

How To Use Certbot Standalone Mode to Retrieve Let's Encrypt SSL Certificates on Ubuntu 20.04

Step 1 — Installing Certbot

Certbot recommends using their *snap* package for installation. Snap packages work on nearly all Linux distributions, but they require that you've installed `snapt` first in order to manage snap packages. Ubuntu 20.04 comes with support for snaps out of the box, so you can start by making sure your `snapt` core is up to date:

```
sudo snap install core; sudo snap refresh core
```

If you're working on a server that previously had an older version of `certbot` installed, you should remove it before going any further:

```
sudo apt remove certbot
```

After that, you can install the `certbot` package:

```
sudo snap install --classic certbot
```

Finally, you can link the `certbot` command from the snap install directory to your path, so you'll be able to run it by just typing `certbot`. This isn't necessary with all packages, but snaps tend to be less intrusive by default, so they don't conflict with any other system packages by accident:

```
sudo ln -s /snap/bin/certbot /usr/bin/certbot
```

Now that we have Certbot installed, let's run it to get our certificate.

Step 2 — Running Certbot

Certbot needs to answer a cryptographic challenge issued by the Let's Encrypt API in order to prove we control our domain. It uses ports `80` (HTTP) or `443` (HTTPS) to accomplish this. Open up the appropriate port(s) in your firewall:

```
sudo ufw allow 443
```

Output:

```
Rule added
Rule added (v6)
```

We can now run Certbot to get our certificate. We'll use the `--standalone` option to tell Certbot to handle the challenge using its own built-in web server. Finally, the `-d` flag is used to specify the domain you're requesting a certificate for. You can add multiple `-d` options to cover multiple domains in one certificate.

```
sudo certbot certonly --standalone -d your_domain
```

When running the command, you will be prompted to enter an email address and agree to the terms of service. After doing so, you should see a message telling you the process was successful and where your certificates are stored:

Output:

```
IMPORTANT NOTES:
Successfully received certificate.
Certificate is saved at: /etc/letsencrypt/live/your_domain/fullchain.pem
Key is saved at: /etc/letsencrypt/live/your_domain/privkey.pem
This certificate expires on 2022-02-10.
These files will be updated when the certificate renews.
Certbot has set up a scheduled task to automatically renew this certificate in the background.
```

If you like Certbot, please consider supporting our work by:

* Donating to ISRG / Let's Encrypt: <https://letsencrypt.org/donate>

* Donating to EFF: <https://eff.org/donate-le>

You should now have your certificates. In the next step, we will inspect some of the files that we downloaded and learn about their functionality.

Step 3 — Configuring Your Application

Configuring your application for SSL is beyond the scope of this article, as each application has different requirements and configuration options, but let's take a look at what Certbot has downloaded for us. Use `ls` to list out the directory that holds our keys and certificates:

```
sudo ls /etc/letsencrypt/live/your_domain
```

Output:

```
cert.pem chain.pem fullchain.pem privkey.pem README
```

The `README` file in this directory has more information about each of these files. Most often you'll only need two of these files:

- `privkey.pem`: This is the private key for the certificate. This needs to be kept safe and secret, which is why most of the `/etc/letsencrypt` directory has very restrictive permissions and is accessible by only the **root** user. Most software configuration will refer to this as something similar to `ssl-certificate-key` or `ssl-certificate-key-file`.
- `fullchain.pem`: This is our certificate, bundled with all intermediate certificates. Most software will use this file for the actual certificate, and will refer to it in their configuration with a name like 'ssl-certificate'.

For more information on the other files present, refer to the “[Where are my certificates” section of the Certbot docs.

Some software will need its certificates in other formats, in other locations, or with other user permissions. It is best to leave everything in the `letsencrypt` directory, and not change any permissions in there (permissions will just be overwritten upon renewal anyway), but sometimes that's just not an option. In that case, you'll need to write a script to move files and change permissions as needed. This script will need to be run whenever Certbot renews the certificates, which we'll talk about next.

Step 4 — Handling Certbot Automatic Renewals

Let's Encrypt's certificates are only valid for ninety days. This is to encourage users to automate their certificate renewal process. The `certbot` package we installed takes care of this for us by adding a renew script to `/etc/cron.d`. This script runs twice a day and will renew any certificate that's within thirty days of expiration.

With our certificates renewing automatically, we still need a way to run other tasks after a renewal. We need to at least restart or reload our server to pick up the new certificates, and as mentioned in Step 3 we may need to manipulate the certificate files in some way to make them work with the software we're using. This is the purpose of Certbot's `renew_hook` option.

To add a `renew_hook`, we update Certbot's renewal config file. Certbot remembers all the details of how you first fetched the certificate, and will run with the same options upon renewal. We just need to add in our hook. Open the config file with your favorite editor:

```
sudo nano /etc/letsencrypt/renewal/your_domain.conf
```

A text file will open with some configuration options. You can add a hook on the last line that will reload any web-facing services, making them use the renewed certificate:

```
/etc/letsencrypt/renewal/<^>your_domain<^>.conf
```

```
renew_hook = systemctl reload your_service
```

Update the command above to whatever you need to run to reload your server or run your custom file munging script. Usually, on Ubuntu, you'll mostly be using `systemctl` to reload a service. Save and close the file, then run a Certbot dry run to make sure the syntax is ok:

```
sudo certbot renew --dry-run
```

If you see no errors, you're all set. Certbot is set to renew when necessary and run any commands needed to get your service using the new files.

Conclusion

In this tutorial, we've installed the Certbot Let's Encrypt client, downloaded an SSL certificate using standalone mode, and enabled automatic renewals with renew hooks. This should give you a good start on using Let's Encrypt certificates with services other than your typical web server.

Revision #1

Created 11 August 2024 14:15:28 by Admin

Updated 11 August 2024 14:21:55 by Admin