
```

Warning: The following steps showing how to disable and rename commands are examples. You should only choose to disable or rename the commands that make sense for you. You can review the full list of commands for yourself and determine how they might be misused at redis.io/commands.

To disable a command, simply rename it to an empty string (signified by a pair of quotation marks with no other characters between them), as shown below:

```
/etc/redis/redis.conf

. . .
# It is also possible to completely kill a command by renaming it into
# an empty string:
#
rename-command FLUSHDB ""
rename-command FLUSHALL ""
rename-command DEBUG ""
. . .
```

To rename a command, give it another name as shown in the examples below. Renamed commands should be difficult for others to guess, but easy for you to remember:

```
/etc/redis/redis.conf
```

SCROLL TO TOP

```
. . .
rename-command CONFIG ""
rename-command SHUTDOWN SHUTDOWN_MENOT
rename-command CONFIG ASC12_CONFIG
. . .
```

Save your changes and close the file.

After renaming a command, apply the change by restarting Redis:

```
$ sudo systemctl restart redis.service
```

To test the new command, enter the Redis command line:

```
$ redis-cli
```

Next, authenticate:

```
127.0.0.1:6379> auth your_redis_password
```

Output

```
OK
```

Let's assume that you renamed the `CONFIG` command to `ASC12_CONFIG`, as in the preceding example. First, try using the original `CONFIG` command. It should fail, because you've renamed it:

```
127.0.0.1:6379> config get requirepass
```

Output

```
(error) ERR unknown command 'config'
```

Calling the renamed command, however, will be successful. It is not case-sensitive:

```
127.0.0.1:6379> asc12_config get requirepass
```

Output

```
1) "requirepass"
2) "your_redis_password"
```


Finally, you can exit from `redis-cli`:

```
127.0.0.1:6379> exit
```

Note that if you're already using the Redis command line and then restart Redis, you'll need to re-authenticate. Otherwise, you'll get this error if you type a command:

Output

```
NOAUTH Authentication required.
```

Regarding the practice of renaming commands, there's a cautionary statement at the end of the `SECURITY` section in `/etc/redis/redis.conf` which reads:

Please note that changing the name of commands that are logged into the AOF file or transmitted to slaves may cause problems.

Note: *The Redis project chooses to use the terms “master” and “slave,” while DigitalOcean generally prefers the alternatives “primary” and “secondary.” In order to avoid confusion we’ve chosen to use the terms used in the Redis documentation [here](#).*

That means if the renamed command is not in the AOF file, or if it is but the AOF file has not been transmitted to slaves, then there should be no problem.

So, keep that in mind when you're trying to rename commands. The best time to rename a command is when you're not using AOF persistence, or right after installation, that is, before your Redis-using application has been deployed.

When you're using AOF and dealing with a master-slave installation, consider [this answer from the project's GitHub issue page](#). The following is a reply to the author's question:

The commands are logged to the AOF and replicated to the slave the same way they are sent, so if you try to replay the AOF on an instance that doesn't have the same renaming, you may face inconsistencies as the command cannot be executed (same for slaves).

Thus, the best way to handle renaming in cases like that is to make sure that renamed commands are applied to all instances in master-slave installations.

Conclusion

Keep in mind that once someone is logged in to your server, it's very easy to circumvent security features we've put in place. Therefore, the most important security feature is your firewall, which

[SCROLL TO TOP](#)

makes it extremely difficult for malicious actors to jump that fence.

If you're attempting to secure Redis communication across an untrusted network you'll have to employ an SSL proxy, as recommended by Redis developers in the [official Redis security guide](#).

By: Mark Drake

 Upvote (2)  Subscribe  Share



We just made it easier for you to deploy faster.

[TRY FREE](#)

Related Tutorials

How To Install and Secure Memcached on Ubuntu 18.04

How To Secure a Containerized Node.js Application with Nginx, Let's Encrypt, and Docker Compose

How to Set Up an Nginx Ingress with Cert-Manager on DigitalOcean Kubernetes

How To Secure Nginx with NAXSI on Ubuntu 16.04

How To Set Up an OpenVPN Server on Debian 9

1 Comment

Leave a comment...

SCROLL TO TOP

Log In to Comment

^ [ballonposte](#) January 11, 2019

0 I basically understand the problem with the practice of renaming commands and the persistence of the old commands but I'm unable to solve it.
Any solutions ?



This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.



Copyright © 2019 DigitalOcean™ Inc.

[Community](#) [Tutorials](#) [Questions](#) [Projects](#) [Tags](#) [Newsletter](#) [RSS](#) 

[Distros & One-Click Apps](#) [Terms, Privacy, & Copyright](#) [Security](#) [Report a Bug](#) [Write for DOnations](#) [Shop](#)

SCROLL TO TOP