
SecureDrop Developer Documentation

Release latest

SecureDrop Team and Contributors

Sep 13, 2022

GETTING STARTED

1	Contributing to SecureDrop	3
1.1	Programmers	3
1.2	Technical Writers	5
1.3	UX Contributors	5
1.4	Release Managers	5
1.5	Translators	5
1.6	Forum Moderators and Support	6
2	Setting Up the Development Environment	7
2.1	Overview	7
2.2	Quick Start	7
2.3	Setting Up a Multi-Machine Environment	10
3	Making a PR to SecureDrop	13
3.1	Forking and Cloning the Project	13
3.2	Make Your Changes and Push to the Fork	13
3.3	Making a Pull Request to Get Your Changes Merged in develop Branch	14
4	Translations	15
4.1	Quick Start Guide	15
4.2	Background Information	16
4.3	How-to Guides	17
4.4	Glossary	24
5	Contributing Guidelines	27
5.1	Signing commits	27
5.2	Branching Strategy	27
5.3	Automated Testing	27
5.4	Code Style	28
5.5	Type Hints in Python code	29
5.6	Git History	30
5.7	Privileges	30
5.8	Other Tips	31
6	Tips & Tricks	33
6.1	Using Tor Browser with the Development Environment	33
6.2	Upgrading or Adding Python Dependencies	33
6.3	Architecture Diagrams	34
7	Journalist Interface API	37
7.1	Versioning	37

7.2	Content Type	37
7.3	Authentication	37
7.4	Errors	38
7.5	Endpoints	38
7.6	Removed functionality	52
8	Development of Securedrop-Admin in the Admin Directory	53
9	Development of SecureDropUpdater in the journalist_gui Directory	55
9.1	Installing the Dependencies in a Virtual Environment	55
9.2	To Update the UI Design	55
9.3	Using Resources in the UI	56
9.4	Adding and Running Test Cases	56
10	Virtual Environments: Servers	57
10.1	Staging	57
10.2	Production	59
11	Virtual Environments: Admin Workstation	63
11.1	Linux	63
12	Virtual Environments: Using Qubes	65
12.1	Overview	65
12.2	Download Ubuntu server ISO	65
12.3	Create the base VM	65
12.4	Boot into installation media	66
12.5	Initial VM configuration	66
12.6	Clone VMs	67
12.7	SecureDrop Installation	68
12.8	Managing Qubes RPC for Admin API capability	69
12.9	Creating staging instance	69
12.10	Accessing the Journalist Interface (Staging) in Whonix-based VMs	70
13	Upgrade Testing using Molecule	71
13.1	Upgrade testing using locally-built packages	71
13.2	Upgrade testing using apt-test.freedom.press	72
14	Database Migrations	73
14.1	Migration Files	73
14.2	Deployment	73
14.3	Developer Workflow	74
15	Internationalization (i18n)	77
15.1	i18n_tool.py	78
15.2	Development tasks	78
15.3	Release Management	83
15.4	Weblate administration	85
16	Documentation Guidelines	87
16.1	Documentation versions	87
16.2	Updating Documentation	87
16.3	Testing Documentation Changes	88
16.4	Pushing to a contributor fork	88
16.5	Updating Screenshots	88
16.6	Updating Upgrade Guides	89

16.7	Style Guide	89
17	Testing SecureDrop	93
18	Testing: Application Tests	95
18.1	Installation	95
18.2	Running the Application Tests	95
18.3	Updating the Application Tests	96
19	Testing: Configuration Tests	97
19.1	Installation	97
19.2	Running the Config Tests	97
19.3	Updating the Config Tests	97
19.4	Config Test Layout	98
19.5	Config Testing Strategy	98
20	Testing: CI	99
20.1	Running the CI Staging Environment	99
21	SecureDrop apt Repository	103
22	Updating OSSEC Rules	105
22.1	Alerting Strategy	105
22.2	Using <code>ossec-logtest</code>	105
22.3	Writing Automated Tests for OSSEC Rules	106
22.4	Adding new OSSEC rules	106
22.5	Deployment	108
23	Generating AppArmor Profiles for Tor and Apache	109
24	Portable SecureDrop Demo	111
24.1	Hardware	111
25	Release Management	113
25.1	Pre-Release	113
25.2	Release Process	116
25.3	Post-Release	118
26	Build container	119
26.1	Who can update the build container?	119
26.2	Updating the build container	119
27	Updating Tor	121
27.1	Identifying new releases	121
27.2	Testing	121
27.3	Promoting	121
28	Developing the SecureDrop Client Application	123
28.1	Developer Setup	123
28.2	How to Find Help	123
28.3	Client Architecture	123
28.4	Client Database Structure	124
28.5	Tests	126
28.6	Contributing	126

This documentation is intended for contributors to the SecureDrop project. If you are looking for information on how to install, use or maintain SecureDrop, please see our [documentation for sources, journalists and administrators](#).

CONTRIBUTING TO SECUREDROP

Thank you for your interest in contributing to SecureDrop! We welcome both new and experienced open-source contributors and are committed to making it as easy as possible to contribute. Whether you have a few minutes or many hours, there are a variety of ways to help. We are always looking for help from:

- *Programmers*, to help us develop SecureDrop;
- *Technical writers*, to help improve the documentation;
- *UX contributors*, to help improve the product experience for end users;
- *Translators*, to translate SecureDrop;
- *Release managers*, to create and maintain Debian GNU/Linux packages and repositories;
- *Forum moderators and support* volunteers, to help with the support forums.

You can always find a regular project contributor to answer any questions you may have on the [SecureDrop instant messaging channel](#). You can also register on [the forum](#) for more information and to participate in longer discussions.

Note: The SecureDrop GitHub repositories and other project resources are managed by [Freedom of the Press Foundation employees](#). All SecureDrop contributors are required to abide by the project's [Code of Conduct](#).

- To start contributing to the [codebase](#), see our [contributing guidelines](#).
- To start making documentation changes, see our [documentation guidelines](#).
- To start translating, see our [translator guide](#).
- Not sure where to start? You can always ask for advice in the [chat room](#).

1.1 Programmers

The SecureDrop system includes [Flask](#)-based web applications for sources and journalists. It is deployed across multiple machines with [Ansible](#). Most of SecureDrop's code is written in [Python](#).

A contributing programmer can work on either newcomer or advanced developer issues.

1.1.1 Newcomer Issues

If you are a novice programmer, you can start with these issues in the following repositories:

- [SecureDrop](#)
- [SecureDrop Workstation](#)
- [SecureDrop Client](#)

1.1.2 Advanced Issues

Programmers who are more comfortable with contributing to the SecureDrop codebase can work on issues related to the following topics:

Application development and general tasks:

- [Application code cleanup](#)
- [Developer workflow](#)
- [Needs/Research](#)
- [Source and journalist applications](#)
- [Journalist experience](#)
- [Source experience](#)
- [Tests](#)

Infrastructure focus:

- [Continuous Integration](#)
- [Ansible logic/installation](#)
- [Operations and deployment](#)

Security focus:

- [IDS noise](#)
- [OSSEC](#)
- [Security](#)

You may also want to consider contributing to the new [SecureDrop Workstation](#) project and its components, including the graphical [SecureDrop Client](#) app.

1.1.3 Preparing and submitting changes

Before beginning your work on any given issue, we recommend asking questions or sharing an implementation proposal on the relevant GitHub issue. Alternatively, you can often find the development team on [Gitter chat](#). Communicating early and often is especially important for larger changes.

When you're ready to share your work with the SecureDrop team for review, submit a [pull request](#) with the proposed changes. [Tests](#) will run automatically on GitHub.

If you would like to contribute on a regular basis, you'll want to read the [developer documentation](#) and set up a local development environment to preview changes, run tests locally, etc.

1.2 Technical Writers

Technical writers and editors are invited to review the [documentation](#) and fix any mistakes in accordance with the [documentation guidelines](#). Our documentation code is located in our [documentation repository](#).

If this is your first time contributing to SecureDrop documentation, consider working on low-hanging fruit to become familiar with the process.

If you would like to contribute to copywriting user-facing text in the SecureDrop UI, see [these issues](#) in our separate [User Experience repo](#).

1.3 UX Contributors

If you have interaction or visual design skills, UI copywriting skills, or user research skills, check out our [User Experience repository](#). It includes a wiki with notes from UX meetings, design standards, design principles, links to past research synthesis efforts, and ongoing and past work documented in the form of issues.

If you have front-end development skills, take a look at these issues in the primary SecureDrop repository in GitHub:

- [All issues labeled “UX”](#)
- [CSS/SASS and HTML](#)
- [All issues labeled “Journalist Experience”](#)

1.4 Release Managers

All software deployed with SecureDrop is installed via Debian GNU/Linux packages via Ansible. The [primary repository](#) is controlled, maintained, and signed by [Freedom of the Press Foundation employees](#). The current responsibilities of the release manager are covered in [detailed documentation](#).

If you are a [Debian developer](#) you can help improve packaging and the release process:

- [Building SecureDrop application and OSSEC packages and pending bugs and tasks](#)
- [Building grsecurity kernels and pending bugs and tasks](#)

1.5 Translators

Translating SecureDrop is crucial to making it useful for investigative journalism around the world. If you know English and another language, we would welcome your help.

SecureDrop is translated using [Weblate](#). We provide a [detailed guide](#) for translators, and feel free to contact us in the [translation section](#) of the SecureDrop forum for help. Non-English forum discussions are also welcome.



1.6 Forum Moderators and Support

Those running a production instance of SecureDrop are encouraged to [read the support documentation](#) to get help from the [Freedom of the Press Foundation](#). For less sensitive topics such as running a demo or getting help to understand a concept, a [public forum section](#) is better suited. To assist on the forum:

- Look for [the latest unanswered questions in the forum](#) and answer them.
- If you find questions [elsewhere in the forum](#) that have a better chance at getting an answer in the [support section](#), suggest in Gitter to move topics from a category to another.

SETTING UP THE DEVELOPMENT ENVIRONMENT

Note: SecureDrop maintains two documentation versions: [stable](#) and [latest](#). [stable](#) is the default, and is built from our latest signed git tag. [latest](#) is built from the head of the [main git branch](#) of the [securedrop-docs repository](#). In almost all cases involving development work, you'll want to make sure you have the [latest](#) version selected by using the menu in the bottom left corner of the documentation site.

2.1 Overview

SecureDrop is a multi-machine design. To make development and testing easier, we provide a set of virtual environments, each tailored for a specific type of development task. We use Ansible playbooks to provision these environments on either virtual machines or physical hardware. We use Libvirt to manage our virtual machines, Docker to run them, and Molecule to test the provisioning logic.

2.2 Quick Start

The Docker based environment is suitable for developing the web application and updating the documentation. Follow the instructions below to install the requirements for the Docker-based environment for your operating system.

2.2.1 Ubuntu or Debian GNU/Linux

Run the following commands to update the package index and to install Git and `make`:

```
sudo apt-get update
sudo apt-get install -y make git
```

We recommend using the stable version of Docker CE (Community Edition) which can be installed via the official documentation links:

- [Docker CE for Ubuntu](#)
- [Docker CE for Debian](#)

Make sure to follow the [post-installation steps for Linux](#).

Experimental support for using [Podman](#) is available, set the `USE_PODMAN=1` environment variable to enable it.

2.2.2 Fedora Linux

Note: To install Docker Engine, you need the 64-bit version of Fedora 30 or higher.

Run the following command to update the package index and to install Git and make:

```
sudo dnf install -y make git
```

We recommend using the stable version of Docker CE (Community Edition) which can be installed via the official documentation link:

- [Docker CE for Fedora](#)

Make sure to follow the [post-installation steps for Linux](#).

Experimental support for using [Podman](#) is available, set the `USE_PODMAN=1` environment variable to enable it.

2.2.3 macOS

Install [Docker](#).

2.2.4 Qubes

Create a StandaloneVM based on Debian 10, called `sd-dev`. You can use the **Q** menu to configure a new VM, or run the following in `dom0`:

```
qvm-clone --class StandaloneVM debian-10 sd-dev
qvm-start sd-dev
qvm-sync-appmenus sd-dev
```

The commands above create a new StandaloneVM, boot it, and then update the Qubes menus with applications within that VM. Open a terminal in `sd-dev`, and proceed with installing [Docker CE for Debian](#).

Tip: If you experience an error with the `aufs-dkms` dependency when installing Docker CE, you can safely skip that package using the `--no-install-recommends` argument for `apt`.

2.2.5 Fork & Clone the Repository

Now you are ready to get your own copy of the source code. Visit our [repository](#), fork it, and clone it on your local machine.

```
git clone git@github.com:<your_github_username>/securedrop.git
```

Note: Pull requests should be opened against the `develop` branch of our [repository](#), which is the primary branch used for development.

2.2.6 Using the Docker Environment

The Docker based helpers are intended for rapid development on the SecureDrop web application and documentation. They use Docker images that contain all the dependencies required to run the tests, a demo server etc.

Tip: When run for the first time, building Docker images will take a few minutes, even one hour when your Internet connection is not fast. If you are unsure about what happens, you can get a more verbose output by setting the environment variable `export DOCKER_BUILD_VERBOSE=true`.

The SecureDrop repository is bind mounted into the container and files modified in the container are also modified in the repository. This container has no security hardening or monitoring.

To get started, you can try the following:

```
cd securedrop
make dev                                # run development servers
make test                              # run tests
securedrop/bin/dev-shell bin/run-test tests/functional # functional tests only
securedrop/bin/dev-shell bash           # shell inside the container
```

Tip: The interactive shell in the container does not run `redis`, `Xvfb` etc. However you can import shell helper functions with `source bin/dev-deps` and call `run_xvfb`, `maybe_create_config_py` etc.

SecureDrop consists of two separate web applications (the Source Interface and the *Journalist Interface*) that run concurrently. In the development environment they are configured to detect code changes and automatically reload whenever a file is saved. They are made available on your host machine by forwarding the following ports:

- Source Interface: `localhost:8080`
- *Journalist Interface*: `localhost:8081`

You should use Tor Browser to test web application changes, [see here for instructions](#).

A test administrator (`journalist`) and non-admin user (`dellsberg`) are created by default when running `make dev`. In addition, sources and submissions are present. The test users have the following credentials. Note that the password and TOTP secret are the same for both accounts for convenience during development.

- **Username:** `journalist` or `dellsberg`
- **Password:** `correct horse battery staple profanity oil chewy`
- **TOTP secret:** `JHCO G07V CER3 EJ4L`

If you need to generate the six digit two-factor code, use the TOTP secret in combination with an authenticator application that implements [RFC 6238](#), such as [FreeOTP](#) (Android and iOS) or [oathtool](#) (command line tool, multiple platforms). Instead of typing the TOTP code, you can simply scan the following QR code:



You can also generate the two-factor code using the Python interpreter:

```
>>> import pyotp
>>> pyotp.TOTP('JHCOG07VCER3EJ4L').now()
u'422038'
```

2.3 Setting Up a Multi-Machine Environment

Note: You do not need this step if you only plan to work on the web application or the documentation.

To get started, you will need to install Vagrant, Libvirt, Docker, and Ansible on your development workstation.

2.3.1 Ubuntu or Debian GNU/Linux

Note: Tested on: Debian GNU/Linux 10 Buster

```
sudo apt-get update
sudo apt-get install -y build-essential libssl-dev libffi-dev python3-dev \
    dpkg-dev git linux-headers-$(uname -r)
```

We recommend using the most recent version of Vagrant available in your distro's package repositories. For Debian Stable, that's 2.2.3 at the time of this writing. Older versions of Vagrant has been known to cause problems ([GitHub #932](#), [GitHub #1381](#)). If `apt-cache policy vagrant` says your candidate version is not at least 1.8.5, you should download the current version from the [Vagrant Downloads page](#) and then install it.

```
# If your OS vagrant is recent enough
sudo apt-get install vagrant
# OR this, if you downloaded the deb package.
sudo dpkg -i vagrant.deb
```

Warning: We do not recommend installing `vagrant-cachier`. It destroys `apt`'s state unless the VMs are always shut down/rebooted with Vagrant, which conflicts with the tasks in the Ansible playbooks. The instructions in Vagrantfile that would enable `vagrant-cachier` are currently commented out.

Finally, install Ansible so it can be used with Vagrant to automatically provision VMs. We recommend installing Ansible from PyPi with `pip` to ensure you have the latest stable version.

```
sudo apt-get install python3-pip
```

The version of Ansible recommended to provision SecureDrop VMs may not be the same as the version in your distro's repos, or may at some point flux out of sync. For this reason, and also just as a good general development practice, we recommend using a Python virtual environment to install Ansible and other development-related tooling. Using `virtualenvwrapper`:

```
sudo apt-get install virtualenvwrapper
source /usr/share/virtualenvwrapper/virtualenvwrapper.sh
mkvirtualenv -p /usr/bin/python3 securedrop
```

Note: You'll want to add the command to source `virtualenvwrapper.sh` to your `~/.bashrc` (or whatever your default shell configuration file is) so that the command-line utilities `virtualenvwrapper` provides are automatically available in the future.

2.3.2 macOS

Developers on macOS should use the Docker-based container environment. We don't support running VMs on macOS.

2.3.3 Fork & Clone the Repository

Now you are ready to get your own copy of the source code. Visit our [repository](#) fork it and clone it on you local machine:

```
git clone git@github.com:<your_github_username>/securedrop.git
```

2.3.4 Install Python Requirements

SecureDrop uses many third-party open source packages from the Python community. Ensure your virtualenv is activated and install the packages.

```
pip install --no-deps --require-hashes -r securedrop/requirements/python3/develop-  
requirements.txt
```

Note: You will need to run this everytime new packages are added.

2.3.5 Qubes

To configure a multi-machine environment in Qubes, follow the Quick Start instructions above to create a standalone VM named sd-dev, then follow the Linux instructions above to install the required packages.

Then, complete the steps described in *Virtual Environments: Using Qubes*.

MAKING A PR TO SECUREDROP

3.1 Forking and Cloning the Project

1. Fork SecureDrop on GitHub from the Main Repository to your own profile.
2. Clone the forked repository.

```
git clone https://github.com/<your-username>/securedrop.git  
cd securedrop
```

3. Add the Main Repository as an upstream remote.

```
git remote add upstream https://github.com/freedomofpress/securedrop.git
```

3.2 Make Your Changes and Push to the Fork

3.2.1 Create a Branch

Create a branch on which you make your changes.

```
git checkout -B change-one
```

3.2.2 Make Your Changes and Commit

Now enter the directory of your fork and make changes as you wish. Run tests for the changes you have made.

If you create a new file, remember to add it with `git add`.

```
git add <new-file>
```

Commit your changes, adding a description of what was added. If you're not used to Git, the simplest way is to commit all modified files and add a description message of your changes in a single command like this:

```
git commit -a -m "<Description of changes made>"
```

3.2.3 Pull the Upstream Changes

We get any updates made in the upstream repository.

```
git pull upstream develop
```

3.2.4 Rebasing

Rebasing is the process of moving or combining a sequence of commits to a new base commit. Rebasing is most useful and easily visualized in the context of a feature branching workflow.

Assume the following history exists:

```
      A---B---C change-one
      /
D---E---F---G develop
```

From this point, the result of either of the following commands:

```
git rebase develop
git rebase develop change-one
```

would be:

```
      A`--B`--C` change-one
      /
D---E---F---G develop
```

Note: A and A` represents the same set of changes, but have different committer information.

3.2.5 Pushing the Changes to GitHub Fork

Once your changes are committed and rebased, push the changes to your GitHub fork.

```
git push origin <branch-name>
```

3.3 Making a Pull Request to Get Your Changes Merged in develop Branch

1. Through GitHub make a pull request from the branch that you committed your code to.
2. Once PR is made, the Circle CI build server checks all tests and Codecov runs a report on test coverage. The reports are available in the PR page and also emailed to admins.
3. From there, a maintainer will accept your PR or they may request comments for you to address prior to merge. The maintainer may also ask you to [squash your commits](#) prior to merge.

TRANSLATIONS

4.1 Quick Start Guide

SecureDrop is a system that lets people share sensitive information with investigative journalists anonymously and securely. Learn *more about SecureDrop*.

The SecureDrop Client is a component of the *SecureDrop Workstation*, a new tool to enable journalists to communicate with anonymous sources and manage submitted documents via their SecureDrop, while providing mitigations against malware and other security risks. The Workstation and its components, including the Client, are currently in a limited beta phase.

Both SecureDrop and the SecureDrop Client are written in English and translated into multiple other languages. Translations are managed using **Weblate**, a web platform that enables collaborative translation projects.

4.1.1 Getting help

If you're interested in helping with translation and have questions about anything in this document, here's how to ask for help:

- Post a message in the *translations* category of the *SecureDrop* forum
- Chat in the *SecureDrop* instant messaging channel
 - *Localization Lab*, with whom we coordinate SecureDrop's translation, also maintains *their own channel*, hosted by the Internet Freedom Festival.
- Read the *Weblate* documentation

4.1.2 Get started using Weblate

You can choose to *register on Weblate* with your email address, or by linking a GitHub account.

You can contribute to any language, and Weblate has some conveniences to make it easier to work with your preferred languages. Learn *how to choose your preferred languages on Weblate*.

Our Weblate instance only contains one project, SecureDrop, which has four translation components:

1. **SecureDrop:** The main SecureDrop web application.
2. **desktop:** The translations for the desktop icons of the admin and journalist workstations used by news organizations.
3. **SecureDrop Glossary:** Weblate's internal *glossary* for SecureDrop terms.
4. **SecureDrop Client:** The SecureDrop Client interface of the *SecureDrop Workstation*.

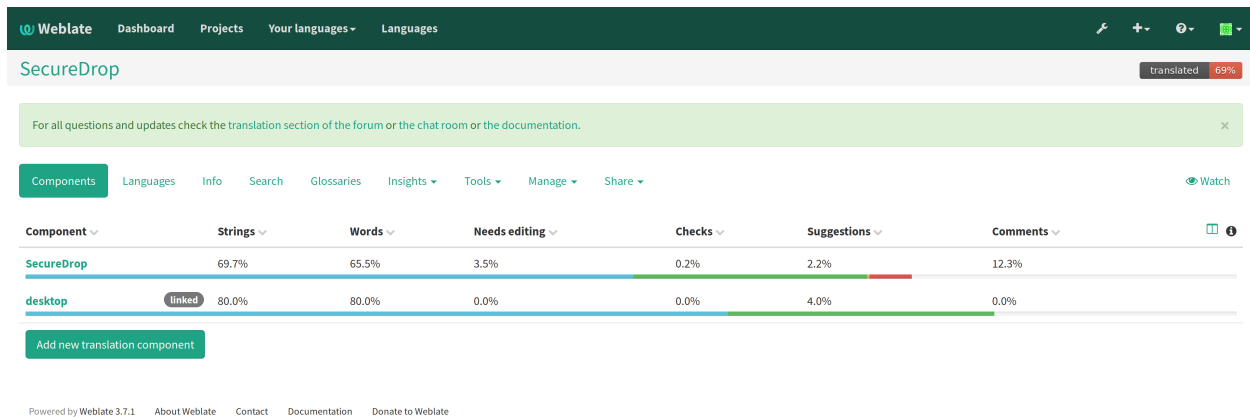


Fig. 1: Components of the SecureDrop project in Weblate. Click on the image to see it full-size.

Once a string is translated, it must be approved by a [reviewer](#) before being accepted into SecureDrop.

Approved strings can only be modified by reviewers. However, translators can still suggest modifications and make comments if they notice something wrong. SecureDrop translations are a collaborative endeavour!

Learn [how to translate SecureDrop using Weblate](#).

4.2 Background Information

4.2.1 What is SecureDrop?

SecureDrop is a system that lets people share sensitive information with investigative journalists anonymously and securely. It's designed to protect its users with strong cryptography and network communications that hide locations and activity. For more information:

- Learn about [what makes SecureDrop unique](#)
- Watch [The Globe and Mail guide to using SecureDrop](#)
- Read the [Localization Lab "Ask Me Anything" on SecureDrop](#)

4.2.2 Who uses SecureDrop?

There are two kinds of SecureDrop users: [Sources](#) and [Journalists](#). A source is an individual who wants to communicate securely and anonymously with a journalist. Sources are not expected to have any technical background. Journalists using SecureDrop have usually received proper training and understand the basic workflow of SecureDrop.

4.2.3 How is SecureDrop translated?

SecureDrop is translated using the **Weblate** platform. *Get started using Weblate*

Sources, journalists and admins use localized versions of SecureDrop. A malicious actor could attempt to modify their behavior by creating misleading translations. In order to mitigate that risk, all translations must be reviewed and accepted by designated *reviewers* before they become part of SecureDrop.

4.2.4 When does SecureDrop's translation happen?

Most of SecureDrop's components *components* are translated during SecureDrop's periodic releases, when the latest *source strings* are made available for translation in Weblate.

The SecureDrop Client can be translated continuously, as new code and source strings are merged into the project. Translations are then finalized during the release process.

Both processes are coordinated in collaboration with [Localization Lab](#). You can watch for the announcements published in *multiple locations*.

4.3 How-to Guides

4.3.1 How to register an account on Weblate using an email address

1. Visit the [Weblate registration page](#).
2. Fill the form **Register using email** and click **Register**.
3. Check your email for a message from **weblate@securedrop.org** with the subject **[Weblate] Your registration on Weblate**.
4. That message contains a confirmation link. Click that link to complete your registration.

Fig. 2: Weblate registration page. Click on the image to see it full-size.

4.3.2 How to register an account on Weblate using a GitHub account

1. Visit the [Weblate registration page](#).
2. Click on the GitHub icon, under **Third party registration**.
3. Log into GitHub if necessary.
4. Click the green **Authorize freedomofpress** button.

The authorization request looks like this:

4.3.3 How to manage your preferred languages on Weblate

1. Visit the [Weblate dashboard](#).
2. Click the **Manage your languages** button.
3. Select the languages your want to translate.
4. Click the **Save** button.

4.3.4 How to translate a language on Weblate

1. Visit the [Weblate dashboard](#).
2. Click on the **component** in order to display the list of languages in which it is translated.
3. Click the **Translate** button.
4. *Start translating*.

4.3.5 How to suggest changes to a source string

If you notice errors in our [source strings](#), or catch us using English idioms that are hard to translate, please add comments letting us know. We appreciate your feedback very much and our release schedule includes a few days at the beginning of every translation cycle for incorporating it.

4.3.6 How to translate a phrase on Weblate

1. *Select a language*.
2. Read the translatable string in the text area labelled **Source**.
3. Review the suggested translations if there are any in the **Glossary** sidebar.
4. Review the contextual information about the [source string](#) in the **Source information** sidebar, like its location in our source code.
5. If a screenshot of the SecureDrop user interface is available, read the [source string](#) in context.
 - For SecureDrop, you can also use [SecureDrop's demo server](#) to preview the source string in context.
 - For the (beta) SecureDrop Client, consult the screenshots published with the [SecureDrop Workstation documentation](#). Feel free to [contact the SecureDrop team](#) with any questions or feedback.
6. Input your translation in the **Translation** test area near the [source string](#).
7. Click **Save**. The next untranslated string will appear automatically.

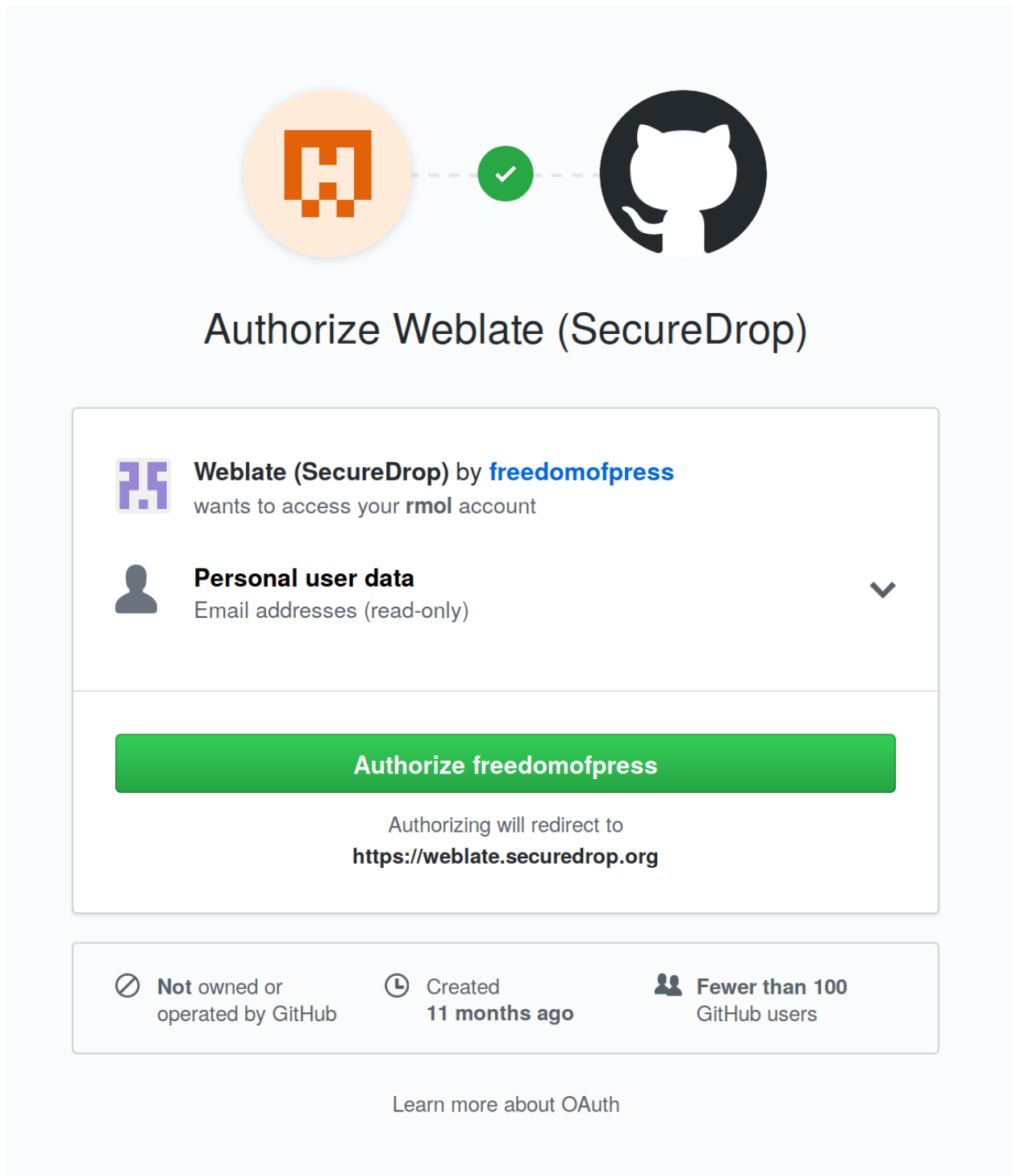


Fig. 3: GitHub authorization request. Click on the image to see it full-size.

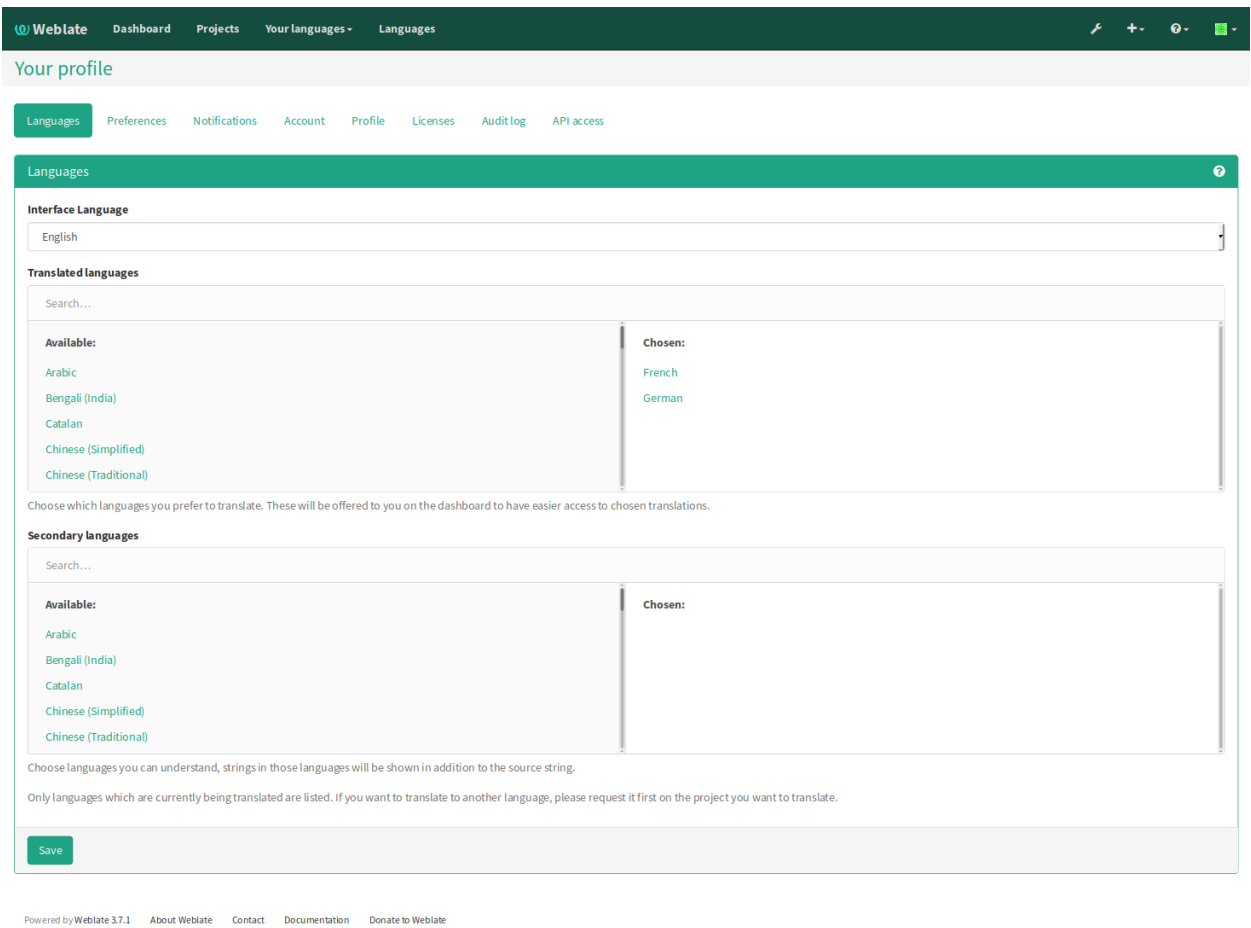


Fig. 4: Language preferences in Weblate. Click on the image to see it full-size.

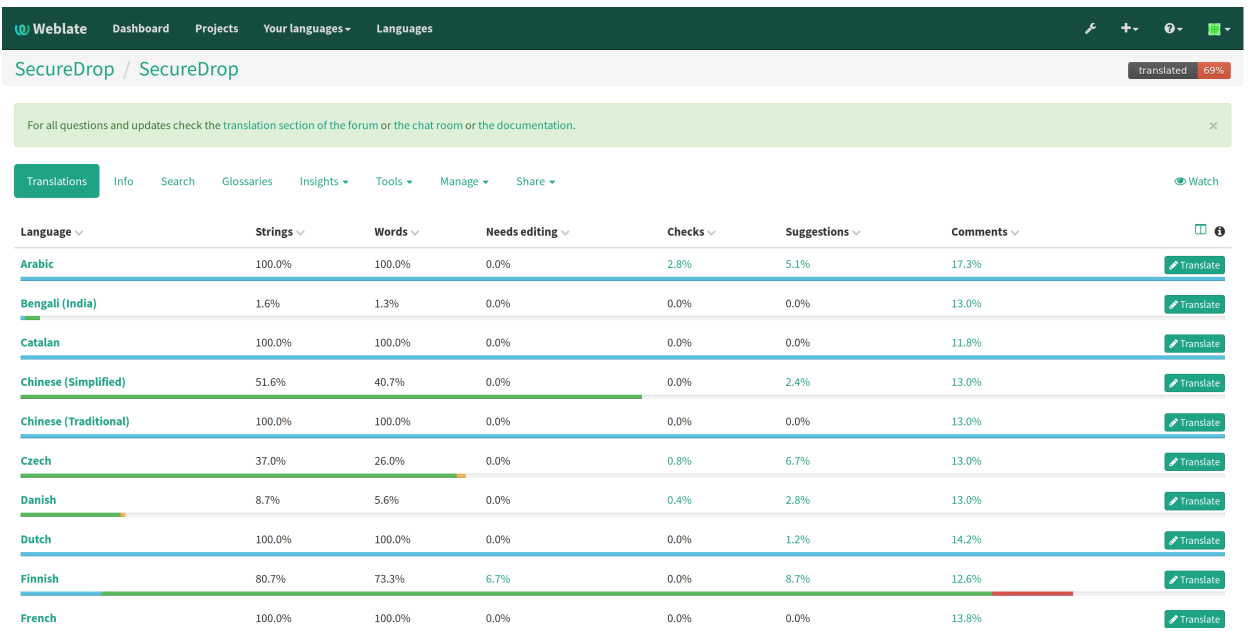


Fig. 5: List of available languages in Weblate. Click on the image to see it full-size.

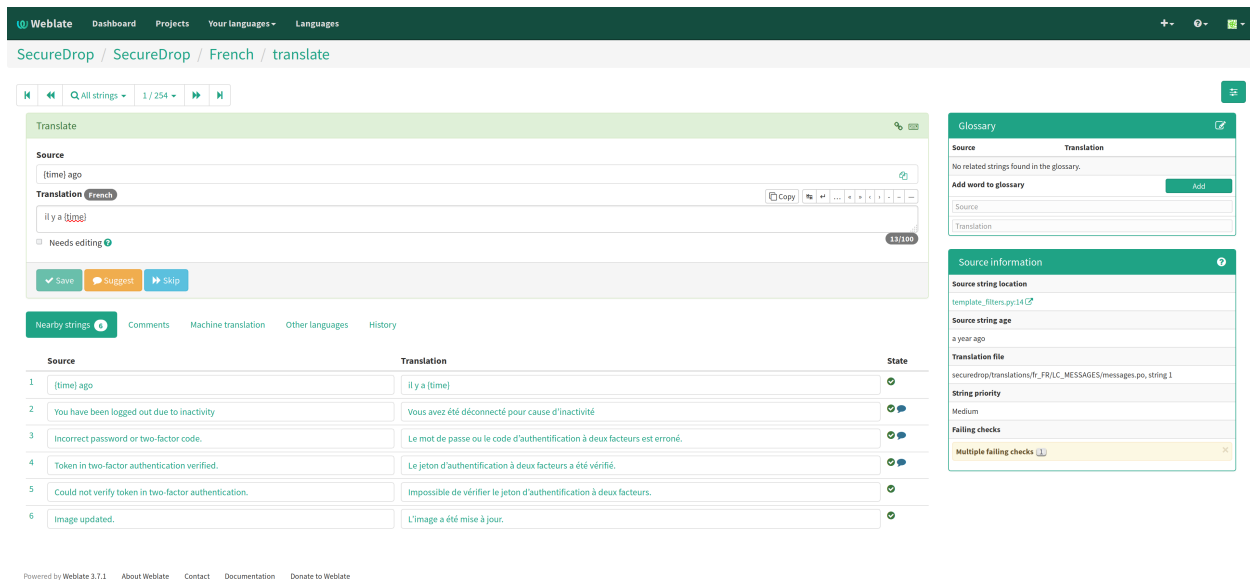


Fig. 6: Translating a phrase in Weblate. Click on the image to see it full-size.

- [Learn more about translating phrases with placeholders](#)
- [Learn more about translating phrases with HTML code](#)
- [Learn more about using language glossaries on Weblate](#)
- [Learn more about using SecureDrop's demo server](#)

4.3.7 How to change an existing translation

If you think a translation can be improved, please don't run roughshod over another translator's work. Make a suggestion or comment first, to allow for discussion before saving your changes.

Exceptions to this policy would be:

- Obvious errors in spelling, grammar, or punctuation
- A string in our interface that is supposed to match another project. For example, we include instructions for adjusting Tor Browser settings, so if our wording is out of date, it has to be corrected to reduce confusion for people using SecureDrop.

In those cases, please feel free to correct the existing translation.

4.3.8 How to translate a phrase with placeholders

Source strings may contain placeholder text in curly braces, for example `{count}`. These represent variable content (like a username, as in the example below), and must be left unmodified, but they can be moved around in a string. For instance:

```
Edit user {user}
```

might be displayed to the user as:

```
Edit user Jean-Claude
```

The French translated string should look like:

```
Modifier l'utilisateur {user}
```

And it would be **incorrect** to translate the placeholder like so:

```
Modifier l'utilisateur {utilisateur}
```

4.3.9 How to translate a phrase with HTML code

Some *source strings* represent HTML that will be presented in the SecureDrop web interface.

HTML elements (embraced by in `<`, `>`, example: ``) can contain multiple so-called *attributes*.

The text of the two attributes called `alt` and `title` should be translated. The text of the other attributes should not be translated.

Attribute alt

Image elements (``) in HTML place a picture on the page. Because people with visual impairments rely on a special note on the image element – the `alt` attribute – to describe the image, it is necessary to translate those. Here's an example that contains an image with both an `alt` attribute *and* a placeholder:

```

```

As explained above, the placeholder `{icon}` in the `src` attribute of the `` element should not be translated. The `alt` attribute text ("shield icon") should be. The correctly translated HTML in Portuguese would be:

```

```

Attribute title

Links (`<a>`) and abbreviations (`<abbr>`) sometimes rely on an additional `title` attribute. The content of that attribute is usually shown when placing a cursor over the link or abbreviation.

```
<a id="recommend-tor" title="How to install Tor Browser" href="{url}">Learn how to  
↪ install it</a>
```

It is necessary to translate the contents of any `title` attribute. The correctly translated HTML in Spanish would be:

```
<a id="recommend-tor" title="Cómo instalar Tor Browser" href="{url}">Aprenda cómo
↪ instalarlo</a>
```

As explained above, the text content `recommend-tor` of the `id` attribute in the `<a>` element should not be translated. Neither should the `{url}` placeholder of `href` attribute. Only the text content of the `title` attribute ("How to install Tor Browser") should be translated.

Other attributes

No attribute other than `alt` and `title` should be translated.

In particular, please make sure the attributes `class`, `id`, `height`, `href`, `rel`, `src` and `width` are never translated.

4.3.10 How to use the language glossaries on Weblate

Weblate contains an internal glossary for each language, to which we can add suggested translations. If a *source string* contains terms from this glossary, the glossary entries will be displayed in a box on the right side of the translation page.

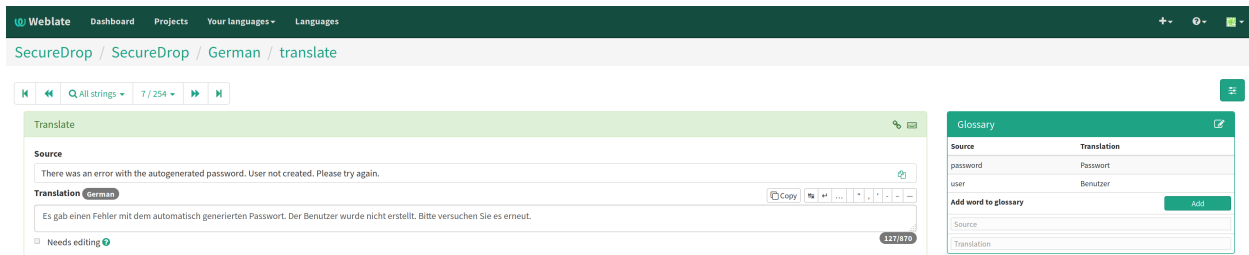


Fig. 7: Glossary sidebar in Weblate. Click on the image to see it full-size.

If you find that a *source string* contains terms from the [SecureDrop glossary](#) or the [EFF Surveillance Self-Defense glossary](#), but the glossary sidebar says `No related strings found in the glossary.`, we'd really appreciate it if you could add those terms to the glossary of the language you're working with.

4.3.11 How to use SecureDrop's demo server

The demo server always showcases the latest release candidate of SecureDrop.

Unlike a real SecureDrop instance, you can access the demo server using any web browser.

You can use it to review new *source strings* in the context in which either a *source*, or a *journalist* would read them. Those two experiences are called the *Source Interface* and the *Journalist Interface*.

In order to review the demo server as a *source*:

1. Visit [SecureDrop's demo server](#).
2. Click on the "Source Interface" link.

In order to review the demo server as a *journalist*:

1. Visit [SecureDrop's demo server](#).
2. Take note of the **username**, **passphrase** and **current TOTP token** at the bottom of the page; you will need them to log in.

- Click on the “Journalist Interface” link.
- Input the **username**, **passphrase** and **current TOTP token** (“Two-factor Code”) to log in.

4.3.12 How to become a reviewer

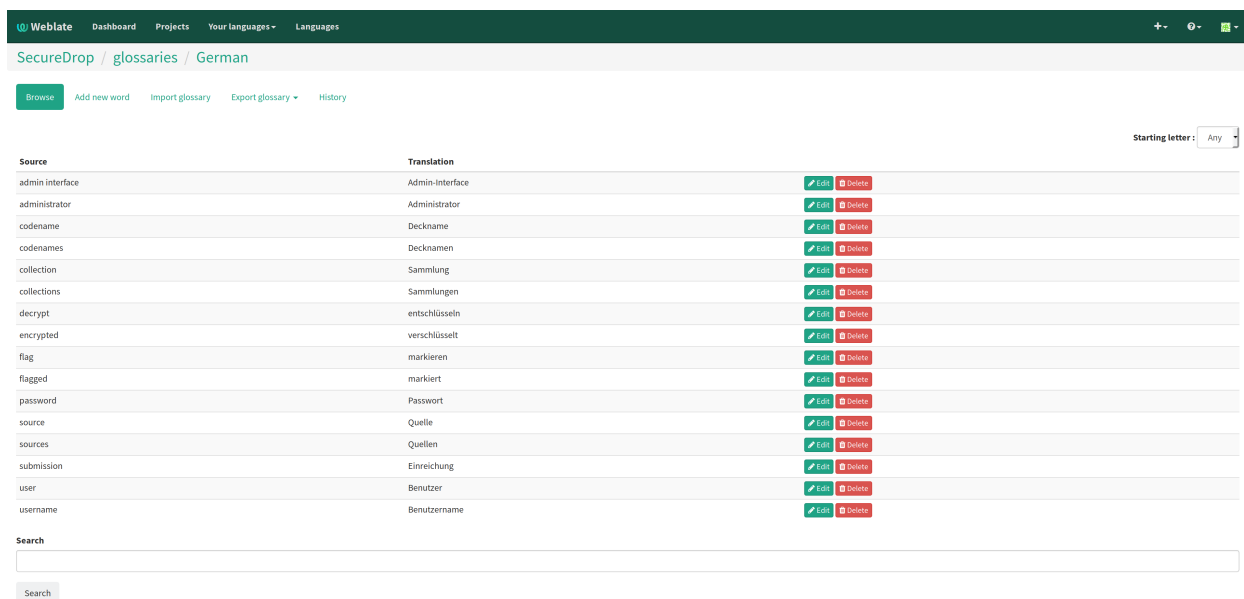
You can ask to become a reviewer for a language by posting a message in the [translations category](#) of the [SecureDrop forum](#).

4.3.13 How to add a new language to SecureDrop

We love seeing SecureDrop translated into new languages. Just ask us to add yours by posting in the [translations category](#) of the [SecureDrop forum](#).

4.4 Glossary

Weblate contains an internal glossary for each language, to which we can add suggested translations. Learn more about [using language glossaries on Weblate](#).



Source	Translation	
admin interface	Admin-Interface	Edit Delete
administrator	Administrator	Edit Delete
codename	Deckname	Edit Delete
codenames	Decknamen	Edit Delete
collection	Sammlung	Edit Delete
collections	Sammlungen	Edit Delete
decrypt	entschlüsseln	Edit Delete
encrypted	verschlüsselt	Edit Delete
flag	markieren	Edit Delete
flagged	markiert	Edit Delete
password	Passwort	Edit Delete
source	Quelle	Edit Delete
sources	Quellen	Edit Delete
submission	Einreichung	Edit Delete
user	Benutzer	Edit Delete
username	Benutzername	Edit Delete

Fig. 8: A language glossary in Weblate. Click on the image to see it full-size.

If a term is missing from the glossary for the language you’re translating into, you can refer to the following technical glossaries for additional context. Then you can contribute to improving your own language glossary on Weblate by suggesting a translation yourself!

- The [SecureDrop glossary](#) explains terms specific to SecureDrop
- The [EFF Surveillance Self-Defense glossary](#) explains many general security concepts

Additionally, here is a list of terms that are specific to the usage of Weblate for SecureDrop.

4.4.1 Reviewer

Reviewers are people who are trusted to review and accept new translations into SecureDrop. Learn [how to become a reviewer](#).

4.4.2 Source string

On Weblate, the phrases being translated are called *source strings*. (No relation with the terms *source* and *journalist* in SecureDrop.)

Source strings are English phrases and are automatically extracted from SecureDrop's code. Because of that, they can only be modified by developers outside Weblate. Learn [how to suggest changes to a source string](#).

CONTRIBUTING GUIDELINES

5.1 Signing commits

Commits should be signed, as [explained in the GitHub documentation](#). This helps verify commits proposed in a pull request are from the expected author.

5.2 Branching Strategy

Development for the upcoming release of SecureDrop takes place on `develop`, which is the default branch. If you want to contribute, you should branch from and submit pull requests to `develop`. If you want to install or audit SecureDrop, you should use the latest tag that is not a release candidate (e.g. `0.6` not `0.6-rc1`).

Tip: After you have cloned the SecureDrop repository, you can run `git tag` locally to see all the tags. Alternatively, you can view them on [GitHub](#).

5.3 Automated Testing

When a pull request is submitted, we have Circle CI automatically run the SecureDrop test suites, which consist of:

1. Unit tests of the Python SecureDrop application code.
2. Functional tests that use Selenium to drive a web browser to verify the function of the application from the user's perspective.
3. Tests of the system configuration state using `testinfra`.

Before a PR can be merged, these tests must all pass. If you modify the application code, you should verify the tests pass locally before submitting your PR. If you modify the server configuration, you should run the `testinfra` tests. Please denote in the checklist when you submit the PR that you have performed these checks locally.

5.4 Code Style

We use code linters to keep a consistent code quality and style. These linters also run in CI and will produce build failures. To avoid this, we have included a git pre-commit hook. You can install it with the following command run at the root of the repository:

```
ln -sf ../../git/pre-commit .git/hooks/pre-commit
```

Note: The code linters are installed automatically on the Development VM, but for the pre-commit hook to work, you will need to install the linting tools locally on your host machine. From the root of the repo you can run the following:

```
pip install --no-deps --require-hashes -r securedrop/requirements/python3/develop-  
↪requirements.txt
```

5.4.1 Python

All Python code should be [flake8](#) compliant. You can run flake8 locally via:

```
make flake8
```

5.4.2 Shell

All Shell code (e.g. bash, sh) should be [shellcheck](#) compliant. You can run shellcheck locally via:

```
make shellcheck
```

For reference, consult the [shellcheck](#) [wiki](#) for detailed explanations of any reported violations.

5.4.3 HTML

HTML should be in compliance with [Google’s HTML style guide](#). We use [html-linter](#) to lint our HTML templates in `securedrop/source_templates` and `securedrop/journalist_templates`. Run the HTML linting options we use via:

```
make html-lint
```

Accessibility

SecureDrop’s accessibility guidelines and tooling are a work in progress. At a minimum, if you make changes involving images, make sure they have `alt` attributes in accordance with the W3C’s “[alt decision tree](#)”, so that the interfaces will be navigable by people using screen-readers. For more-involved changes to the UIs, consult resources such as [the A11y Project checklist](#).

If you have accessibility expertise to offer, the “[a11y](#)” [label](#) in GitHub is a great place to contribute.

5.4.4 YAML

The Ansible configuration is specified in YAML files, including variables, tasks, and playbooks. All YAML files in the project should pass the `yamllint` standards declared in the `.yamllint` file at the root of the repository. Run the checks locally via:

```
make yamllint
```

5.5 Type Hints in Python code

By adding type hints/annotations in the Python code, we are making the codebase easier to maintain in the long run by explicitly specifying the expected input/output types of various functions.

Any pull request with Python code in SecureDrop should have corresponding type hints for all the functions. Type hints and function annotations are defined in [PEP 484](#) and in [PEP 3107](#). We also use the `mypy` tool in our CI to find bugs in our Python code.

If you are new to Python type hinting, please read the above mentioned PEP documents, and then go through the examples in the [mypy documentation](#). Some type annotations are included as code comments due to SecureDrop being Python 2 only when they were added, but any annotation syntax supported in Python 3.5 is allowed (i.e. function but not variable annotations which were added in Python 3.6).

5.5.1 Example of Type Hint

```
import typing
# https://www.python.org/dev/peps/pep-0484/#runtime-or-type-checking
if typing.TYPE_CHECKING:
    # flake8 can not understand type annotation yet.
    # That is why all type annotation relative import
    # statements has to be marked as noqa.
    # https://flake8.pycqa.org/en/latest/user/error-codes.html?highlight=f401
    from typing import Dict # noqa: F401

class Config(object):

    def __init__(self):
        # type: () -> None
        self.NAMES = {} # type: Dict[str, str]

    def add(self, a, b):
        # type: (int, int) -> float
        c = 10.5 # type: float
        return a + b + c

    def update(self, uid, Name):
        # type: (int, str) -> None
        """
        This method updates the name example.
        """
        self.NAMES[uid] = Name
```

(continues on next page)

(continued from previous page)

```
def main():
    # type: () -> None
    config = Config() # type: Config
    config.add(2, 3)
    config.update(223, "SD")

if __name__ == '__main__':
    main()
```

The above example shows how to do a conditional import of Dict class from `typing` module. `typing.TYPE_CHECKING` will only be true when we use mypy to check type annotations.

5.5.2 How to Use mypy?

`make lint` already checks for any error using the mypy tool. In case you want to have a local installation, you can do that using your Python 3 virtualenv.

```
$ python3 -m venv ../.py3
$ source ../.py3/bin/activate
$ pip install mypy
$ mypy securedrop
```

5.6 Git History

We currently use an explicit merge strategy to merge feature branches into `develop`.

Note: It is generally good practice to maintain a clean git history by reducing the number of commits to a reasonable minimum. You can do this by squashing closely related commits through an interactive rebase once your PR is close to being merged. If you are unfamiliar with how to squash commits with rebase, check out this [blog post](#).

If you would like a project maintainer to help you with squashing commits in a PR, please don't hesitate to leave a comment requesting assistance.

5.7 Privileges

Note: The privilege escalation workflow is different for *code maintainers* and *translation maintainers*.

Dedicated contributors to SecureDrop will be granted extra privileges such as the right to push new branches or to merge pull requests. Any contributor with the right technical and social skills is entitled to ask. The people who have the power to grant such privileges are committed to do so in a transparent way as follows:

1. The contributor posts a message [in the forum](#) asking for privileges (review or merge, etc.).
2. After at least a week someone with permissions to grant such privilege reviews the thread and either:
 - grants the privilege if there are no objections from current maintainers and adds a message to the thread; or
 - explains what is expected from the contributor before they can be granted the privilege.

3. The thread is closed.

The privileges of a developer who has not been active for six months or more are revoked. They can apply again at any time.

5.8 Other Tips

- To aid in review, please write [clear commit messages](#) and include a descriptive PR summary. We have a PR template that specifies the type of information you should include.
- To maximize the chance that your PR is merged, please include the minimal changes to implement the feature or fix the bug.
- If there is not an existing issue for the PR you are interested in submitting, you should submit an issue first or comment on an existing issue outlining how you intend to approach the problem.

TIPS & TRICKS

6.1 Using Tor Browser with the Development Environment

We strongly encourage sources to use Tor Browser when they access the Source Interface. Tor Browser is the easiest way for the average person to use Tor without making potentially catastrophic mistakes, makes disabling JavaScript easy via the handy NoScript icon in the toolbar, and prevents state about the source's browsing habits (including their use of SecureDrop) from being persisted to disk.

Since Tor Browser is based on an older version of Firefox (usually the current ESR release), it does not always render HTML/CSS the same as other browsers (especially more recent versions of browsers). Therefore, we recommend testing all changes to the web application in the Tor Browser instead of whatever browser you normally use for web development. Unfortunately, it is not possible to access the local development servers by default, due to Tor Browser's proxy configuration.

To test the development environment in Tor Browser, you need to modify Tor Browser's default settings to prevent localhost from being resolved by the proxy:

1. In a new tab, navigate to `about:config`.
2. Click "I accept the Risk!"
3. In the search bar, enter `network.proxy.allow_hijacking_localhost`.
4. The default value is `true`. Double-click to set it to `false`.

Now you should be able to navigate to `127.0.0.1:8080` and `127.0.0.1:8081` in Tor Browser. For some reason, you have to use `127.0.0.1` – `localhost` doesn't work.

The modified value persists across restarts of Tor Browser.

6.2 Upgrading or Adding Python Dependencies

We use a `pip-compile` based workflow for adding Python dependencies. If you would like to add a Python dependency, instead of editing the `securedrop/requirements/python3/*.txt` files directly, please:

1. Edit the relevant `*.in` file in `securedrop/requirements/python3`
2. Use the following shell script to generate `securedrop/requirements/python3/*.txt` files:

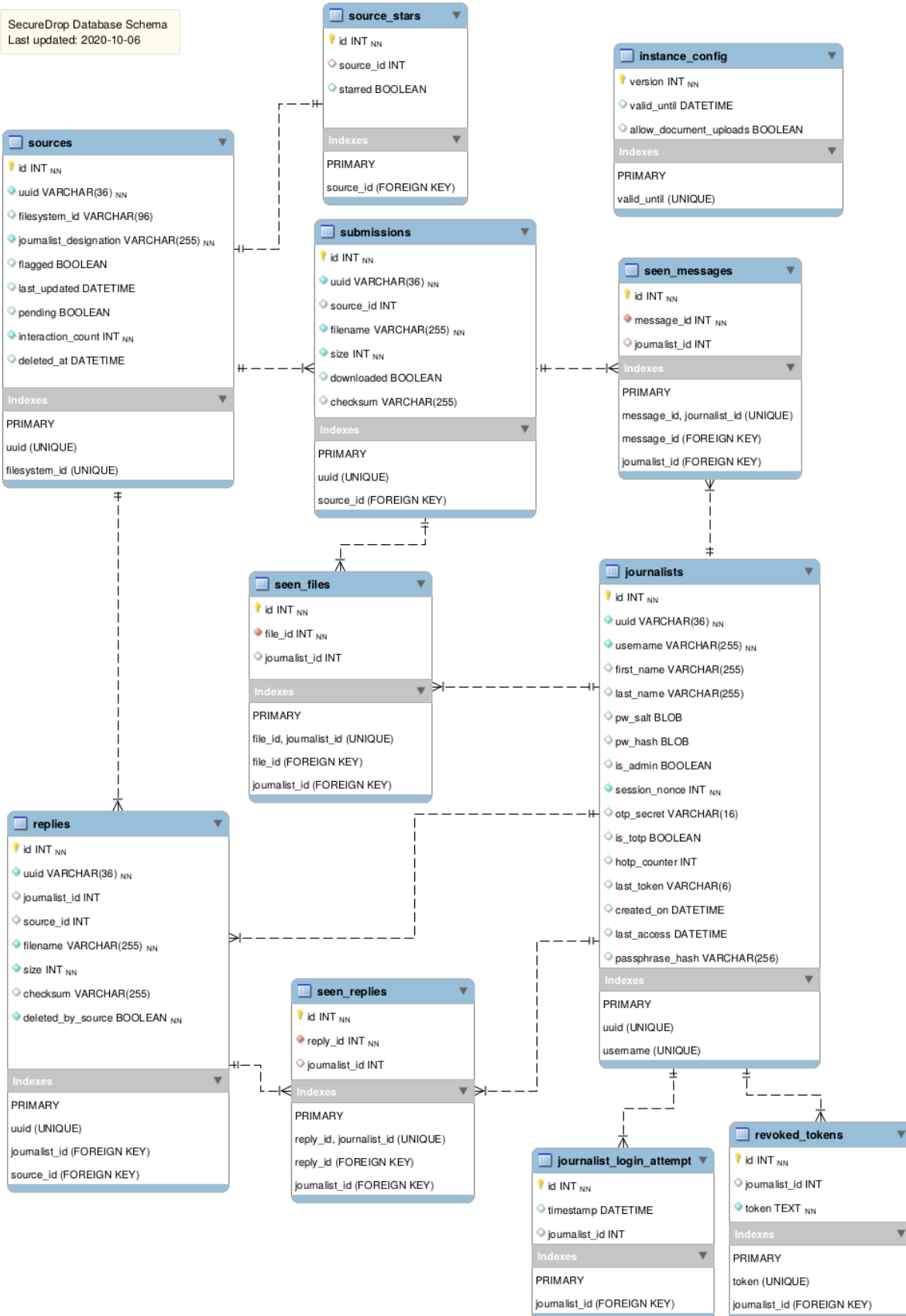
```
make update-pip-requirements
```

3. Commit both the `securedrop/requirements/python3/*.in` and `securedrop/requirements/python3/*.txt` files

Note that application dependency changes are subject to closer review, using *diffoscope* or a similar tool to compare the old and updated dependencies. You can request a review when submitting a PR.

6.3 Architecture Diagrams

Some helpful diagrams for getting a sense of the SecureDrop application architecture are stored [here](#), including a high-level view of the SecureDrop database structure:



JOURNALIST INTERFACE API

This document describes the endpoints for SecureDrop's Journalist Interface API.

7.1 Versioning

The API is versioned and we are currently using version 1. This is set via the base URL, which is:

```
/api/v1/
```

7.2 Content Type

Clients shall send the following headers:

```
'Accept': 'application/json',  
'Content-Type': 'application/json'
```

7.3 Authentication

POST /api/v1/token to get a token with the username, password, and two-factor code in the request body:

```
{  
  "username": "journalist",  
  "passphrase": "monkey potato pizza quality silica growing deduce",  
  "one_time_code": "123456"  
}
```

This will produce a response with your Authorization token:

```
{  
  "expiration": "2018-07-10T04:29:41.696321Z",  
  "token": "eyJhbGciOiJIUzI1NiIsImV4cCI6MTUzMTE5Njk4MSwiaWF0IjoxNTMxMTY4MTgxfQ.  
↪eyJpZCI6MX0.TBSvfrICMxtvWgpVZzqTl6wHYNQuGPOaZpuAKwwIXXo",  
  "journalist_uuid": "54d81dae-9d94-4145-8a57-4c804a04cfe0",  
  "journalist_first_name": "daniel",  
  "journalist_last_name": "ellsberg"  
}
```

Thereafter in order to authenticate to protected endpoints, send the token in HTTP Authorization header:

```
Authorization: Token eyJhbGciOiJIUzI1NiIsImV4cCI6MTUzMDU4NjU4MiwiYWV0IjoxNTMwNTc5MzgyfQ.  
↪eyJpZCI6MX0.P_PfcLMk1Dq5VCIANo-1Jbu0ZyCL2VcT8qf9fIZsTCM
```

This header will be checked with each API request to see if it is valid and not yet expired. Tokens currently expire after 8 hours.

7.3.1 Logout

Clients should use the logout endpoint to invalidate their token:

POST /api/v1/logout with the token in the HTTP Authorization header and you will get the following response upon successful invalidation of the API token:

```
{  
  "message": "Your token has been revoked."  
}
```

7.4 Errors

The API will respond to all errors (400-599) with a JSON object with the following fields:

```
{  
  "message": "This is a detailed error message."  
}
```

7.5 Endpoints

7.5.1 Root Endpoint

Does not require authentication.

The root endpoint describes the available resources:

```
GET /api/v1/
```

Response 200 (application/json):

```
{  
  "all_users_url": "/api/v1/users",  
  "auth_token_url": "/api/v1/token",  
  "current_user_url": "/api/v1/user",  
  "replies_url": "/api/v1/replies",  
  "seen_url": "/api/v1/seen",  
  "sources_url": "/api/v1/sources",  
  "submissions_url": "/api/v1/submissions"  
}
```

7.5.2 Sources

Get all sources

Requires authentication. Provides a list of all sources and data about them (such as number of documents, submissions, and their public key that replies should be encrypted to).

```
GET /api/v1/sources
```

Response 200 (application/json):

```
{
  "sources": [
    {
      "add_star_url": "/api/v1/sources/9b6df7c9-a6b1-461d-91f0-5b715fc7a47a/add_
      ↪star",
      "interaction_count": 2,
      "is_flagged": false,
      "is_starred": false,
      "journalist_designation": "validated benefactress",
      "key": {
        "fingerprint": "8C71EA66B0278309A31DBD691733DA655854DB12",
        "public": "-----BEGIN PGP PUBLIC KEY BLOCK-----\n\
        ↪nmQINBFGRfoABEACf5Y+6prky4JcWmKSsuh/52ZLw1FTCqrgAIK0QVFZ+cy2riFHv\
        ↪njQXYB4bPOct7PmYbmMxxIWkXqJCaPvklbpi7p5X2Wkgh+qGgjIjotq2Y9iPP6KQ3\nGvJdpG3rWwbOsrt4rDh/
        ↪L/lStn+ty4io3cDr7l7IS0t0cmOPKeKv6eGxSmCAYsnJ\nKKsIWcSjfb82KhCzL/
        ↪BBApqXt9uc6Jqjh1RPL3bGIG0tq37yX/zbFefDBDF8m8d6\nc7pvvYMa090PGViBVg6hh8+rPq/
        ↪rK7YyHOWZl7t6MXw7cm/GaH+DkGxGKe8Yuj92R\nOPNQFfpAI/tXldEcEvdG/
        ↪4mba7uxrEMe33tsnbQamFZtXFAIrSjXa904CEEWnRCz\nNE90u9FeM4bk/lModsr7gOrWbO6QwctVt/
        ↪YnvI7blUXzpMzDsbgvR89auKS9VHGZ\nY5L3yz0yVwRAIw3/
        ↪CwsJEYajKiPadcExhZhc8OCTTe8zPXxQ80WrvmfBA6x6cfvq\nSsqoH3NXrDVY/
        ↪6w9dCqVXi7cYynATqm0Qkk81jXE3BEfx7AQPXHXGasvFM1mqeQU\n+WQPqUKheomy7/
        ↪7z3heasKub3MYLkuW6y7c31z6cmvt6h5fYcNPvQXCox4BJkVcK\
        ↪nPhbst612sbqhTQEeSsDnVU1sPLxpfbx7fKuWQlEV8kfm4JsMbryqG9Z0RQARAQAB\
        ↪ntHxBdXRvZ2VuzXJhdGVkIEtleSA8UF1NR0IzRE9BNVFLVFozNjVPuTNQWUpDMk9a\
        ↪nQ0RXQjIyM1dFS1Q3V0o1NDI0QUZUT1ZFSjI0SEpaSFRYQTZTQjVGVkFBVjdHRVFQ\
        ↪nS01HQjQzUUxMVzNTRUxWFENYUWk1VRk5QWU2WT0+iQI3BBMBCgAhBQJRkX6AAhsv\
        ↪nBQsJCAcCBhUICQoLagQWAgMBAh4BAheAAAJEBcz2mVYVNsSQ88P/3e54noTBb/O\
        ↪nFVVNYw5oY9zIQPsoYUkCCvKCv26bi3qpfSDWjohyupKLth9AfFBTk3oiNhzeFhiv\
        ↪nZ5RbLgJYAWuzWNdMCSd3RAqZbbzFx3255oR9t+/RNwjeOqKpo0313myAKsRR1z+N\nbRF0A1C8GiMOCrvV/
        ↪9p+rsTDrV+8fXkrQz55nGkt6JlI43EqLH0Eg7wxI+HMgTdz\
        ↪nsPWB63INNhrR5Ln7YSh0BmnUWjpEjFYvZlAbzkMbbfznDZ2g7auRpT0S8vNgcG\
        ↪n9k9dG3gpMFhHiaE4Smd0Ib82qv9X6Q70wwxmz85JAe/P/CYsndUbRHSfXmp16igm\nj0RfcC7J0E/
        ↪SkwBY9jc+YtGCWfqqXa1a4uY03vN1YqqFWqb+exa/Qv14wwgcS17p\n80/
        ↪X1y9gPV0qleikFgNt8sPd+a21VdRSjh4Xh7l6eTHMqoDUJXtFu0evSg3oBFZj\n80IXe8KZltJCylxN+1/
        ↪xlvZjAVfmYT6kxOXYsPB3o3Z9Hemgsw2PnjI04ZMwTSyb\n101xfgB1XBd1Hrv9WQ5PNoPwXRhx7/bfzQWTx/
        ↪uP8luT6yqEerLiF0m/ShvYvKQa\
        ↪ncLuwT3Rlj1BD5CpdG+491jJ6cRXq8xfYmCd2MmBTtMAoq4DobYw75NKIssZ5gs6\
        ↪nu6NXuCW0sf8lQNBKxkNpuohLlTef8n1y\n=Zp4Z\n-----END PGP PUBLIC KEY BLOCK-----\n",
        "type": "PGP"
      },
      "last_updated": "2018-07-10T00:52:21.157409Z",
      "number_of_documents": 0,
    }
  ]
}
```

(continues on next page)

(continued from previous page)

```

    "number_of_messages": 2,
    "remove_star_url": "/api/v1/sources/9b6df7c9-a6b1-461d-91f0-5b715fc7a47a/
↪remove_star",
    "replies_url": "/api/v1/sources/9b6df7c9-a6b1-461d-91f0-5b715fc7a47a/replies
↪",
    "submissions_url": "/api/v1/sources/9b6df7c9-a6b1-461d-91f0-5b715fc7a47a/
↪submissions",
    "url": "/api/v1/sources/9b6df7c9-a6b1-461d-91f0-5b715fc7a47a",
    "uuid": "9b6df7c9-a6b1-461d-91f0-5b715fc7a47a"
  },
  {
    "add_star_url": "/api/v1/sources/f086bd03-1c89-49fb-82d5-00084c17b4ce/add_
↪star",
    "interaction_count": 2,
    "is_flagged": false,
    "is_starred": false,
    "journalist_designation": "navigational firearm",
    "key": {
      "fingerprint": "C20D06197FFAE44552358AA5886EEA0A360D9FF1",
      "public": "-----BEGIN PGP PUBLIC KEY BLOCK-----\n\
↪nmQINBFRfoABEACdO+SPazdXyWRnK6JQmDvwL5Vfmp4bxK3fzM6JF00X6B6T8Unj\
↪n5bLyUM3+K7Cwp4x1uANo60X5k6zMJFqxFVbIdXearfU0DyGWG3DINGsIwf1NNkuA\
↪noj3QVcv+jhigpn1wZvDT8AyJqaEisUddREUw1Cpv0dCFw1uIFfodz5GJmVXZnApN\n27BJKNnsJtL8lWrUvTY/
↪n4afXgMZ78ZH8aOkdmJ7wmVbIhrZlHu4UHJP6DbCm/+D\
↪n7o74ozWCv6si9bfbpG6UbCxVqaeRYjb1kGT0y36TLy8W6+JXw+yISgKT0RETTjQX\
↪nzzHP5gfLu8ZTJhSvMV+xpKxc0HaX6P80rQR40QfVYRg01uZ1Bfab+rPdUrQSPdnb\
↪ntN6Rh6rN0QfucupPYpiS8Ajl1Si9ztyIdkYLJTL/Cse06SWDc/krIj8mX4VbN0h0\
↪nYwECCbtv5uX8q3Jhkc8oTjpW+DRxfb1UW7us1nOoXVj9a0QaUM6QZtbVz0qQDJ9e\
↪nSOqIx2tv5qToTxKim8E9HjX+NCvZKDIqvaoDpreMHkFP/Fo0t0tnbHTZAWcUMaih\
↪n5WNqrFqpGym1fdFYDIL9m3DPVaFhk3e07apxQXwDrckeRY7Bma+YLOXG4yVf/If6\
↪nKedgBz0NxlGZcU6c10Fy3Dn90jcjYtT0trEsVORdFE/1SVBKmA0jpYirnQARAQAB\
↪ntHxBdXRvZ2VuzXJhdGvKIEtleSA8MldXQlhaRlo1Q1RYSkVCQzZYQUNZUVhMWlNN\
↪nNEdCUk0zUVlZWJjJMR0VPQUxQTEZKSjVCR1lPSzRUZU0SkYSlQzTlhVTkpLQ0VH\
↪nTFU2RFVQUldGWEM1WlEzRk1UVFhDM0VSRLQzWT0+iQI3BBMBCgAhBQJRkX6AAhsv\
↪nBQsJCAcCBhUICQoLAgQWAgMBAh4BAheAAA0JEIhu6go2DZ/xLcgP/1lEL1F7hoQr\nLQm8T/
↪DqjoExh0F8am9SKb2LH9HSBUJPY9b/oPjptxyg/3NlGXP/GJGcI6SVXtnq\nGU2D2+vMUUrNV/
↪AemAtBUIquIXMEUjbGdK0uWTBcntgj6PJL6/VNi2o+v9FxAtn1\n6hefcd0Ik7DMaK8y56BJA+aI/
↪7TnCr1ndHLUMXh0rKd8GS13vXtv2kuY8iSqiOmj\
↪nu0tW1w2lByFBglNLgnozdbudwVqNvKX8j3oWJKsJ525Y3HsWka/l4GbkowveUYR\
↪nU66usAX6KS1zt01pLDmYFCL7LX8SPkZq97qHoFa1C9NIHW2gP+y8Q922E9QWBqy7\n/
↪g30ZF73MgZCOnFOChswH607LBvMGUyz+A2Qjpd7Zvf67G33inY7Q1GkMI59Zz4T\nnXXv/
↪1U3G16LLkwGWrTDhqHgK2KA9+B6gPYDV9xh/1HTvLBE4Wf8EHhtUyW1ZxzY5\nnuXvZt50H/UKpuhcsuN6c/
↪5+Qqk0i85jTBPXm7/0XcbbRuBTnl6CiVM8vGuaLjOdW\
↪ntAlRmX9hS7jmdE9e3Yl17qUPw1EEKSFH8Z6GgEEommoHPsgmDrQxUS6v68zfcmf3\
↪nAE+dfKUDfC7muZfZQ0YaqeHMrDyLozRIjVtx6P3fxZPZfUvfrV4guJOVOMwi+Z1F\
↪n5UrZB6IrSA4njr9Vr+Fb0p+v73pfV6NT\nn=e+yq\n-----END PGP PUBLIC KEY BLOCK-----\n",
      "type": "PGP"
    },
    "last_updated": "2018-07-10T00:52:25.696391Z",
    "number_of_documents": 0,
    "number_of_messages": 2,
    "remove_star_url": "/api/v1/sources/f086bd03-1c89-49fb-82d5-00084c17b4ce/

```

(continues on next page)

(continued from previous page)

```

    ↪remove_star",
        "replies_url": "/api/v1/sources/f086bd03-1c89-49fb-82d5-00084c17b4ce/replies
    ↪",
        "submissions_url": "/api/v1/sources/f086bd03-1c89-49fb-82d5-00084c17b4ce/
    ↪submissions",
        "url": "/api/v1/sources/f086bd03-1c89-49fb-82d5-00084c17b4ce",
        "uuid": "f086bd03-1c89-49fb-82d5-00084c17b4ce"
    }
]
}

```

Get a single source

Requires authentication.

```
GET /sources/<source_uuid>
```

Response 200 (application/json):

```

{
  "add_star_url": "/api/v1/sources/9b6df7c9-a6b1-461d-91f0-5b715fc7a47a/add_star",
  "interaction_count": 2,
  "is_flagged": false,
  "is_starred": false,
  "journalist_designation": "validated benefactress",
  "key": {
    "fingerprint": "8C71EA66B0278309A31DBD691733DA655854DB12",
    "public": "-----BEGIN PGP PUBLIC KEY BLOCK-----\n\
    ↪nmQINBFGRfoABEACf5Y+6prky4JcWmKSuh/52ZLw1FTCqrgAIK0QVFZ+cy2riFHv\
    ↪njQXYB4bPOCt7PmYbmMxxIWkXqJCaPVkLbpi7p5X2Wkgh+qGgjIjotq2Y9iPP6KQ3\nGvJdpG3rWwbOsrt4rDh/
    ↪L/lStn+ty4io3cDr7l7IS0t0cmOPKeKv6eGxSmCAYsnJ\nKKsIWcSjfb82KhCzL/
    ↪BBAppqXt9uc6Jqjh1RPL3bGIG0tq37yX/zbFefDBDF8m8d6\nc7pvvYMa090PGViBVg6hh8+rPq/
    ↪rK7YyHOWZlt6MXw7cm/GaH+DkGxGKe8Yuj92R\nOPNQFfpAI/tXldEcEvdG/
    ↪4mba7uxrEMe33tsnbQamFZtXFAIrSjXa904CEEwNRcz\nNE90u9FeM4bk/lModsr7gOrWbO6QwctVt/
    ↪YnvI7blUXzpMzDsbgvR89auKS9VHGZ\nY5L3yz0yVwRAIw3/
    ↪CwsJEYajKiPadcExhZhc8OCTTe8zPXxQ8OWrmFBA6x6cfvq\nSsqH3NXrDVY/
    ↪6w9dCqVXitcYynATqm0Qkkr81jXE3BEfx7AQPXHXGasvFM1mqeQU\n+WQPqUKheomy7/
    ↪7z3heasKub3MYLkuW6y7c31z6cmvt6h5fYcNPvQXCox4BJkVcK\
    ↪nPbzst612sbqhtQEeSsDnVU1sPLxpfbxKfKuWQLEv8kfm4JsMbryqG9Z0RQARAQAB\
    ↪ntHxBdXRvZ2VuZXJhdGVkIETleSA8UF1NR0IzRE9BNVFLVfFozNjVPuTNQWUpDMk9a\
    ↪nQ0RXQjIyM1dFS1Q3V0o1NDI0QUZUT1ZFSjI0SEpaSFRYQTZTQjVGVUkFBVjdHRVFQ\
    ↪nS01HQjQzUUxMVzNTRUxFWENYWklVRk5QWTU2WT0+iQI3BBMBCgAhBQJRkX6AAhsv\
    ↪nBQsJCACBhUICQoLAqQWAgMBAh4BAheAAAoJEBcz2mVYVNsSQ88P/3e54noTBb/O\
    ↪nFVVNYw5oY9zIQPsoYUkCCvKCv26bi3qpfSDWjohyupKLth9AfFBTk3oiNhzeFhiv\
    ↪nZ5RbLgJYAWuzWNdMCSd3RAqZbbzFx3255oR9t+/RNwje0qKpo0313myAKsRR1z+N\nbRF0A1C8GiMOCrvV/
    ↪9p+rsTDrv+8fXkrQz55Ngt6JlI43Eq1H0Eg7wxI+HMgTdz\
    ↪nsPWRB63INNhrR5Ln7YShOBmnUWjpeJfYvZ1AbzkMbbfznDZ2g7auRpT0S8vNgcG\
    ↪n9k9dG3gpMFnHiaE4Smd0Ib82qv9X6Q7Owwxmz85JAe/P/CYsndUbRHSfXmp16igm\nj0RfcC7J0E/
    ↪SkwBY9jc+YtGCWfqQXa1a4uY03vN1YqqFWqb+exa/Qv14wwgcS17p\n80/
    ↪X1y9gPV0qleikFgNt8sPd+a2lVdRSjh4Xh7l6eTHMQDUJXtFu0evSg3oBFZj\n80IXe8KZltJCylxN+1/
    ↪xlvZjAVfmYT6kxOXYsPB3o3Z9Hemgsw2PnjI04ZMwTSyb\n101xfgB1XBd1Hrv9WQ5PNoPwXRhx7/bfzQWTx/
  }
}

```

(continues on next page)

(continued from previous page)

```

↪uP8luT6yqEerLiF0m/ShvYvKQa\
↪ncLuwtW3Rlj1BD5CpdG+491jJ6cRXq8xfYmCd2MmBTtMAoq4DobYw75NKIssZ5gs6\
↪nu6NXuCWOf8lQNBKxkNpuohLlTef8n1y\n=Zp4Z\n-----END PGP PUBLIC KEY BLOCK-----\n",
  "type": "PGP"
},
  "last_updated": "2018-07-10T00:52:21.157409Z",
  "number_of_documents": 0,
  "number_of_messages": 2,
  "remove_star_url": "/api/v1/sources/9b6df7c9-a6b1-461d-91f0-5b715fc7a47a/remove_star
↪",
  "replies_url": "/api/v1/sources/9b6df7c9-a6b1-461d-91f0-5b715fc7a47a/replies",
  "submissions_url": "/api/v1/sources/9b6df7c9-a6b1-461d-91f0-5b715fc7a47a/submissions
↪",
  "url": "/api/v1/sources/9b6df7c9-a6b1-461d-91f0-5b715fc7a47a",
  "uuid": "9b6df7c9-a6b1-461d-91f0-5b715fc7a47a"
}

```

Get all submissions associated with a source

Requires authentication.

```
GET /api/v1/sources/<source_uuid>/submissions
```

Response 200 (application/json):

```

{
  "submissions": [
    {
      "download_url": "/api/v1/sources/e5a42bdb-1fef-4d66-9876-b2d592f90704/submissions/
↪b7a7b6ca-9a11-4a51-8b59-7e454f6bf8d0/download",
      "filename": "1-dark-haired_insolation-msg.gpg",
      "is_file": false,
      "is_message": true,
      "is_read": true,
      "seen_by": [
        "1c914871-a335-44ba-b2ae-da878cbc3630"
      ],
      "size": 593,
      "source_url": "/api/v1/sources/e5a42bdb-1fef-4d66-9876-b2d592f90704",
      "submission_url": "/api/v1/sources/e5a42bdb-1fef-4d66-9876-b2d592f90704/
↪submissions/b7a7b6ca-9a11-4a51-8b59-7e454f6bf8d0",
      "uuid": "b7a7b6ca-9a11-4a51-8b59-7e454f6bf8d0"
    },
    {
      "download_url": "/api/v1/sources/e5a42bdb-1fef-4d66-9876-b2d592f90704/submissions/
↪00d24bed-8d13-4f90-b068-52341593a727/download",
      "filename": "2-dark-haired_insolation-doc.gz.gpg",
      "is_file": true,
      "is_message": false,
      "is_read": true,
      "seen_by": [

```

(continues on next page)

(continued from previous page)

```

    "1c914871-a335-44ba-b2ae-da878cbc3630"
  ],
  "size": 179404,
  "source_url": "/api/v1/sources/e5a42bdb-1fef-4d66-9876-b2d592f90704",
  "submission_url": "/api/v1/sources/e5a42bdb-1fef-4d66-9876-b2d592f90704/
↳ submissions/00d24bed-8d13-4f90-b068-52341593a727",
  "uuid": "00d24bed-8d13-4f90-b068-52341593a727"
}
]
}

```

Get a single submission associated with a source

Requires authentication.

```
GET /api/v1/sources/<source_uuid>/submissions/<submission_uuid>
```

Response 200 (application/json):

```

{
  "download_url": "/api/v1/sources/e5a42bdb-1fef-4d66-9876-b2d592f90704/submissions/
↳ 00d24bed-8d13-4f90-b068-52341593a727/download",
  "filename": "2-dark-haired_insolation-doc.gz.gpg",
  "is_file": true,
  "is_message": false,
  "is_read": true,
  "seen_by": [
    "1c914871-a335-44ba-b2ae-da878cbc3630"
  ],
  "size": 179404,
  "source_url": "/api/v1/sources/e5a42bdb-1fef-4d66-9876-b2d592f90704",
  "submission_url": "/api/v1/sources/e5a42bdb-1fef-4d66-9876-b2d592f90704/submissions/
↳ 00d24bed-8d13-4f90-b068-52341593a727",
  "uuid": "00d24bed-8d13-4f90-b068-52341593a727"
}

```

Get all replies associated with a source

Requires authentication.

```
GET /api/v1/sources/<source_uuid>/replies
```

Response 200 (application/json):

```

{
  "replies": [
    {
      "filename": "3-electrocardiographic_lost-and-found-reply.gpg",
      "is_deleted_by_source": false,
      "journalist_first_name": "",

```

(continues on next page)

(continued from previous page)

```

    "journalist_last_name": "",
    "journalist_username": "journalist",
    "journalist_uuid": "3ae405e0-01bb-41f5-98b6-c4707c5c4b96",
    "reply_url": "/api/v1/sources/55b96e66-688a-4333-b429-f1a3233b40e9/replies/
↪5d6260ce-cf70-420a-9ca0-250b09d6cc58",
    "seen_by": [
        "3ae405e0-01bb-41f5-98b6-c4707c5c4b96"
    ],
    "size": 753,
    "source_url": "/api/v1/sources/55b96e66-688a-4333-b429-f1a3233b40e9",
    "uuid": "5d6260ce-cf70-420a-9ca0-250b09d6cc58"
  },
  {
    "filename": "4-electrocardiographic_lost-and-found-reply.gpg",
    "is_deleted_by_source": false,
    "journalist_first_name": "",
    "journalist_last_name": "",
    "journalist_username": "journalist",
    "journalist_uuid": "3ae405e0-01bb-41f5-98b6-c4707c5c4b96",
    "reply_url": "/api/v1/sources/55b96e66-688a-4333-b429-f1a3233b40e9/replies/
↪3400b55f-9bfb-4368-b975-0f6950fd5631",
    "seen_by": [
        "3ae405e0-01bb-41f5-98b6-c4707c5c4b96"
    ],
    "size": 901,
    "source_url": "/api/v1/sources/55b96e66-688a-4333-b429-f1a3233b40e9",
    "uuid": "3400b55f-9bfb-4368-b975-0f6950fd5631"
  }
]
}

```

Get a single reply associated with a source

Requires authentication.

```
GET /api/v1/sources/<source_uuid>/replies/<reply_uuid>
```

Response 200 (application/json):

```

{
  "filename": "4-electrocardiographic_lost-and-found-reply.gpg",
  "is_deleted_by_source": false,
  "journalist_first_name": "",
  "journalist_last_name": "",
  "journalist_username": "journalist",
  "journalist_uuid": "3ae405e0-01bb-41f5-98b6-c4707c5c4b96",
  "reply_url": "/api/v1/sources/55b96e66-688a-4333-b429-f1a3233b40e9/replies/3400b55f-
↪9bfb-4368-b975-0f6950fd5631",
  "seen_by": [
    "3ae405e0-01bb-41f5-98b6-c4707c5c4b96"
  ],

```

(continues on next page)

(continued from previous page)

```

"size": 901,
"source_url": "/api/v1/sources/55b96e66-688a-4333-b429-f1a3233b40e9",
"uuid": "3400b55f-9bfb-4368-b975-0f6950fd5631"
}

```

Download a reply

Requires authentication.

```
GET /api/v1/sources/<source_uuid>/replies/<reply_uuid>/download
```

Response 200 will have Content-Type: application/pgp-encrypted and is the content of the PGP encrypted reply.

An ETag header is also present containing the SHA256 hash of the response data:

```
"sha256:c757c5aa263dc4a5a2bca8e7fe973367dbd2c1a6c780d19c0ba499e6b1b81efa"
```

Note that these are not intended for cryptographic purposes and are present for clients to check that downloads are not corrupted.

Delete a reply

Requires authentication.

```
DELETE /api/v1/sources/<source_uuid>/replies/<reply_uuid>
```

Response 200:

```

{
  "message": "Reply deleted"
}

```

Add a reply to a source

Requires authentication. Clients are expected to encrypt replies prior to submission to the server. Replies should be encrypted to the public key of the source.

Including the uuid field in the request is optional. Clients may want to pre-set the uuid so they can track in-flight messages.

```
POST /api/v1/sources/<source_uuid>/replies
```

with the reply in the request body:

```

{
  "uuid": "@bc588dd-f613-4999-b21e-1ceb9adc2c",
  "reply": "-----BEGIN PGP MESSAGE-----[...]-----END PGP MESSAGE-----"
}

```

Response 201 created (application/json):

```
{
  "message": "Your reply has been stored",
  "uuid": "@bc588dd-f613-4999-b21e-1ceb9adc2c"
}
```

The returned `uuid` field is the UUID of the reply and can be used to reference this reply later. If the client set the `uuid` in the request, this will have the same value.

Replies that do not contain a GPG encrypted message will be rejected:

Response 400 (application/json):

```
{
  "message": "You must encrypt replies client side"
}
```

Delete a submission

Requires authentication.

```
DELETE /api/v1/sources/<source_uuid>/submissions/<submission_uuid>
```

Response 200:

```
{
  "message": "Submission deleted"
}
```

Download a submission

Requires authentication.

```
GET /api/v1/sources/<source_uuid>/submissions/<submission_uuid>/download
```

Response 200 will have `Content-Type: application/pgp-encrypted` and is the content of the PGP encrypted submission.

An ETag header is also present containing the SHA256 hash of the response data:

```
"sha256:c757c5aa263dc4a5a2bca8e7fe973367dbd2c1a6c780d19c0ba499e6b1b81efa"
```

Note that these are not intended for cryptographic purposes and are present for clients to check that downloads are not corrupted.

Delete a source and all their associated submissions

Requires authentication.

```
DELETE /api/v1/sources/<source_uuid>
```

Response 200:

```
{
  "message": "Source and submissions deleted"
}
```

Delete a source conversation (messages/files/replies) while preserving the source

Requires authentication.

```
DELETE /api/v1/sources/<source_uuid>/conversation
```

Response 200:

```
{
  "message": "Source data deleted"
}
```

Star a source

Requires authentication.

```
POST /api/v1/sources/<source_uuid>/star
```

Response 201 created:

```
{
  "message": "Star added"
}
```

Unstar a source

Requires authentication.

```
DELETE /api/v1/sources/<source_uuid>/star
```

Response 200:

```
{
  "message": "Star removed"
}
```

7.5.3 Submissions

Get all submissions

Requires authentication. This gets details of all submissions across sources.

```
GET /api/v1/submissions
```

Response 200:

```
{
  "submissions": [
    {
      "download_url": "/api/v1/sources/e5a42bdb-1fef-4d66-9876-b2d592f90704/submissions/
↳ b7a7b6ca-9a11-4a51-8b59-7e454f6bf8d0/download",
      "filename": "1-dark-haired_insolation-msg.gpg",
      "is_file": false,
      "is_message": true,
      "is_read": true,
      "seen_by": [
        "1c914871-a335-44ba-b2ae-da878cbc3630"
      ],
      "size": 593,
      "source_url": "/api/v1/sources/e5a42bdb-1fef-4d66-9876-b2d592f90704",
      "submission_url": "/api/v1/sources/e5a42bdb-1fef-4d66-9876-b2d592f90704/
↳ submissions/b7a7b6ca-9a11-4a51-8b59-7e454f6bf8d0",
      "uuid": "b7a7b6ca-9a11-4a51-8b59-7e454f6bf8d0"
    },
    {
      "download_url": "/api/v1/sources/e5a42bdb-1fef-4d66-9876-b2d592f90704/submissions/
↳ 00d24bed-8d13-4f90-b068-52341593a727/download",
      "filename": "2-dark-haired_insolation-doc.gz.gpg",
      "is_file": true,
      "is_message": false,
      "is_read": true,
      "seen_by": [
        "1c914871-a335-44ba-b2ae-da878cbc3630"
      ],
      "size": 179404,
      "source_url": "/api/v1/sources/e5a42bdb-1fef-4d66-9876-b2d592f90704",
      "submission_url": "/api/v1/sources/e5a42bdb-1fef-4d66-9876-b2d592f90704/
↳ submissions/00d24bed-8d13-4f90-b068-52341593a727",
      "uuid": "00d24bed-8d13-4f90-b068-52341593a727"
    }
  ]
}
```

7.5.4 Replies

Get all replies

Requires authentication. This gets details of all replies across sources.

```
GET /api/v1/replies
```

Response 200:

```
{
  "replies": [
    {
      "filename": "3-electrocardiographic_lost-and-found-reply.gpg",
      "is_deleted_by_source": false,
      "journalist_first_name": "",
      "journalist_last_name": "",
      "journalist_username": "journalist",
      "journalist_uuid": "3ae405e0-01bb-41f5-98b6-c4707c5c4b96",
      "reply_url": "/api/v1/sources/55b96e66-688a-4333-b429-f1a3233b40e9/replies/
↪5d6260ce-cf70-420a-9ca0-250b09d6cc58",
      "seen_by": [
        "3ae405e0-01bb-41f5-98b6-c4707c5c4b96"
      ],
      "size": 753,
      "source_url": "/api/v1/sources/55b96e66-688a-4333-b429-f1a3233b40e9",
      "uuid": "5d6260ce-cf70-420a-9ca0-250b09d6cc58"
    },
    {
      "filename": "3-dark-haired_insolation-reply.gpg",
      "is_deleted_by_source": false,
      "journalist_first_name": "",
      "journalist_last_name": "",
      "journalist_username": "journalist",
      "journalist_uuid": "3ae405e0-01bb-41f5-98b6-c4707c5c4b96",
      "reply_url": "/api/v1/sources/e5a42bdb-1fef-4d66-9876-b2d592f90704/replies/
↪285682f8-2bfb-47aa-9889-f9c41a44cebb",
      "seen_by": [
        "3ae405e0-01bb-41f5-98b6-c4707c5c4b96",
        "1c914871-a335-44ba-b2ae-da878cbc3630"
      ],
      "size": 744,
      "source_url": "/api/v1/sources/e5a42bdb-1fef-4d66-9876-b2d592f90704",
      "uuid": "285682f8-2bfb-47aa-9889-f9c41a44cebb"
    }
  ]
}
```

7.5.5 Users

Get a list of all users

Requires authentication.

```
GET /api/v1/users
```

Response 200:

```
{
  "users": [
    {
      "first_name": "Nellie",
      "last_name": "Bly",
      "username": "nbly",
      "uuid": "2b3f05ef-3695-4522-88bd-f124d2e89d01"
    },
    {
      "first_name": "Daniel",
      "last_name": "Ellsberg",
      "username": "dellsberg",
      "uuid": "89eec426-f8c3-4c7a-921f-59ec8fa9fd69"
    }
  ]
}
```

Get an object representing the current user

Requires authentication.

```
GET /api/v1/user
```

Response 200:

```
{
  "is_admin": true,
  "last_login": "2018-07-09T20:29:41.696782Z",
  "username": "journalist",
  "uuid": "a2405127-1c9e-4a3a-80ea-95f6a71e5738",
  "first_name": "Bob",
  "last_name": "Smith",
}
```


Mark items that have been seen by the current user

Requires authentication. Records that the current user has seen a reply from another user, or a file or message submitted by a source.

```
POST /api/v1/seen
```

The request body should contain one or more lists of UUIDs representing the conversation items to be marked seen. The valid list keys are `files`, `messages`, and `replies`. The type of a given submission (file or message) is available in the responses from endpoints under `/submissions`; each submission will have `is_file` and `is_message` fields.

```
{
  "files": [
    "00d24bed-8d13-4f90-b068-52341593a727"
  ],
  "messages": [
    "b7a7b6ca-9a11-4a51-8b59-7e454f6bf8d0"
  ],
  "replies": [
    "285682f8-2bfb-47aa-9889-f9c41a44cebb"
  ]
}
```

Any of the lists may be omitted, but at least one must be specified. An empty or invalid request will result in a **400 Bad Request** response with the following body:

```
{
  "error": "Bad Request",
  "message": "Please specify the resources to mark seen."
}
```

A successful request will result in a **200 OK** response with the following body:

```
{
  "message": "resources marked seen"
}
```

Any submission or reply marked seen will thereafter include the user's UUID in the `seen_by` field of responses including the item, like `/api/v1/submissions` or `/api/v1/replies`.

If a file, message, or reply cannot be found with one of the specified UUIDs, the response will be **404 Not Found** with details in the response body:

```
{
  "error": "Not Found",
  "message": "reply not found: 285682f8-2bfb-47aa-9889-f9c41a44cebc"
}
```

None of the requested items will be marked seen if any of them cannot be found.

7.6 Removed functionality

7.6.1 Flagging sources

Previous versions of the API supported flagging sources for reply, which would generate a reply keypair for the source upon their next login. This functionality was removed in SecureDrop 2.0.0.

The `/api/v1/sources/<source_uuid>/flag` endpoint (POST) and the `is_flagged` property for sources are retained for backwards compatibility, but no longer function. `is_flagged` is always `false`.

The endpoint and the `is_flagged` property will be fully removed from the API in a future release.

DEVELOPMENT OF SECUREDROP-ADMIN IN THE ADMIN DIRECTORY

The `admin` directory in the SecureDrop repository root contains the source of the `securedrop-admin` script which is used in Tails to perform various administrative tasks. It is a standalone python module which can be tested on Debian GNU/Linux stretch with:

```
python3 bootstrap.py
source .venv3/bin/activate
pip3 install --no-deps --require-hashes -r requirements-dev.txt
tox
```

A Docker helper `bin/dev-shell` is provided to simplify the installation and make it portable on various operating systems. From the `admin` directory, run `bin/dev-shell` without any arguments to execute `securedrop-admin` or other commands interactively in the container. If this is your first time running `bin/dev-shell`, it may take several minutes to build the image.

Note: The SecureDrop repository contains two scripts named `dev-shell`. `admin/bin/dev-shell` is used for `securedrop-admin` while `securedrop/bin/dev-shell` is used for the server environment.

Run only flake8 with:

```
bin/dev-shell tox -e flake8
```

Run only one test foobar with:

```
bin/dev-shell tox -e py3 -- -k foobar
```

Docker has the `admin` directory mounted from the host into the container, at the same location to avoid any trouble with hardcoded absolute paths. It runs with the id of the host user so files created in the container are owned by the host user instead of root. If a script needs root access, it has passwordless `sudo` permissions.

Convenience Makefile targets are also provided for the most common tasks:

```
$ make
Makefile for developing and testing securedrop-admin.
Subcommands:

help                Print this message and exit.
test                Run tox
update-pip-requirements  Updates all Python requirements files via pip-compile.
```


DEVELOPMENT OF SECUREDROPUPDATER IN THE JOURNALIST_GUI DIRECTORY

The `SecureDropUpdater` is a tool used by the journalists and admins, this tool helps them to update their SecureDrop git repository to the latest released tag. It is a GUI tool and it is written using PyQt5 bindings of the [Qt framework](#). This tool is written using Python3.

9.1 Installing the Dependencies in a Virtual Environment

You can use Python's built-in `venv` module to install the dependencies in a virtual environment. From the `journalist_gui` directory:

```
$ python3 -m venv .venv && source .venv/bin/activate
$ pip install --require-hashes -r dev-requirements.txt
```

The first command will create a virtual environment and activate it. The second command will install the dependencies, using the exact hashes specified in `dev-requirements.txt`. Make sure you are using at least Python 3.8.

Note: The Updater GUI does not use a virtual environment on the Tails Workstations. As such, you can only use dependencies present in Tails.

You can run the GUI via:

```
$ python3 SecureDropUpdater
```

Note that since the application expects to run in Tails, you should test its functionality in a Tails VM. You can follow the instructions in the [Virtualizing Tails](#) guide to set up your Tails VM.

9.2 To Update the UI Design

The design of the GUI is saved in the `journalist_gui/mainwindow.ui` file. To update the UI, one has to first install `qtcreator` tool in the system. We are currently using *5.10.1* version of Qt for this project.

```
$ sudo apt install qtcreator python3-pyqt5
```

If we make any changes to the UI, we will have to use `pyuic5` command to update the corresponding Python code.

```
$ pyuic5 journalist_gui/mainwindow.ui -o journalist_gui/updaterUI.py
```

9.3 Using Resources in the UI

All icons and images for the UI is stored in the `journalist_gui/static` directory. These are known as resources for the project. The `journalist_gui/resources.qrc` file contains the list of current resources for the project. Each resource needs to be defined inside of a `<file></file>`.

Example qrc file:

```
<RCC>
  <qresource prefix="/images">
    <file>static/securedrop.png</file>
    <file>static/securedrop_icon.png</file>
  </qresource>
</RCC>
```

We will have to update the corresponding Python file for any change in this resource file. We can do that using the following command:

```
$ pyrcc5 journalist_gui/resources.qrc -o journalist_gui/resources_rc.py
```

Note: The `updaterUI.py` and `resources_rc.py` files are generated by the tools. So, do not make any changes to these files. Any changes made to these files will be overridden.

Warning: As a reviewer of a PR involving changes to this resource file, you should verify the changes to the file by running `pyrcc5` locally.

9.4 Adding and Running Test Cases

We have Python unit tests in the `test_gui.py` file. Any change in the actual application code will also require adding new test cases or updating the old ones. You can run the tests using the following command:

```
$ python3 test_gui.py
```

VIRTUAL ENVIRONMENTS: SERVERS

SecureDrop is a multi-server system, and you may need the full server stack available in order to develop and test some features. To make this easier, the project includes a Vagrantfile that can be used to create two predefined virtual environments:

- *Staging*
- *Production*

This document explains the purpose of, and how to get started working with, each one.

Note: If you plan to alter the configuration of any of these machines, make sure to review the *Testing: Configuration Tests* documentation.

10.1 Staging

A compromise between the development and production environments. This configuration can be thought of as identical to the production environment, with a few exceptions:

- The Debian packages are built from your local copy of the code, instead of installing the current stable release packages from <https://apt.freedom.press>.
- The staging environment is configured for direct SSH access so it's more ergonomic for developers to interact with the system during debugging.
- The Postfix service is disabled, so OSSEC alerts will not be sent via email.

This is a convenient environment to test how changes work across the full stack.

You should first bring up the VM required for building the app code Debian packages on the staging machines:

```
make build-debs
make staging
molecule login -s libvirt-staging-focal -h app-staging
sudo -u www-data bash
cd /var/www/securedrop
./manage.py add-admin
```

To rebuild the local packages for the app code and update the staging VMs:

```
make build-debs
make staging
```

The Debian packages will be rebuilt from the current state of your local git repository and then installed on the staging servers.

The web interfaces and SSH are available over Tor. A copy of the the Onion URLs for *Source* and *Journalist Interfaces*, as well as SSH access, are written to the Vagrant host's `install_files/ansible-base` directory.

To access the *Source Interface* from Tor Browser, use the v3 onion URL from the file `install_files/ansible-base/app-sourcev3-ths`.

To use the *Journalist Interface*, you will need to modify Tor Browser's configuration to allow access to an authenticated onion service:

- First, add the following line to your Tor Browser's `torrc` file, typically found at `tor-browser_en-US/Browser/TorBrowser/Data/Tor/torrc`:

```
ClientOnionAuthDir TorBrowser/Data/Tor/onion_auth
```

- Next, create the `onion_auth` directory:

```
mkdir tor-browser_en-US/Browser/TorBrowser/Data/Tor/onion_auth
chmod 0700 tor-browser_en-US/Browser/TorBrowser/Data/Tor/onion_auth
```

- Finally, copy the file `install_files/ansible-base/app-journalist.auth_private` to the `onion_auth` directory and restart Tor Browser. You should now be able to visit the v3 onion address in `app-journalist.auth_private` from Tor Browser.

For working on OSSEC monitoring rules with most system hardening active, update the OSSEC-related configuration in `install_files/ansible-base/staging.yml` so you receive the OSSEC alert emails.

Direct SSH access is available for staging hosts, so you can use `molecule login -s <scenario> -h app-staging`, where `<scenario>` is either `libvirt-staging-focal` or `qubes-staging-focal`, depending on your environment.

By default, the staging environments are created with an empty submissions database. If you want to set up a staging environment with a preexisting submissions database, you can do so using a SecureDrop backup file as follows:

- Create a directory `install_files/ansible-base/test-data`.
- Copy the backup file to the directory above.
- Define an environmental variable `TEST_DATA_FILE` whose value is the name of the backup file - for example `sd-backup.tar.gz` - and run `make staging`:

```
TEST_DATA_FILE="sd-backup.tar.gz" make staging
```

A staging environment will be created using the submissions and account data from the backup, but ignoring the backup file's Tor configuration data.

Note: It is not recommended to use backup data from a live SecureDrop installation in staging, as the backup may contain sensitive information and the staging environment should not be considered secure.

When finished with the Staging environment, run `molecule destroy -s <scenario>` to clean up the VMs. If the host machine has been rebooted since the Staging environment was created, Molecule will fail to find the VM info, as it's stored in `/tmp`. If you use `libvirt`, run `virt-manager` and destroy the staging VMs manually, by right-clicking on the entries and choosing **Destroy**.

10.2 Production

This is a production installation with all of the system hardening active, but virtualized, rather than running on hardware. You will need to use a virtualized Admin Workstation in order to provision these machines.

10.2.1 Switching to the Vagrant libvirt provider

Make sure you've already installed Vagrant, as described in the [multi-machine setup docs](#).

Ubuntu 20.04 setup

Install libvirt and QEMU:

```
sudo apt-get update
sudo apt-get install libvirt-bin libvirt-dev qemu-utils qemu virt-manager
sudo /etc/init.d/libvirt-bin restart
```

Add your user to the libvirtd group:

```
sudo addgroup libvirtd
sudo usermod -a -g libvirtd $USER
```

Install the required Vagrant plugins for converting and using libvirt boxes:

```
vagrant plugin install vagrant-libvirt
vagrant plugin install vagrant-mutate
```

Note: If Vagrant is already installed it may not recognize libvirt as a valid provider. In this case, remove Vagrant with `sudo apt-get remove vagrant` and reinstall it.

Log out, then log in again. Verify that libvirt is installed and KVM is available:

```
libvirtd --version
kvm-ok
```

Debian stable setup

Install Vagrant, libvirt, QEMU, and their dependencies:

```
sudo apt-get update
sudo apt-get install -y vagrant vagrant-libvirt libvirt-daemon-system qemu-kvm virt-
↪manager
sudo apt-get install -y ansible rsync
vagrant plugin install vagrant-libvirt
vagrant plugin install vagrant-mutate
sudo usermod -a -G libvirt $USER
sudo systemctl restart libvirtd
```

Add your user to the kvm group to give it permission to run KVM:

```
sudo usermod -a -G kvm $USER
sudo rmmmod kvm_intel
sudo rmmmod kvm
sudo modprobe kvm
sudo modprobe kvm_intel
```

Log out, then log in again. Verify that libvirt is installed and your system supports KVM:

```
sudo libvirtd --version
[ `egrep -c 'flags\s*:.*(vmx|svm)' /proc/cpuinfo` -gt 0 ] && \
echo "KVM supported!" || echo "KVM not supported..."
```

Set libvirt as the default provider

Set the default Vagrant provider to libvirt:

```
echo 'export VAGRANT_DEFAULT_PROVIDER=libvirt' >> ~/.bashrc
export VAGRANT_DEFAULT_PROVIDER=libvirt
```

Convert Vagrant boxes to libvirt

Convert the VirtualBox images for Focal from `virtualbox` to `libvirt` format:

```
vagrant box add --provider virtualbox bento/ubuntu-20.04
vagrant mutate bento/ubuntu-20.04 libvirt
```

You can now use the libvirt-backed VM images to develop against the SecureDrop multi-machine environment.

10.2.2 Install from an Admin Workstation VM

In SecureDrop, admin tasks are performed from a Tails *Admin Workstation*. You should configure a Tails VM in order to install the SecureDrop production VMs by following the instructions in the [Virtualizing Tails](#) guide.

Once you're prepared the *Admin Workstation*, you can start each VM:

```
molecule create -s libvirt-prod-focal
```

At this point you should be able to SSH into both `app-prod` and `mon-prod` with the user `vagrant` and the password `vagrant`.

From here you can follow the [:server configuration instructions](#) to test connectivity and prepare the servers.

These instructions will have you generate SSH keys and use `ssh-copy-id` to transfer the key onto the servers. By default, the Vagrant boxes authorize a publicly provided [SSH keypair](#), which you can download on Tails and import via `ssh-add` instead of generating a new SSH keypair.

Note: If you have trouble SSHing to the servers from Ansible, remember to remove any old ATHS files in `install_files/ansible-base`.

Now from your *Admin Workstation*, set up the administration environment with:

```
cd ~/Persistent/securedrop
./securedrop-admin setup
```

If you want to enable HTTPS for the source interface, you can generate a test CA cert, server key, and server cert using the following commands:

```
sudo apt-get install make
make self-signed-https-certs
```

This will generate the files `securedrop_source_onion.ca`, `securedrop_source_onion.crt`, and `securedrop_source_onion.key` in the `install_files/ansible-base` directory, ready for use in the next step.

Important: The self-signed certificates should not be used in a live instance. They are provided for development and testing purposes only.

To proceed with a full install, you will need, at a minimum:

- The IP addresses of the two virtualized servers, `app-prod` and `mon-prod`. You can obtain them via `virsh domifaddr libvirt-prod-focal_app-prod` and `virsh domifaddr libvirt-prod-focal_mon-prod`.
- The username and sudo password (both default to `vagrant` for both servers)
- A *Submission Public Key*. `securedrop-admin` will reject the key included with the development environment. For testing purposes only, you can create a new keypair within the Tails VM.
- An *OSSEC Alert Public Key*. We recommend using your own public key if you intend to test OSSEC email functionality.

Configure and install SecureDrop on the server VMs using the commands:

```
./securedrop-admin sdconfig
./securedrop-admin install
```

After the installation is complete, you can configure your *Admin Workstation* to SSH into each VM via:

```
./securedrop-admin tailsconfig
```

`securedrop-admin` will write the SecureDrop configuration to `~/Persistent/securedrop/install_files/ansible-base/group_vars/all/site-specific`. To simplify subsequent installs, you may wish to make a copy of this file, as well as the two required public keys, in a directory in `~/Persistent` or outside the Tails VM.

VIRTUAL ENVIRONMENTS: ADMIN WORKSTATION

SecureDrop uses Tails for the *Admin Workstation* environment. In order to perform a fully virtualized production install, you will need to first set up Tails in a virtual machine.

Note: For the instructions that follow, you need to download the most recent Tails ISO from the [Tails](#) website.

Only libvirt-based virtualization, on a Linux host, is supported.

11.1 Linux

For the Linux instructions, you will use KVM/libvirt to create a Tails VM that you can use to install SecureDrop on `app-prod` and `mon-prod`.

11.1.1 Create a VM using virt-manager

Follow the Tails virt-manager instructions for [running Tails from a USB image](#). Then proceed with booting to the USB drive, and [configure Persistent Storage](#).

We recommend cloning the SecureDrop repository into the persistent volume for testing and development, instead of attempting to mount a folder from the host operating system.

VIRTUAL ENVIRONMENTS: USING QUBES

SecureDrop currently uses Ubuntu Focal as its server OS. The instructions below cover setting up a SecureDrop staging environment using Focal under Qubes.

It is assumed that you have an up-to-date Qubes R4.1 installation on a compatible laptop, with at least 16GB RAM and 60GB free disk space. The SecureDrop server VMs run Tor locally instead of using `sys-whonix`, so the system clock must be set accurately for Tor to start and hidden services to be available.

12.1 Overview

Follow the the Qubes platform instructions in [Setting Up the Development Environment](#) to create a Debian 10 `sd-dev` Standalone VM. Once done, we'll create three new Standalone (HVM) Qubes VMs for use with staging:

- `sd-staging-base-focal`, a base VM for cloning reusable staging VMs
- `sd-staging-app-base-focal`, a base VM for the *SecureDrop Application Server*
- `sd-staging-mon-base-focal`, a base VM for the *SecureDrop Monitor Server*

12.2 Download Ubuntu server ISO

On `sd-dev`, download the latest Ubuntu server ISO for Focal, along with corresponding checksum and signature files. See the [installation docs](#) for detailed instructions. If you opt for the command line instructions, omit the `torify` prepended to the `curl` command.

12.3 Create the base VM

We're going to build a single, minimally configured Ubuntu VM. Once it's bootable, we'll clone it for the application and monitoring VMs.

In `dom0`, do the following:

```
qvm-create sd-staging-base-focal --class StandaloneVM --property virt_mode=hvm --label_↵
↵green
qvm-volume extend sd-staging-base-focal:root 20g
qvm-prefs sd-staging-base-focal memory 2000
qvm-prefs sd-staging-base-focal maxmem 2000
qvm-prefs sd-staging-base-focal kernel ''
```

The commands above will create a new StandaloneVM, expand the storage space and memory available to it, as well as disable the integrated kernel support. The SecureDrop install process will install a custom kernel.

12.4 Boot into installation media

In dom0:

```
qvm-start sd-staging-base-focal --cdrom=sd-dev:$ISO_PATH
```

where ISO_PATH is the full path to the Ubuntu ISO previously downloaded on sd-dev.

Next, choose **Install Ubuntu**.

For the most part, the install process matches the [hardware install flow](#), with a few exceptions:

- **Subnet:** 10.137.0.0/24
- **Address:** use value returned by `qvm-prefs sd-staging-base-focal ip`
- **Gateway:** use value returned by `qvm-prefs sd-staging-base-focal visible_gateway`
- **Name servers:** 10.139.1.1,10.139.1.2
- **Search domains:** *should be left blank*
- **Your server's name:** sd-staging-base-focal

Make sure to configure LVM and use **Virtual disk 1 (xvda 20.0GB Xen Virtual Block device)** when asked for a target partition during installation. It should be the default option.

You'll be prompted to add a "regular" user for the VM: this is the user you'll be using later to SSH into the VM. We're using a standardized name/password pair: `sdadmin/securedrop`.

Once installation is done, let the machine shut down and then restart it with

```
qvm-start sd-staging-base-focal
```

in dom0. You should get a login prompt.

12.5 Initial VM configuration

Before cloning this machine, we'll update software to reduce provisioning time on the staging VMs. In the new sd-staging-base-focal VM's console, do:

```
sudo apt update
sudo apt dist-upgrade -y
```

Before we continue, let's allow your user to `sudo` without their password. Edit `/etc/sudoers` using `visudo` to make the `sudo` group line look like

```
%sudo ALL=(ALL) NOPASSWD: ALL
```

Finally, update the machine's Grub configuration to use a consistent Ethernet device name across kernel versions. Edit the file `/etc/default/grub`, changing the line:

```
GRUB_CMDLINE_LINUX=""
```


to

```
GRUB_CMDLINE_LINUX="net.ifnames=0 biosdevname=0"
```

When initial configuration is done, run `qvm-shutdown sd-staging-base-focal` to shut it down.

12.6 Clone VMs

We're going to configure the VMs to use specific IP addresses, which will make various routing issues easier later. We'll also tag the VMs for management by the `sd-dev` VM. Doing so will require Qubes RPC policy changes, documented below. Run the following in `dom0`:

```
qvm-clone sd-staging-base-focal sd-staging-app-base-focal
qvm-clone sd-staging-base-focal sd-staging-mon-base-focal
qvm-prefs sd-staging-app-base-focal ip 10.137.0.50
qvm-prefs sd-staging-mon-base-focal ip 10.137.0.51
qvm-tags sd-staging-app-base-focal add created-by-sd-dev
qvm-tags sd-staging-mon-base-focal add created-by-sd-dev
```

Now start both new VMs:

```
qvm-start sd-staging-app-base-focal
qvm-start sd-staging-mon-base-focal
```

On the consoles which eventually appear, you should be able to log in with `sdadmin/securedrop`.

12.6.1 Configure cloned VMs

We'll need to fix each machine's idea of its own IP. In the console for each machine, edit `/etc/netplan/00-installer-config.yaml` to update the `addresses` entry with the machine's IP.

Edit `/etc/hosts` on each host to include the hostname and IP for itself. Use `app-staging` and `mon-staging` as appropriate.

Next, on each host edit `/etc/hostname` to reflect the machine's name.

Halt each machine, then restart each from `dom0`. The prompt in each console should reflect the correct name of the VM. Confirm you have network access by running `ping freedom.press`. It should show no errors.

12.6.2 Inter-VM networking

We want to be able to SSH connections from `sd-dev` to these new standalone VMs. In order to do so, we have to adjust the firewall rules. Make the following changes on `fedora-36-dvm`, which is the template for `sys-firewall` under a default setup.

Note: These changes to the firewall rules will also apply to all other DispVMs based off `fedora-36-dvm`, and are meant for a testing/development machine only.

Let's get the IP address of `sd-dev`. On `dom0`:

```
qvm-prefs sd-dev ip
```

Get a shell on `fedora-36-dvm`. Create or edit `/rw/config/qubes-firewall-user-script`, to include the following:

```
sd_dev="<sd-dev-addr>"
sd_app="10.137.0.50"
sd_mon="10.137.0.51"

iptables -I FORWARD 2 -s "$sd_dev" -d "$sd_app" -j ACCEPT
iptables -I FORWARD 2 -s "$sd_dev" -d "$sd_mon" -j ACCEPT
iptables -I FORWARD 2 -s "$sd_app" -d "$sd_mon" -j ACCEPT
iptables -I FORWARD 2 -s "$sd_mon" -d "$sd_app" -j ACCEPT
```

Shut down `fedora-36-dvm`, then restart `sys-firewall`.

Now from `sd-dev`, you should be able to do

```
ssh sdadmin@10.137.0.50
```

and log in with the password `securedrop`.

Tip: See the official Qubes guide on configuring [inter-VM networking](#) for more details.

12.6.3 SSH using keys

Tip: You likely already have an SSH keypair configured for access to GitHub. If not, create one with `ssh-keygen -b 4096 -t rsa`. The configuration logic will use the key at `~/.ssh/id_rsa` to connect to the VMs.

Later we'll be using Ansible to provision the application VMs, so we should make sure we can SSH between those machines without needing to type a password. On `sd-dev`:

```
ssh-copy-id sdadmin@10.137.0.50
ssh-copy-id sdadmin@10.137.0.51
```

Confirm that you're able to ssh as user `sdadmin` from `sd-dev` to both IP addresses without a password.

12.7 SecureDrop Installation

We're going to configure `sd-dev` to build the SecureDrop `.deb` files, then we're going to build them, and provision `sd-staging-app` and `sd-staging-mon`. Follow the instructions in the [developer documentation](#) to set up the development environment.

Once finished, build the Debian packages for installation on the staging VMs:

```
make build-debs
```

12.8 Managing Qubes RPC for Admin API capability

We're going to be running Qubes management commands on `sd-dev`, which requires some additional software. Install it with

```
sudo apt install qubes-core-admin-client
```

You'll need to grant the `sd-dev` VM the ability to create other VMs, by editing the Qubes RPC policies in `dom0`. Here is an example of a permissive policy, sufficient to grant `sd-dev` management capabilities over VMs it creates. The lines below should be inserted at the beginning of their respective policy files, before other more general rules:

```
/etc/qubes/policy.d/include/admin-local-rwx:
sd-dev @tag:created-by-sd-dev allow target=@adminvm

/etc/qubes/policy.d/include/admin-global-rwx:
sd-dev @adminvm allow target=@adminvm
sd-dev @tag:created-by-sd-dev allow target=@adminvm
```

Tip: See the Qubes documentation for details on leveraging the [Admin API](#).

12.9 Creating staging instance

After creating the StandaloneVMs as described above:

- `sd-dev`
- `sd-staging-base-focal`
- `sd-staging-app-base-focal`
- `sd-staging-mon-base-focal`

And after building the SecureDrop `.debs`, we can finally provision the staging environment:

```
make staging
```

The commands invoke the appropriate Molecule scenario for your choice of `focal`. You can also run constituent Molecule actions directly, rather than using the Makefile target:

```
molecule create -s qubes-staging-focal
molecule converge -s qubes-staging-focal
molecule test -s qubes-staging-focal
```

That's it. You should now have a running, configured SecureDrop staging instance running on your Qubes machine. For day-to-day operation, you should run `sd-dev` in order to make code changes, and use the Molecule commands above to provision staging VMs on-demand. To remove the staging instance, use the Molecule command:

```
molecule destroy -s qubes-staging-focal
```

12.10 Accessing the Journalist Interface (Staging) in Whonix-based VMs

Warning: These instructions are only appropriate for a staging setup and should not be used to access a production instance of SecureDrop.

To access the *Source* and *Journalist Interfaces* (staging) in a Debian- or Fedora-based VM, follow the instructions [here](#).

To use a Whonix-based VM, the following steps are required to configure access to the *Journalist Interface* (staging).

12.10.1 In sd-dev

You will have to copy the `app-journalist.auth_private` file (located in your `sd-dev` VM in `${SECUREDROP_HOME}/install_files/ansible-base` and generated after a successful staging build) into your Whonix gateway VM. On standard Qubes installations this VM is called `sys-whonix`.

To do this, in an `sd-dev` terminal, run the command:

```
qvm-copy ${SECUREDROP_HOME}/install_files/ansible-base/app-journalist.auth_private
```

and select `sys-whonix` in the resulting permissions dialog.

12.10.2 In the Whonix Gateway

Open a terminal in `sys-whonix` and create a directory with appropriate ownership and permissions, then move your credential file there:

```
sudo mkdir -p /var/lib/tor/onion_auth
sudo mv ~/QubesIncoming/sd-dev/app-journalist.auth_private /var/lib/tor/onion_auth
sudo chown --recursive debian-tor:debian-tor /var/lib/tor/onion_auth
```

Next, edit the Tor configuration so it recognizes the directory containing your credentials:

```
sudo vi /usr/local/etc/torrc.d/50_user.conf
```

In this file, enter the following:

```
ClientOnionAuthDir /var/lib/tor/onion_auth
```

Save and close the file. Finally, reload Tor by clicking **Qubes Application Menu > sys-whonix > Reload Tor**

At this point, you should be able to access the *Journalist Interface* (staging) in a Whonix VM that uses `sys-whonix` as its gateway.

Note that you will have to replace the `app-journalist.auth_private` file and reload Tor on the Whonix gateway every time you rebuild the staging environment.

UPGRADE TESTING USING MOLECULE

The SecureDrop project includes Molecule scenarios for developing and testing against multi-server configurations, including a scenario to simulate the process of upgrading an existing system. This document explains how to work with this scenario to test features that make potentially release-breaking changes such as database schema updates.

The Molecule upgrade scenario sets up a local apt server, to imitate how new package versions will be installed in production. You'll need to use a virtualized Admin Workstation to configure the base server VMs with the current stable version, prior to testing the upgrade.

Note: The upgrade scenario uses QEMU/KVM via Vagrant's libvirt provider. If you haven't already done so, you'll need to set up the libvirt provider before proceeding. For more information, see [Switching to the Vagrant libvirt provider](#).

13.1 Upgrade testing using locally-built packages

First, create prod VMs for use with the current stable version. These machines will be upgraded with newer, locally built deb packages in a subsequent step.

```
molecule create -s libvirt-prod-focal
```

Next, boot your Admin Workstation VM and proceed with a full install on these VMs, via `./securedrop-admin install`. Make sure to run `./securedrop-admin tailsconfig` to finalize the installation.

Next, build the app code packages and create the environment:

```
make build-debs
make upgrade-start
```

The playbook will create a local apt server on your host machine, and serve the locally built deb packages from that local endpoint. In order to add the local apt server to the VMs, switch back to the Admin Workstation and run:

```
source admin/.venv3/bin/activate
cd install_files/ansible-base
ansible-playbook -vv --diff securedrop-apt-local.yml
```

Both VMs will now be able to view newer, locally built packages. To confirm:

```
ssh app
```

From the *Application Server*:

```
apt-cache policy securedrop-app-code
```

The installed package version should match the latest stable version, but the locally built package with higher version should be available as a candidate for installation.

13.2 Upgrade testing using apt-test.freedom.press

You can also evaluate packages on the <https://apt-test.freedom.press/> repository. As above, create prod VMs and configure them via the Admin Workstation. After installation, you can enable the `apt-test` repo like so:

```
source admin/.venv3/bin/activate
cd install_files/ansible-base
ansible-playbook -vv --diff securedrop-qa.yml
```

Then, log into the *Application Server*:

```
ssh app
apt-cache policy securedrop-config
```

The installed package version should match the latest stable version, with the locally built package of a higher version available as a candidate for installation.

DATABASE MIGRATIONS

SecureDrop uses [Alembic](#) for database schema migrations. This guide is not a complete explanation of what alembic is or how it is used, so the original documentation should be read.

14.1 Migration Files

In the `securedrop/` directory, the file `alembic.ini` contains the configuration needed to run alembic commands, and the directory `alembic/` contains the Python code that executes migrations.

The directory looks like this.

```
.
├── alembic
│   ├── env.py
│   ├── script.py.mako
│   └── versions
│       ├── 15ac9509fc68_init.py
│       └── faac8092c123_enable_security_pragmas.py
└── alembic.ini
```

The subdirectory `versions/` individual migrations that are generated by alembic. In the example above, there are two migrations. alembic orders these migrations based off of values in the Python files, not off any sort of lexicographic ordering. The file `faac8092c123_enable_security_pragmas.py` has a module-level documentation string that specifies that it comes after `15ac9509fc68_init.py` as well as variables used by alembic that specify the ordering of migrations.

14.2 Deployment

Database migrations are automatically applied to production instances via the command `alembic upgrade head` in the `postinst` script in the `securedrop-app-code` Debian package. You do not need to worry about when or how these migrations are applied.

14.3 Developer Workflow

14.3.1 Updating the Models

When you want to modify the database schema, you need to add adjust the models in the file `models.py`. All indices, constraints, or other metadata about the scheme needs to be in this file. The development server creates tables directly from the subclasses of `db.Model` so that they are available for manual and automated testing.

14.3.2 Creating Migrations

Once you are satisfied with your new model, `alembic` can auto-generate migrations using SQLAlchemy metadata and comparing it to the schema of an up-to-date SQLite database. To generate a new migration use the following steps.

```
cd securedrop/  
./bin/dev-shell  
source bin/dev-deps  
maybe_create_config_py  
./bin/new-migration 'my migration message'
```

This will output a new migration into `alembic/versions/`. You will need to verify that this migration produced the desired output. While still in the `dev-shell`, you can run the following command to see an output of the SQL that will be generated.

```
alembic upgrade head --sql
```

14.3.3 Unit Testing Migrations

The test suite already comes with a test runner (`test_alembic.py`) that runs a series of checks to ensure migration's upgrade and downgrade commands are idempotent and don't break the database. The test runner uses dynamic module import to iterate through all the migrations. You will need to create a python module in the `tests/migrations/` directory. Your module **MUST** be named `migration_<revision identifier>.py`. For example, if your revision is named `15ac9509fc68_init.py`, your test module will be named `migration_15ac9509fc68.py`. Example modules for the first two revisions are shown below.

```
tests/migrations/  
├── __init__.py  
├── migration_15ac9509fc68.py  
└── migration_faac8092c123.py
```

Your module **MUST** contain the following classes with the following attributes.

```
class UpgradeTester:  
  
    def __init__(self, config):  
        '''This function MUST accept an argument named `config`.  
        You will likely want to save a reference to the config in your  
        class so you can access the database later.'''  
        self.config = config  
  
    def load_data(self):
```

(continues on next page)

(continued from previous page)

```
    """This function loads data into the database and filesystem. It is
        executed before the upgrade.
    """
    pass

def check_upgrade(self):
    """This function is run after the upgrade and verifies the state
        of the database or filesystem. It MUST raise an exception if the
        check fails.
    """
    pass

class DowngradeTester:

    def __init__(self, config):
        """This function MUST accept an argument named `config`.
            You will likely want to save a reference to the config in your
            class so you can access the database later.
        """
        self.config = config

    def load_data(self):
        """This function loads data into the database and filesystem. It is
            executed before the downgrade.
        """
        pass

    def check_downgrade(self):
        """This function is run after the downgrade and verifies the state
            of the database or filesystem. It MUST raise an exception if the
            check fails.
        """
        pass
```

Your migration test needs to load data that covers all edge cases such as potentially broken foreign keys or columns with unexpected content.

Additionally, your test **MUST NOT** import anything from the `models` module as this will not accurately test your migration, and it will likely break during future code changes. In fact, you should use as few dependencies as possible in your test including other securedrop code as well as external packages. This may be a rather annoying requirement, but it will make the tests more robust against future code changes.

14.3.4 Release Testing Migrations

In order to ensure that migrations between from the previous to current version of SecureDrop apply cleanly in production-like instances, we have a helper script that is designed to load semi-randomized data into the database. You will need to modify the script `loaddata.py` to include sample data. This sample data should intentionally include edge cases that might behave strangely such as data whose nullability is only enforced by the application or missing files.

During QA, the release manager should follow these steps to test the migrations.

1. Checkout the previous SecureDrop release
2. Build Debian packages locally
3. Provision staging VMs
4. `vagrant ssh app-staging`
5. `sudo -u www-data bash`
6. `cd /var/www/securedrop && ./loaddata.py`
7. Checkout the release candidate
8. Re-provision the staging VMs
9. Check that nothing went horribly wrong

INTERNATIONALIZATION (I18N)

SecureDrop is translated into a number of languages. We use a web-based collaborative translation platform called [Weblate](#) to make it easier. Under the hood, all translation is done using GNU [gettext](#).

With [gettext](#), text to be translated is specially marked in source code. A Python example:

```
if not (msg or fh):
    flash(gettext("You must enter a message or choose a file to submit."), "error")
    return redirect(url_for('main.lookup'))
```

In this code, the string `You must enter a message or choose a file to submit.` can be automatically extracted for translation. The `gettext` function to which it is passed is used as a marker by [pybabel](#) or similar tools to collect the strings to be translated and store them into a `.pot` file at `securedrop/translations/messages.pot`. For instance:

```
#: source_app/main.py:111
msgid "You must enter a message or choose a file to submit."
msgstr ""
```

The `.pot` file serves as a template for all the language-specific `.po` files, which are where Weblate stores the contributed translations. For each language to be translated, a directory is created, such as `securedrop/translations/fr_FR`, and populated with a `.po` file derived from the template. For instance, `securedrop/translations/fr_FR/LC_MESSAGES/messages.po` is almost identical to `securedrop/translations/messages.pot` except for the `msgstr` fields, which will contain the French translations, e.g.:

```
#: source_app/main.py:111
msgid "You must enter a message or choose a file to submit."
msgstr "Vous devez saisir un message ou sélectionner un fichier à envoyer."
```

There's one last type of file in the [gettext](#) system, a machine-readable version of the `.po` translations called a `.mo` file. Applications use these to get translations at runtime. The `.po` files are compiled to `.mo` files when the SecureDrop package is built.

The desktop icons installed on SecureDrop workstations are also translated. The icon templates are in the `install_files/ansible-base/roles/tails-config/templates` directory. Their labels are collected in the `desktop.pot` file and translated in the corresponding `.po` files in the same directory (`fr.po`, `de.po` etc.). All translations are merged from the `*.j2.in` files into the corresponding `*.j2` file and committed to the SecureDrop repository. They are then installed when configuring Tails with the `tasks/create_desktop_shortcuts.yml` tasks.

We don't expect translators to deal with all these files directly. Translation happens on our [Weblate](#) server, which is configured to use a fork of the [main SecureDrop repository](#).

At the start of the *release process*, the localization manager collects string changes on the `develop` branch in the [main SecureDrop repository](#) and merges them to the `i18n` branch of the [securedrop-i18n repository](#). The changes will then appear in Weblate, and translation can begin. At the end of the translation period, the localization manager collects

the changes to the PO files on `securedrop-i18n/i18n` in a pull request for `securedrop/develop`. Once that pull request is merged, the translations are backported to the release branch in the [main SecureDrop repository](#).

15.1 i18n_tool.py

Most of the work in managing translations within the SecureDrop code base is supported by `securedrop/i18n_tool.py`. It provides convenient wrappers around `pybabel` and `gettext`, and is used to update strings to be translated; pull translations from Weblate; to compile translations before running tests and while packaging SecureDrop.

15.2 Development tasks

15.2.1 Add a new language

A user with weblate admin rights must visit the [Weblate translation creation page](#) and the [Weblate desktop translation creation page](#) to add a new language.

SecureDrop only supports a subset of all the languages being worked on in [Weblate](#): some of them are partially translated or not fully reviewed. The list of fully supported languages is hard-coded in the `i18n_tool.py` file, in the `SUPPORTED_LANGUAGES` variable. When a new language is completely translated and reviewed, the `i18n_tool.py` file must be manually edited to add this new language to the `SUPPORTED_LANGUAGES` variable.

15.2.2 Update strings to be translated

Whenever strings are modified in the SecureDrop source, whether in Python code, HTML templates, or desktop icon labels, the translation files should also be updated by running `make translate` in the root of the SecureDrop working copy.

The `translate` target runs `i18n_tool.py translate-messages` and `i18n_tool.py translate-desktop`, which in turn use `pybabel extract` to gather source strings. These commands update the `.pot` files for the SecureDrop server code and the desktop icons, as well as the `.po` files for each language.

After running `make translate`, carefully review the output of `git diff`. Check `securedrop/messages.pot` first for updated strings, looking for problems like:

- overly idiomatic English
- fragmented text, such as pieces of a sentence intended to be concatenated together, which can be difficult to translate
- messages that are marked with plain `gettext` and contain plurals based on numeric placeholder variables – these should generally be marked with `ngettext` so that they can be translated properly in languages with complex plural forms

Then review the `messages.po` of one existing translation, with a focus on new translations, which are often marked `fuzzy`. There is no need to review multiple languages' `.po` files because they are processed in the same way.

Once you've reviewed the changes, submit them in a pull request for the `develop` branch in the [main SecureDrop repository](#).

The new source strings will only be visible to translators in [Weblate](#) after they've been merged to `securedrop/develop` and that branch has been merged into `securedrop-i18n/i18n`. The localization manager does this at the beginning of our release cycle.

15.2.3 Merge develop into the Weblate fork

1) First make sure the translation files on the `develop` branch of the main SecureDrop repository contain the latest source strings. Follow the steps under *Update strings to be translated*.

2) Then, translation must be suspended in Weblate, and any uncommitted changes committed and pushed, to avoid conflicts:

- Go to the Weblate repository page for SecureDrop.

The screenshot shows the Weblate web interface for the SecureDrop project. At the top, there's a navigation bar with 'Weblate', 'Dashboard', and 'Documentation'. The user 'Kushal Das' is logged in. Below the navigation bar, there's a header for 'SecureDrop' with a 'translated 64%' badge. A green message box states 'Project is now open for translation updates.' Below that, a light blue box provides information about the 0.11.0 string freeze. The main content area has a 'Repository status' section showing 'The local repository is up to date.' and a 'Repository details' section showing 'SecureDrop/SecureDrop' on branch 'i18n'. To the right, a 'Repository tools' panel lists actions: Commit, Pull, Push (highlighted), Rebase, Reset, and Lock. The 'Push' button is highlighted with a blue border.

- Click **Commit**.
- Click **Push**.
- And finally, click **Lock**.

The screenshot shows the Weblate web interface for the SecureDrop project after locking. The header shows 'translated 37%'. A green message box states 'Component is now locked for translation updates!'. Below that, a yellow box states 'This translation is currently locked for updates!'. The main content area has a 'Translations' section with a table showing the status of translations for different languages.

Language	Translated	Words	Review	Checks	Suggestions	
Arabic	11.0%	9.2%	5.5%	4	0	Translate
Bengali (Bangladesh)	0.0%	0.0%	0.0%	0	0	Translate

3) The `securedrop/develop` branch can now be merged into `securedrop-i18n/i18n`:

```
$ git clone https://github.com/freedomofpress/securedrop
$ cd securedrop
```

(continues on next page)

(continued from previous page)

```
$ git remote add i18n git@github.com:freedomofpress/securedrop-i18n.git
$ git fetch i18n
$ git checkout -b i18n i18n/i18n
$ git merge origin/develop
$ git commit --amend -m 'l10n: sync with upstream origin/develop'
$ git push i18n i18n
```

4) Verify that Weblate has the latest changes, and unlock the repository.

- Go to the [Weblate commit page for SecureDrop](#) and verify the commit hash matches the last commit of the `i18n` branch. This must happen instantly after the branch is pushed because Weblate is notified via a webhook. If it is different, [ask for help](#).
- Click **Unlock**.

Translation can now begin. As translators make progress, [Weblate](#) pushes the translations done via the web interface in commits to the `i18n` branch of the [securedrop-i18n](#) repository (a fork of the [main SecureDrop](#) repository). When the translation period ends, these commits will be collected into a pull request for the [main SecureDrop](#) repository.

15.2.4 Merge translations back to develop

Weblate automatically pushes the translations done via the web interface as a series of commits to the `i18n` branch in the `securedrop-i18n` repository, which is a fork of the `develop` branch of the main `SecureDrop` repository. These translations need to be submitted back to the `securedrop/develop` branch via pull requests. When you create a branch for this, begin its name with `i18n-`, as that prefix triggers special CI tests for translations.

To fetch the latest translations from the `securedrop-i18n/i18n` branch into your working copy of the SecureDrop repository, run these commands in your repo root:

```
$ git checkout -b i18n-merge origin/develop
$ securedrop/bin/dev-shell ./i18n_tool.py --verbose update-from-weblate
$ securedrop/bin/dev-shell ./i18n_tool.py --verbose update-docs --docs-repo-dir /path/to/
↪ documentation
```

You now have the latest translations on your `i18n-merge` branch.

Note: It is **very** important to check that each translated string looks like a plausible translation, with no markup. Even if the reviewer does not understand the language, if a translated string looks strange, someone other than the reviewer must be consulted to verify it means something. It is extremely unlikely that a contributor will manipulate a translated string to introduce a vulnerability in SecureDrop, but any suspicious translation should be investigated.

To check the new translations, you'll need to compile them and verify them by running our automated tests and, ideally, by checking them in the SecureDrop source and journalist interfaces.

Compile translations

At runtime, `gettext` needs a compiled file for each language (the `.mo` files). Before you can check the translations in the SecureDrop web interfaces, these need to be created:

```
$ securedrop/bin/dev-shell ./i18n_tool.py --verbose translate-messages --compile
```

For the desktop icons of the source and journalist interfaces, compilation updates their template files with all the translations collected from the `.po` files.

This can be done by running the following command:

```
$ securedrop/bin/dev-shell ./i18n_tool.py --verbose translate-desktop --compile
```

Verify translations

SecureDrop web interfaces

After a translation is compiled, the web page in which it appears can be verified visually by starting the SecureDrop development servers and navigating via `http://localhost:8080` for the source interface or `http://localhost:8081` for the journalist interface. You can start the development servers with:

```
$ make dev
```

The translations can be checked automatically by running the SecureDrop page layout tests:

```
$ export PAGE_LAYOUT_LOCALES="en_US,fr_FR" # may be set to any supported languages
$ make test TESTFILES=tests/pageslayout
[...]
tests/pageslayout/test_journalist.py::TestJournalistLayout::test_account_edit_hotp_
↪secret[en_US] PASSED
tests/pageslayout/test_journalist.py::TestJournalistLayout::test_account_edit_hotp_
↪secret[fr_FR] PASSED
[...]
```

Note: if unset, `PAGE_LAYOUT_LOCALES` defaults to `en_US` (US English) and `ar` (Arabic).

After running the tests, screenshots for each locale are available in `securedrop/tests/pageslayout/screenshots/<locale>`, e.g. `securedrop/tests/pageslayout/screenshots/fr_FR`. Screenshot filenames can be found in the tests that created them, in `securedrop/tests/pageslayout/test_journalist.py` or `securedrop/tests/pageslayout/test_source.py`.

Desktop icons

The translated templates for the desktop icons are:

- `install_files/ansible-base/roles/tails-config/templates/desktop-journalist-icon.j2`
- `install_files/ansible-base/roles/tails-config/templates/desktop-source-icon.j2`

Check that each of them contains a `Name` line for each of SecureDrop’s supported locales.

Push your branch and create a pull request

After you’ve checked the translations, you’re ready to push your `i18n-merge` branch and create a pull request to get the translations merged to the SecureDrop `develop` branch.

Note: If there have been multiple commits per language, as can happen if source strings need to be translated again after being changed to correct critical errors, or to incorporate suggestions from the source string feedback period, they should be combined via an interactive rebase. Reorder the commits to group them by language, then squash the commits for each language into one. The goal is to end up with one commit per supported language on the merge branch.

When you’re happy with the state of language commits on your merge branch:

```
$ git commit -m "i18n: compile desktop icons' translations" # if needed
$ git push -u origin i18n-merge
```

Note: The CI job `translation-tests` will automatically run the above page layout tests in all supported languages on branches named with the prefix `i18n-`. If you’ve followed that naming convention, the translation tests should soon be run on your pull request.

If you have an abundance of time, you can run all the translation tests locally with:

```
$ make translation-test
```


And at long last, you're done. Go to <https://github.com/freedomofpress/securedrop> and propose a pull request.

Note: Unlike the SecureDrop application translations, the desktop icon translations are compiled and merged into the repository. They need to be available in their translated form when `securedrop-admin tailsconfig` is run, because the development environment is not available.

15.2.5 Update Weblate screenshots

You can use the script `securedrop/upload_screenshots.py` to update UI screenshots that are used to illustrate strings in Weblate. The script depends on the existence of up-to-date layout test results, which you can generate using this command in the base directory:

```
$ LOCALES=en_US make translation-test
```

Inspect the screenshots in the directory `securedrop/tests/pageslayout/screenshots/en_US` and make sure that their content corresponds to the expected version of the codebase.

Obtain your [API key](#) in Weblate. Export the token to the environment variable `WEBLATE_API_TOKEN`. You can now run this command to perform an upload:

```
$ securedrop/upload-screenshots.py
```

If new screenshots were added as part of this run, make sure to associate them with relevant strings in Weblate, which you can do from the [screenshots list](#).

15.3 Release Management

15.3.1 Two weeks before the release: string freeze

When features for a new SecureDrop release are frozen, the localization manager for the release will:

- *Merge develop into the Weblate fork.*
- *Update Weblate screenshots* so translators can see new or modified source strings in context.
- Update the [i18n timeline](#) in the translation section of the forum.
- Add a [Weblate announcement](#) for the `securedrop/securedrop` component with the translation timeline for the release.
 - **Important:** Make sure the `Notify users` box is checked, so that translators receive an email alert.
 - You can view a history of past announcements in Weblate's [Django admin panel](#), or use this template:

Translation for the SecureDrop X.Y.Z release has begun. If you have suggestions for source strings, please get them to us by YYYY-MM-DD. Translation will end on YYYY-MM-DD.
 - Set the **Expiry date** to release day itself (the day *after* the translation deadline).
- Remind all developers about the string freeze in [Gitter](#), for example using this template:

Hello! We've just opened translations for the upcoming SecureDrop 2.3.0 release. If you have suggestions for source strings, please get them to us by 2022-03-20. Translation will end on 2022-03-27.

Translations are done using Weblate (<https://weblate.securedrop.org/projects/securedrop/securedrop/>). If you haven't used it before, <<https://docs.securedrop.org/en/stable/development/translations.html>> has instructions on how to get started.

- Update Localization Lab via the [SecureDrop Coordination](#) channel in the [IFF Mattermost](#).
- During the feedback period, monitor Weblate comments and suggestions, and open a pull request for every source string suggestion coming from translators.

Remember that *supported languages* are the priority during this period. That is, while translation contributions are welcome for all languages, the pre-release goal is to keep the current set of supported languages at 100% translation in Weblate. Localization Lab can marshal individual translators to help meet this goal.

15.3.2 Release day

Prior to cutting the final release, the localization manager must:

- *Merge translations back to develop* (see [example PR](#))
- Submit a backport PR of these changes into the release branch (see [example PR](#))
- *Update the documentation screenshots*.
- Provide translator credits to add to the SecureDrop release announcement.

Then, post-release, either same day or day-after, the localization manager should:

- Remove the [Weblate announcement](#) about this release's translation timeline (if you set an end-date on the original announcement, this may happen automatically)
- Update the [i18n timeline](#) in the forum.

15.3.3 Translator credits

Correct acknowledgment of translators' contributions is important, so `i18n_tool.py` makes it easy to list the translators who have helped since the last merge of Weblate translations, with `i18n_tool.py list-translators`. A list of everyone who has ever contributed translations to SecureDrop can be obtained with `i18n_tool.py list-translators --all`. There are Makefile targets for these, `list-translators` and `list-all-translators`, e.g:

```
$ make list-all-translators
ar:
  A. Nonymous
  Ahmad Gharbeia
  Ahmed Essam
  Ali Boshanab
[...]
```

15.4 Weblate administration

Note: The privilege escalation workflow is different for *code maintainers* and *translation maintainers*.

A translation admin has special permissions on Weblate and the repositories. When someone is willing to become an admin, a thread is started in [the translation section of the forum](#). If there is consensus after a week, the permissions of the new admin are elevated. If there is not yet consensus, a public vote is organized among the current admins.

The privileges of an admin who has not been active for six months or more are revoked, but they can apply again at any time.

The community of SecureDrop translators works very closely with the SecureDrop developers and some of them participate in both groups. However, the translator community has a different set of rules and permissions, and therefore independent policies from SecureDrop itself.

15.4.1 Admin permissions

The full set of admin permissions can be granted at:

- https://weblate.securedrop.org/admin/weblate_auth/user/ (grant staff and superuser status)
- <https://forum.securedrop.org/admin/users/list/active> (click on the user and Grant Moderation)
- <https://github.com/freedomofpress/securedrop-i18n> (make sure that the user has commit access)

15.4.2 Granting reviewer privileges in Weblate

- Visit https://weblate.securedrop.org/admin/weblate_auth/user/.
- Click on the user name.
- **In the Groups block:**
 - Select Localizationlab in the Available groups list and click on the right arrow to move it to the Chosen groups list.
 - Select Users in the Chosen groups list and click on the left arrow to remove it.

15.4.3 Update the Weblate full text index

Weblate's full-text index can occasionally get out of sync. When this happens, Weblate's search may fail to find a word that you know exists in the source strings. You can rebuild the index with:

```
$ ssh debian@weblate.securedrop.org
$ cd /app/weblate
$ sudo docker-compose run weblate rebuild_index --all --clean
```

Note that the new index may not be used right away. Some workers may still have the old index open. If the index is holding up translators with a release looming, the server can be rebooted.

DOCUMENTATION GUIDELINES

SecureDrop's documentation is available at <https://docs.securedrop.org>. It is written in [reStructuredText](#) (reST) and hosted by [Freedom of the Press Foundation](#) using a theme by [Read the Docs](#). The documentation files are stored in the `docs/` directory of the [SecureDrop docs repository](#).

16.1 Documentation versions

Note: SecureDrop maintains two documentation versions: `stable` and `latest`. `stable` is the default, and is built from our latest signed git tag. `latest` is built from the head of the `main` git branch of the [securedrop-docs repository](#). In almost all cases involving development work, you'll want to make sure you have the `latest` version selected by using the menu in the bottom left corner of the documentation site.

16.2 Updating Documentation

To get started editing the docs:

1. Clone the SecureDrop documentation repository:

```
git clone https://github.com/freedomofpress/securedrop-docs.git
```

2. Install the dependencies:

Note: You can install, upgrade, and remove Python packages using `pip`, the Python package installer. By default, `pip` will attempt to install packages system-wide. Use `venv` or `virtualenv` to avoid installing Python packages globally and to isolate the installation of dependencies of a specific application.

Run the command `python3 -m venv .venv` to create and manage your virtual environment. To begin using the virtual environment, use the command `source .venv/bin/activate` to activate.

You can also use [virtualenvwrapper](#) which provides the `mkvirtualenv` shell function and is a set of extensions to the `virtualenv` tool. The extensions include wrappers for creating and deleting virtual environments and otherwise managing your development workflow.

```
pip install --require-hashes -r requirements/requirements.txt
```

3. Build the docs for viewing in your web browser:

```
make docs
```

You can then preview the documentation at <http://127.0.0.1:8000>. Navigate to the docs/ directory to make changes to the documentation rendered on <https://docs.securedrop.org>. The documentation pages will automatically rebuild in the browser window, as you make changes; you don't need to refresh the page manually.

After performing lint checks, open a PR against the main branch of the [SecureDrop docs repository](#).

16.3 Testing Documentation Changes

You can check for formatting violations by running the linting option:

```
make docs-lint
```

The `make docs` command will display warnings if mistakes are found, but will still build the documentation. Using `make docs-lint` will convert any warnings to errors, causing the build to fail.

To test the documentation for broken links, run the following command from a reliable internet connection:

```
make docs-linkcheck
```

Project maintainers will need to approve the PR before it can be merged.

Note: It is generally good practice to maintain a clean git history by reducing the number of commits to a reasonable minimum. You can do this by squashing closely related commits through an interactive rebase once your PR is close to being merged. If you are unfamiliar with how to squash commits with rebase, check out this [blog post](#).

If you would like a project maintainer to help you with squashing commits in a PR, please don't hesitate to leave a comment requesting assistance.

16.4 Pushing to a contributor fork

As a maintainer, you can push directly to a contributor fork, as long as there is an active Pull Request corresponding to the branch you are pushing to, and you have added the contributor remote with authentication enabled (i.e. the `url` value in `.git/config` starts with `git@github.com`).

16.5 Updating Screenshots

The user guides for SecureDrop contain screenshots of the web applications. To update these screenshots automatically you can run this command from within your main SecureDrop repository checkout:

```
DOCS_REPO_DIR=/path/to/docs make update-user-guides
```

This will generate screenshots for each page in the web application and copy them to the folder `docs/images/manual/screenshots` in your documentation repository checkout, where they will replace the existing screenshots. Stage for commit any screenshots you wish to update. If you wish to update all screenshots, simply stage for commit all changed files in that directory.

Prior to generating screenshots as part of a release update:

1. Ensure that the version string shown in the screenshots is the version that will be released. You can manually edit `securedrop/version.py` in your SecureDrop repository checkout if it currently contains a release candidate string.
2. Configure the [New York World sample instance](#) logo. Because the functional tests used to generate the logo may themselves update the logo, it is safest to temporarily overwrite the stock logo in `securedrop/static/i/logo.png` in your SecureDrop repository checkout.

Note: The automated screenshots update does not update screenshots for Tails, the Tor Browser UI, the firewall captive portal, etc. If you notice discrepancies in those screenshots, please open issues so they can be addressed at a later point.

16.6 Updating Upgrade Guides

We ship an upgrade guide for each release. As part of updating the documentation for a release:

1. Create a new upgrade guide copied from the most recent one and add it to the index.
2. Ensure that the `latest_upgrade_guide` reference at the top of the document is only present in the latest upgrade guide.
3. If this is a [major-level](#) or [minor-level](#) release, remove the oldest upgrade guide and associated patch-level guides from the documentation.

Example: If you are adding a guide to upgrade from 4.5.0 to 4.6.0, and the oldest guide present is from 4.2.0 to 4.3.0, remove it along with any guides for 4.2.1, 4.2.2, etc.).

4. If this is a major-level or minor-level release, make sure to include the reminders in `docs/includes/backup-and-update-reminders.txt` towards the end of the document.
5. If this release includes a kernel update, make sure to include a reference to the [Kernel Troubleshooting Guide](#).
6. If you are not also the release manager, check with them about any other pertinent release-specific instructions that should be included.
7. Finally, ensure that mentions of the current version are up to date. You can use the `update_version.sh` convenience script to do so.

Example: If you are adding a guide to upgrade to 2.4.2, you can run `./update_version.sh 2.4.2`, then verify that the version changes are pertinent and save them.

16.7 Style Guide

Please see the [reStructuredText](#) Primer by the Sphinx project as a reference for writing in the markup language used for this documentation.

16.7.1 Code Blocks

Ensure that example commands in codeblocks are easy to copy and paste. Do not prepend the `$` shell prompt indicator to example commands:

```
echo hello
```

In the context of a terminal session with both typed commands and printed output text, use `$` before the typed commands:

```
$ echo hello
hello
$ echo sunshine
sunshine
```

16.7.2 Date Format

Follow AP guidelines for formatting dates. Don't use the ISO format for adding dates to the documentation.

To avoid confusion, format dates in the documentation as Month_Name Day, Year:

```
October 13, 2020
```

not

```
13 October, 2020
13/10/2020
10/13/2020
10/13/20
2020-10-13
2020-Oct-13
```

16.7.3 File Paths

Cloning the SecureDrop git repository creates a directory called `securedrop`. This `securedrop` directory also contains a `securedrop` subdirectory for app code.

```
.
├── securedrop
│   │
│   ├── ...
│   └── securedrop
... ..
```

To avoid confusion, paths to files anywhere inside the SecureDrop git repository should be written as `./some_dir/file`, where `.` is the top level directory of the SecureDrop repo.

Use absolute paths when referring to files outside the SecureDrop repository: `/usr/local/bin/tor-browser`.

16.7.4 Glossary

Text taken directly from a user interface is in **bold face**.

“Once you’re sure you have the right drive, click **Format Drive**.”

SecureDrop-specific [glossary](#) is in *italics*.

“To get started, you’ll need two Tails drives: one for the *Admin Workstation* and one for the *Secure Viewing Station*.”

When referring to virtual machines in the development environment, use lowercase for the name:

app-staging VM

16.7.5 Line Wrapping

Lines in the plain-text documentation files should wrap at 80 characters. (Some exceptions: complex code blocks showing example commands, or long URLs.)

16.7.6 Usage and Style

To avoid confusion, lists should include the “Oxford comma”:

“You will need an email address, a public GPG key for that address, and the fingerprint for that key.”

Capitalize all section headings in title case:

```
Before You Begin
=====

Set up the Environment
-----
```

not

```
Before you begin
=====

Set up the environment
-----
```


TESTING SECUREDROP

The SecureDrop project ships both application code for running on servers hosted on-site at news organizations, as well as configuration scripts for provisioning the servers to accept updates to the application code, and to harden the system state. Therefore testing for the project includes *Application Tests* for validating that the app code behaves as expected, and *Configuration Tests* to ensure that the servers are appropriately locked down, and able to accept updates to the app code.

In addition, the *Continuous Integration* automatically runs the above Application and Configuration tests against cloud hosts, to aid in PR review.

TESTING: APPLICATION TESTS

The application test suite uses:

- [Pytest](#)
- [Selenium](#)

The application tests consist of unit tests for the Python application code and functional tests that verify the functionality of the application code from the perspective of the user through a web browser.

The functional tests use an outdated version of Firefox chosen specifically for compatibility with Selenium 2, and a rough approximation of the most recent Tor Browser.

Note: We're working on running the Selenium tests in Tor Browser. See [GitHub #1629](#) for more info.

18.1 Installation

The application tests are installed automatically in the development and app-staging VMs, based on the contents of `securedrop/requirements/test-requirements.txt`. If you wish to change the dependencies, see [Upgrading or Adding Python Dependencies](#).

18.2 Running the Application Tests

The tests can be run inside the development VM:

```
make test
```

Or the app-staging VM:

```
vagrant ssh app-staging
sudo bash
cd /var/www/securedrop
pytest -v tests
chown -R www-data /var/lib/securedrop /var/www/securedrop
```

Warning: The `chown` is necessary because running the tests as root will change ownership of some files, creating problems with the source and journalist interfaces.

For explanation of the difference between these machines, see [Virtual Environments: Servers](#).

If you just want to run the functional tests, you can use:

```
securedrop/bin/dev-shell bin/run-test -v tests/functional
```

Similarly, if you want to run a single test, you can specify it through the file, class, and test name:

```
securedrop/bin/dev-shell bin/run-test \
    tests/test_journalist.py::TestJournalistApp::test_invalid_credentials
```

The `gnupg` library can be quite verbose in its output. The default log level applied to this package is `ERROR` but this can be controlled via the `GNUPG_LOG_LEVEL` environment variable. It can have values such as `INFO` or `DEBUG` if some particular test case or test run needs greater verbosity.

18.2.1 Page Layout Tests

You can check the rendering of the layout of each page in each translated language using the page layout tests. These will generate screenshots of each page and can be used for example to update the SecureDrop user guides when modifications are made to the UI.

You can run all tests, including the page layout tests with the `--page-layout` option:

```
securedrop/bin/dev-shell bin/run-test --page-layout tests
```

18.3 Updating the Application Tests

Unit tests are stored in the `securedrop/tests/` directory and functional tests are stored in the functional test directory:

```
securedrop/tests/
├── functional
│   ├── test_admin_interface.py
│   ├── test_submit_and_retrieve_file.py
│   ├── ...
│   └── submission_not_in_memory.py
├── utils
│   ├── db_helper.py
│   ├── env.py
│   └── asynchronous.py
├── test_journalist.py
├── test_source.py
├── ...
└── test_store.py
```

`securedrop/tests/utils` contains helper functions for writing tests. If you want to add a test, you should see if there is an existing file appropriate for the kind of test, e.g. a new unit testing `manage.py` should go in `test_manage.py`.

TESTING: CONFIGURATION TESTS

`Testinfra` tests verify the end state of the staging VMs. Any changes to the Ansible configuration should have a corresponding spectest.

19.1 Installation

```
pip install --no-deps --require-hashes -r securedrop/requirements/python3/develop-  
requirements.txt
```

19.2 Running the Config Tests

`Testinfra` tests are executed against a virtualized staging environment. To provision the environment and run the tests, run the following commands:

```
make build-debs  
make staging  
make testinfra
```

Test failure against any host will generate a report with informative output about the specific test that triggered the error. Molecule will also exit with a non-zero status code.

19.3 Updating the Config Tests

Changes to the Ansible config should result in failing config tests, but only if an existing task was modified. If you add a new task, make sure to add a corresponding spectest to validate that state after a new provisioning run. Tests import variables from separate YAML files than the Ansible playbooks:

```
molecule/testinfra/staging/vars/  
├── app-prod.yml  
├── app-staging.yml  
├── mon-prod.yml  
├── mon-staging.yml  
└── staging.yml
```

Any variable changes in the Ansible config should have a corresponding entry in these vars files. These vars are dynamically loaded for each host via the `molecule/testinfra/staging/conftest.py` file. Make sure to add your tests to the relevant location for the host you plan to test:

```
molecule/testinfra/staging/app/
├── apache
│   ├── test_apache_journalist_interface.py
│   ├── test_apache_service.py
│   ├── test_apache_source_interface.py
│   └── test_apache_system_config.py
├── test_apparmor.py
├── test_appenv.py
├── test_network.py
└── test_ossec.py
```

In the example above, to add a new test for the app-staging host, add a new file to the `testinfra/staging/app` directory.

Tip: Read [Updating OSSEC Rules](#) to learn how to write tests for the OSSEC rules.

19.4 Config Test Layout

With some exceptions, the config tests are broken up according to platform definitions in the Molecule configuration:

```
molecule/testinfra/staging
├── app
├── app-code
├── common
├── mon
├── ossec
└── vars
```

Ideally the config tests would be broken up according to roles, mirroring the Ansible configuration. Prior to the re-organization of the Ansible layout, the tests are rather tightly coupled to hosts. The layout of config tests is therefore subject to change.

19.5 Config Testing Strategy

The config tests currently emphasize testing implementation rather than functionality. This was a temporary measure to increase the testing baseline for validating the Ansible provisioning flow, which aided in migrating to a current version of Ansible (v2+). Now that the Ansible version is current, the config tests can be improved to validate behavior, such as confirming ports are blocked via external network calls, rather than simply checking that the iptables rules are formatted as expected.

TESTING: CI

The SecureDrop project uses [CircleCI](#) for running automated test suites on code changes.

The relevant files for configuring the CI tests are the `Makefile` in the main repo, the configuration file at `.circleci/config.yml`, and the scripts in `devops/`. You may want to consult the [CircleCI Configuration Reference](#) to interpret the configuration file. Review the `workflows` section of the configuration file to understand which jobs are run by CircleCI.

The files under `devops/` are used to create a libvirt-compatible environment on GCE. The GCE host is used as the Ansible controller, mimicking a developer’s laptop, to provision the machines and run the *tests* against them.

Note: We skip unnecessary jobs, such as the staging run, for pull requests that only affect the documentation; to do so, we check whether the branch name begins with `docs-`. These checks are enforced in different parts of the configuration, mainly within the `Makefile`.

Warning: In CI, we rebase branches in PRs on HEAD of the target branch. This rebase does not occur for branches that are not in PRs. When a branch is pushed to the shared `freedomofpress` remote, CI will run, a rebase will not occur, and since opening a PR *does not trigger a re-build*, the CI build results are not shown rebased on the latest of the target branch. This is important to maintain awareness of if your branch is behind the target branch. Once your branch is in a PR, you can rebuild, push an additional commit, or manually rebase your branch to update the CI results.

20.1 Running the CI Staging Environment

The staging environment tests will run automatically in CircleCI, when changes are submitted by Freedom of the Press Foundation staff (i.e. members of the `freedomofpress` GitHub organization). The tests also perform basic linting and validation, like checking for formatting errors in the Sphinx documentation.

Tip: You will need a Google Cloud Platform account to proceed. See the [Google Cloud Platform Getting Started Guide](#) for detailed instructions.

In addition to a GCP account, you will need a working [Docker installation](#) in order to run the container that builds the deb packages.

You can verify that your Docker installation is working by running `docker run hello-world` and confirming you see “Hello from Docker” in the output as shown below:

```
$ docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.
...
```

20.1.1 Setup Environment Parameters

Source the setup script using the following command:

```
source ./devops/gce-nested/ci-env.sh
```

You will be prompted for the values of the required environment variables. There are some defaults set that you may want to change. You will need to export `GOOGLE_CREDENTIALS` with authentication details for your GCP account, which is outside the scope of this guide.

20.1.2 Use Makefile to Provision Hosts

Run `make help` to see the full list of CI commands in the Makefile:

```
$ make help
Makefile for developing and testing SecureDrop.
Subcommands:
  ci-go                Creates, provisions, tests, and destroys GCE host for
↳testing staging environment.
  ci-lint              Runs linting in linting container.
  ci-teardown          Destroys GCE host for testing staging environment.
```

To run the tests locally:

```
make ci-go
```

You can use `./devops/gce-nested/ci-runner.sh` to provision the remote hosts while making changes, including rebuilding the Debian packages used in the Staging environment. See *Virtual Environments: Servers* for more information.

20.1.3 Debugging CI Issues and Connecting to Remote Instances

For the staging tests, a container will be spawned on CircleCI, which will then create a Google Compute instance with nested virtualization and will set up the virtual environment and run the playbooks on that remote.

Cloud instances are deleted after the test run is completed, whether a test run passes or fails. In order to debug the state of the remote instance, we must first ensure that the instance is not automatically destroyed. Note that there is also a cron job that destroys instances daily as well. The following is an example of a commit to apply to a branch in order to disable the deletion for the Focal staging job:

```
diff --git a/.circleci/config.yml b/.circleci/config.yml
index 4d61769f1..af74672bc 100644
--- a/.circleci/config.yml
+++ b/.circleci/config.yml
@@ -251,13 +251,6 @@ jobs:
```

(continues on next page)

(continued from previous page)

```

    make ci-go
    no_output_timeout: 35m

-   - run:
-       name: Ensure environment torn down
-       # Always report true, since env should will destroyed already
-       # if all tests passed.
-       command: make ci-teardown || true
-       when: always
-
-   - store_test_results:
-       path: ~/sd/junit

diff --git a/devops/gce-nested/ci-go.sh b/devops/gce-nested/ci-go.sh
index ff80aa107..65bbcd7b9 100755
--- a/devops/gce-nested/ci-go.sh
+++ b/devops/gce-nested/ci-go.sh
@@ -16,4 +16,3 @@ export BASE_OS="${BASE_OS:-focal}"

./devops/gce-nested/gce-start.sh
./devops/gce-nested/gce-runner.sh
-./devops/gce-nested/gce-stop.sh
diff --git a/devops/scripts/create-staging-env b/devops/scripts/create-staging-env
index 8b296be94..df8a4d674 100755
--- a/devops/scripts/create-staging-env
+++ b/devops/scripts/create-staging-env
@@ -32,7 +32,7 @@ printf "Creating staging environment via '%s'...\n" "${securedrop_
↪ staging_scena
virtualenv_bootstrap
# Are we in CI? Then lets do full testing post install!
if [ "$USER" = "sdci" ]; then
-   molecule test -s "${securedrop_staging_scenario}"
+   molecule test --destroy=never -s "${securedrop_staging_scenario}"
else
    molecule "${MOLECULE_ACTION:-converge}" -s "${securedrop_staging_scenario}" "$
↪ {EXTRA_ANSIBLE_ARGS[@]}"
fi

```

Once that commit is pushed, run the staging-test-with-rebase job with ssh using with CircleCI. Once logged into that container, you can ssh into the Google Compute host:

```
ssh -i /tmp/gce-nested/gce sdci@<ip adress>
```

Once on the GCP host, the SecureDrop source is in `/home/sdci/securedrop-source` and you may activate the virtualenv, list the molecule instances and connect to VM instances:

```
cd securedrop-source
source .venv/bin/activate
molecule list
molecule login -s libvirt-staging-focal --host app-staging

```


SECUREDROP APT REPOSITORY

This document contains brief descriptions of the Debian packages hosted and maintained by Freedom of the Press Foundation in our apt repository (apt.freedom.press).

linux-image-*-grsec

This package contains the Linux kernel image, patched with grsecurity. Listed as a dependency of `securedrop-grsec`.

ossec-agent

Installs the OSSEC agent, repackaged for Ubuntu. Listed as a dependency of `securedrop-ossec-agent`.

ossec-server

Installs the OSSEC manager, repackaged for Ubuntu. Listed as a dependency of `securedrop-ossec-server`.

securedrop-app-code

Packages the SecureDrop application code, Python pip dependencies and AppArmor profiles.

securedrop-ossec-agent

Installs the SecureDrop-specific OSSEC configuration for the *Application Server*.

securedrop-ossec-server

Installs the SecureDrop-specific OSSEC configuration for the *Monitor Server*.

securedrop-grsec

SecureDrop grsecurity kernel metapackage, depending on the latest version of `linux-image-3.14-*-grsec`.

securedrop-keyring

Packages the public signing key for this apt repository. Allows for managed key rotation via automatic updates, as implemented in [SecureDrop 0.3.10](#).

Note: The SecureDrop install process configures a custom Linux kernel hardened with the grsecurity patch set. Only binary images are hosted in the apt repo. For source packages, see the [Source Offer](#).

UPDATING OSSEC RULES

SecureDrop uses the OSSEC open source host-based intrusion detection system (IDS) for log analysis, file integrity checking, policy monitoring, rootkit detection and real-time alerting. Refer to our [OSSEC guide](#) to learn more about how SecureDrop admins set up and monitor OSSEC alerts.

22.1 Alerting Strategy

The goals of the OSSEC alerts in SecureDrop is to notify admins of:

1. Suspicious security events
2. Changes that require some kind of admin action
3. Other important notifications regarding system state.

If an alert is purely informational and there is no realistic action an admin is expected to take, you should think carefully before suggesting a rule for it. Each additional alert that admins must read and/or respond to takes time. Alerts that are unimportant or otherwise require no action can lead to alert fatigue and thus to critical alerts being ignored.

22.2 Using ossec-logtest

Development on the OSSEC rules should be done from the staging environment.

On `mon-staging`, there is a utility installed as part of OSSEC called `ossec-logtest` that you can use to test log events. In order to evaluate whether an alert will be produced, and if so, what rule triggered it and its level, you can simply pass the event to `ossec-logtest`:

```
root@mon-staging:/home/vagrant# sudo echo "Feb 10 23:34:40 app-prod kernel: [ 124.
→188641] grsec: denied RWX mmap of <anonymous mapping> by /usr/sbin/
→apache2[apache2:1328] uid/euid:33/33 gid/egid:33/33, parent /usr/sbin/
→apache2[apache2:1309] uid/euid:0/0 gid/egid:0/0" | /var/ossec/bin/ossec-logtest
2017/08/16 22:28:25 ossec-testrule: INFO: Reading local decoder file.
2017/08/16 22:28:25 ossec-testrule: INFO: Started (pid: 18973).
ossec-testrule: Type one log per line.

**Phase 1: Completed pre-decoding.
    full event: 'Feb 10 23:34:40 app-prod kernel: [ 124.188641] grsec: denied RWX
→mmap of <anonymous mapping> by /usr/sbin/apache2[apache2:1328] uid/euid:33/33 gid/
→egid:33/33, parent /usr/sbin/apache2[apache2:1309] uid/euid:0/0 gid/egid:0/0'
    hostname: 'app-prod'
    program_name: 'kernel'
```

(continues on next page)

(continued from previous page)

```
log: '[ 124.188641] grsec: denied RWX mmap of <anonymous mapping> by /usr/sbin/
↳ apache2[apache2:1328] uid/euid:33/33 gid/egid:33/33, parent /usr/sbin/
↳ apache2[apache2:1309] uid/euid:0/0 gid/egid:0/0'

**Phase 2: Completed decoding.
    decoder: 'iptables'

**Phase 3: Completed filtering (rules).
    Rule id: '100101'
    Level: '7'
    Description: 'grsec error was detected'

**Alert to be generated.
```

This is the utility we use in automated tests of OSSEC.

22.3 Writing Automated Tests for OSSEC Rules

We strongly recommend before making changes to OSSEC rules to attempt to write a failing test which you then can make pass with a patch to the OSSEC rules:

1. Identify a log event you can use to trigger the alert.

Warning: Be sure to use only log events from test SecureDrop instances or those you have verified do not contain any sensitive data.

2. Write a `Testinfra` test to verify that the log event does or does not trigger an alert.
3. Apply your patch to the OSSEC rule on the relevant VM (likely `app`).
4. Restart the service via `sudo service ossec restart` on `mon`.

Note: Currently we only have automated tests for alerts triggered due to log events (for example not for `syscheck`, OSSEC's integrity checking process). If you have ideas for additional automated test coverage of alerts, please suggest them in ticket 2134 on GitHub.

22.4 Adding new OSSEC rules

OSSEC processes events in two steps:

1. **Decoders** parse and filter log events that meet certain criteria for subsequent processing. SecureDrop's custom rules are defined in `install_files/securedrop-ossec-server/var/ossec/rules/local_rules.xml`.
2. **Rules** check decoded events against conditions and optionally yield alerts. SecureDrop's custom rules are defined in `install_files/securedrop-ossec-server/var/ossec/etc/local_decoder.xml`.

A basic decoder filters log events by `program_name` (e.g., `fwupd`). If a decoder is already defined for the program of interest, you can go straight to *defining a new rule* unless you have a reason to add additional *decoders* for further filtering.

22.4.1 The decoder file

For example, to add a decoder for log events from fwupd, you can add to `local_decoder.xml`:

```
<!--
  The default fwupd tries to auto-update and generates error.
-->
<decoder name="fwupd">
  <program_name>fwupd</program_name>
</decoder>
```

You can find this `program_name` value using the *“ossec-logtest”* command. Copy-paste the log event as input to this command, and it will give you some parsed output:

```
$ echo "Mar  1 13:22:53 app fwupd[133921]: 13:22:53:0883 FuPluginUefi      Error
↳opening directory 'â€œ/sys/firmware/efi/esrt/entriesâ€œ: No such file or directory" |
↳sudo /var/ossec/bin/ossec-logtest
[...]
**Phase 1: Completed pre-decoding.
  full event: 'Mar  1 13:22:53 app fwupd[133921]: 13:22:53:0883 FuPluginUefi
↳Error opening directory 'â€œ/sys/firmware/efi/esrt/entriesâ€œ: No such file or directory
↳'
  hostname: 'app'
  program_name: 'fwupd'
  log: '13:22:53:0883 FuPluginUefi      Error opening directory 'â€œ/sys/firmware/
↳efi/esrt/entriesâ€œ: No such file or directory'

**Phase 2: Completed decoding.
  No decoder matched.

**Phase 3: Completed filtering (rules).
  Rule id: '1002'
  Level: '2'
  Description: 'Unknown problem somewhere in the system.'
**Alert to be generated.
```

22.4.2 The rules

Next, you can add one or more rules corresponding to the new decoder, making sure that the rules have proper unique *id* numbers and are written in the correct (sorted) place in the `local_rules.xml` file.

```
<group name="fwupd">
<rule id="100111" level="0">
  <decoded_as>fwupd</decoded_as>
  <match>Error opening directory</match>
  <description>fwupd error</description>
  <options>no_email_alert</options>
</rule>
<rule id="100112" level="0">
  <decoded_as>fwupd</decoded_as>
  <match>Failed to load SMBIOS</match>
  <description>fwupd error for auto updates</description>
```

(continues on next page)

(continued from previous page)

```
<options>no_email_alert</options>
</rule>
</group>
```

22.4.3 Verify the new OSSEC rule

On the monitor server you can use the following command as *root* to verify the new rule:

```
/var/ossec/bin/ossec-analysisd -t
```

`ossec-analysisd` will receive log messages and compare them to our rules, including the new rule you just added. Then it creates alerts when a log message matches an applicable rule.

22.4.4 Adding an automated test for staging

You can then add tests in the `molecule/testinfra/mon/test_ossec_ruleset.py` file. Here the test loops over the entries in the `log_events_with_ossec_alerts` and `log_events_without_ossec_alerts` variables in `molecule/testinfra/vars/staging.yml` and makes sure that the `rule_id` and `level` match. See [Writing Automated Tests for OSSEC Rules](#) for details.

22.5 Deployment

The OSSEC rules and associated configuration files are distributed via Debian packages maintained by Freedom of the Press Foundation. Any changes made to OSSEC configuration files will land on production SecureDrop monitoring servers as part of each SecureDrop release. This upgrade will occur automatically.

Note: The use of automatic upgrades for release deployment means that any changes made locally by admins to their OSSEC rules will not persist after a SecureDrop update.

GENERATING APPARMOR PROFILES FOR TOR AND APACHE

```
make staging
molecule login -s libvirt-staging-focal -h app-staging
sudo su
cd /var/www/securedrop
```

Run tests, use the application web interface, restart services, reboot the VMs via `vagrant reload /staging/`. The goal is to create as much interaction with the system as possible, in order to establish an expected baseline of behavior. Then run:

```
aa-logprof
```

Follow the prompts on screen and save the new configuration. Then set the profile to complain mode:

```
aa-complain /etc/apparmor.d/<PROFILE_NAME>
```

Rinse and repeat, again running `aa-logprof` to update the profile. The AppArmor profiles are saved in `/etc/apparmor.d/`. There are two profiles:

- `/etc/apparmor.d/usr.sbin.tor`
- `/etc/apparmor.d/usr.sbin.apache2`

After running `aa-logprof` you will need to copy the modified profile back to your host machine to include them in the `securedrop-app-code` package.

```
ansible -i .vagrant/provisioners/ansible/inventory/vagrant_ubuntu_inventory app-prod -m
↪ fetch -a 'flat=yes dest=install_files/ansible-base/ src=/etc/apparmor.d/usr.sbin.
↪ apache2'
ansible -i .vagrant/provisioners/ansible/inventory/vagrant_ubuntu_inventory app-prod -m
↪ fetch -a 'flat=yes dest=install_files/ansible-base/ src=/etc/apparmor.d/usr.sbin.tor'
```

The AppArmor profiles are packaged with the `securedrop-app-code`. The `securedrop-app-code` postinst puts the AppArmor profiles in enforce mode on production and staging hosts.

PORTABLE SECUREDROP DEMO

When at a conference or traveling, it is possible to prepare a SecureDrop demo using portable hardware and adapted usage scenarios.

24.1 Hardware

- A laptop running the *staging virtual environment*
- A Tails compatible laptop with a physical radio kill switch (for instance a Lenovo T420)
- Four USB keys prepared for the staging environment running on the laptop
 - Transfer
 - Journalist
 - SVS
 - Admin

The Tails compatible laptop has the physical radio kill switch turned off to simulate a SVS and it is rebooted with the physical radio kill switch turned on to simulate the Admin or Journalist workstation.

RELEASE MANAGEMENT

The **Release Manager** is responsible for shepherding the release process to successful completion. This document describes their responsibilities. Some items must be done by people that have special privileges to do specific tasks (e.g. privileges to access the production apt server), but even if the **Release Manager** does not have those privileges, they should coordinate with the person that does to make sure the task is completed.

In addition to the Release Manager, we typically recognize the following roles for a SecureDrop release:

- **Deputy RM:** for additional time zone coverage, to delegate specific tasks, and to act as backup in case of the RM becomes unavailable for any reason.
- **Localization Manager:** to manage outreach to the translator community, and to coordinate translation updates of existing strings.
- **Deputy LM:** like the RM, this role is backed up by another team member.
- **Communications Manager:** to prepare and distribute pre-release and release messaging (including standard upgrade instructions, release notes, social media posts, and support portal announcements)

During the full release cycle, we also recognize the following role:

- **Community Manager:** to engage with community contributors, offer initial responses to new issues and Pull Requests, and follow up with other SecureDrop team members as appropriate.

We aim to rotate membership in these roles regularly.

25.1 Pre-Release

1. Open a **Release SecureDrop <major>.<minor>.<patch>** issue to track release-related activity. Keep this issue updated as you proceed through the release process for transparency.
2. Copy a link of the latest release or release candidate from the [Tails apt repo](#) and include it in the issue. You can compare it with the [Tails release calendar](#) if you're not sure. The goal is to make sure we test against the latest Tails release, including release candidates, so that we can report bugs early to Tails.
3. Create a release branch.

For a regular release, create a release branch off of `develop`:

```
git checkout develop
git checkout -b release/<major>.<minor>.0
```

For a point release, create a release branch off of the latest merged release branch:

```
git checkout release/<major>.<minor>.0
git checkout -b release/<major>.<minor>.1
```

4. For each release candidate, update the version files, code repo changelog, and Debian package changelog.
 - a. First collect a list of changes since the last release. For example, if the last release was version 1.6.0, you can view changes in Github by running:

```
https://github.com/freedomofpress/securedrop/compare/release/1.6.0...develop
```

Also check [SecureDrop milestones](#) to make sure all milestone changes are included. Append GitHub PR numbers to each change. You will add these changes to the changelog in the next step.

- b. Run `update_version.sh` which will walk you through updating the version files and changelogs. When you run the script, pass it the new version in the format `<major>.<minor>.<patch>~rcN`:

```
./update_version.sh <major>.<minor>.<patch>~rcN
```

Note: A tilde is used in the version number passed to `update_version.sh` to match the format specified in the [Debian docs](#) on how to name and version a package, whereas a dash is used in the tag version number since `git` does not support the use of tilde.

Note: In the Debian changelog, we typically just refer the reader to the `changelog.md` file.

- c. Disregard the script-generated `.tag` file since this is only used when we need to sign the final release tag (see [Release Process](#) section).
 - d. Sign the commit that was added by `update_version.sh`:

```
git commit --amend --gpg-sign
```

- e. Push the branch:

```
git push origin release/<major>.<minor>.<patch>
```

- f. Push the unsigned tag (only the final release tag needs to be signed, see [Release Process](#) section):

```
git push origin <major>.<minor>.<patch>-rcN
```

- g. Once the tag is pushed, notify the Localization Manager so that the localization team can get started on translations.

5. Build Debian packages:

- a. Check out the tag for the release candidate.
 - b. Build the packages with `make build-debs`

Note: If the [build container](#) used by `make build-debs` has security updates, then you will see `test_ensure_no_updates_avail` fail in the build output. To get around the bottleneck of tight restrictions around who can update the build container, you can ignore this test failure until you are building a production release.

- c. Build logs should be saved and published according to the [build log guidelines](#).
 - d. Open a PR on [securedrop-dev-packages-lfs](#) that targets the `main` branch with the new debs. Do not include tarballs or any debs that would overwrite existing debs. Changes merged to this branch will be published to `apt-test.freedom.press` within 15 minutes.

Warning: Only commit deb packages with an incremented version number: do not clobber existing packages. That is, if there is already a deb called e.g. `ossec-agent-3.6.0-amd64.deb` in `main`, do not commit a new version of this deb.

Note: If the release contains other packages not created by `make build-debs`, such as Tor or kernel updates, make sure that they also get pushed to `apt-test.freedom.press`.

6. Write a test plan that focuses on the new functionality introduced in the release. Post for feedback and make changes based on suggestions from the community. Once it's ready, publish the test plan in the [wiki](#) and link to it in the **Release SecureDrop <major>.<minor>.<patch>** issue.
7. Create a new QA matrix spreadsheet by copying the google spreadsheet from the last release and adding a new row for testing new functionality specific to the release candidate. Link to this in the **Release SecureDrop <major>.<minor>.<patch>** issue.
8. At this point, QA can begin. During the QA period:
 - Encourage QA participants to QA the release on production VMs and hardware. They should post their QA reports in the release issue such that it is clear what was and what was not tested. It is the responsibility of the release manager to ensure that sufficient QA is done on the release candidate prior to final release.
 - Triage bugs as they are reported. If a bug must be fixed before the release, it's the release manager's responsibility to either fix it or find someone who can.
 - You may, at your discretion, escalate a "release blocker" to "coordinated response" status. In this case, you (or the person you designate, such as the issue's reporter) should coordinate an incident-response-style investigation and resolution of the bug, using tools like Etherpad and Google Docs/Sheets to consolidate information in real time and convening short sync-up meetings as often as needed. After a coordinated response, make sure that the findings gathered in these venues are reported back out publicly (i.e., in the original GitHub issues) for transparency and for future reference.
 - Backport release QA fixes merged into `develop` into the release branch using `git cherry-pick -x <commit>` to clearly indicate where the commit originated from.
 - At your discretion – for example when a significant fix is merged – prepare additional release candidates and have fresh Debian packages prepared for testing.
 - For a regular release, the string freeze will be declared by the translation administrator one week prior to the release. After this is done, ensure that no changes involving string changes are backported into the release branch.
 - Work with the Communications Manager assigned for the release to prepare a pre-release announcement that will be shared on the `support.freedom.press` support portal, `securedrop.org` website, and Twitter. Wait until the day of the release before including an announcement for a SecureDrop security update. For a point release, you may be able to skip the pre-release announcement depending on how small the point release is.

Make sure a draft of the release notes are prepared and shared for review, and that a draft PR is prepared into the `securedrop-docs` repository which:

- bumps the SecureDrop version of the documentation using the `update_version.sh` script in that repository;
- adds *upgrade instructions and other release-specific technical documentation*;
- *updates the screenshots*.

25.2 Release Process

1. If this is a regular release, work with the translation administrator responsible for this release cycle to review and merge the final translations and screenshots (if necessary) they prepare. Refer to the [i18n documentation](#) for more information about the i18n release process. Note that you *must* manually inspect each line in the diff to ensure no malicious content is introduced.
2. Prepare the final release commit and tag. Do not push the tag file.
3. Step through the signing ceremony for the tag file. If you do not have permissions to do so, coordinate with someone that does.
4. Once the tag is signed, append the detached signature to the unsigned tag:

```
cat 1.x.y.tag.sig >> 1.x.y.tag
```

5. Delete the original unsigned tag:

```
git tag -d 1.x.y
```

6. Make the signed tag:

```
git mktag < 1.x.y.tag > .git/refs/tags/1.x.y
```

7. Verify the signed tag:

```
git tag -v 1.x.y
```

8. Push the signed tag:

```
git push origin 1.x.y
```

9. Ensure there are no local changes (whether tracked, untracked or git ignored) prior to building the debs. If you did not freshly clone the repository, you can use git clean:

Dry run (it will list the files/folders that will be deleted):

```
git clean -ndfx
```

Actually delete the files:

```
git clean -dfx
```

10. Build Debian packages:
 - a. Verify and check out the signed tag for the release.
 - b. Build the packages with `make build-debs`.
 - c. Build logs should be saved and published according to the [build log guidelines](#).
11. In a clone of the private `securedrop-debian-packages-lfs` repository, create a branch from main called `release`.
12. In your local branch, commit the built packages to the `core/focal` directory.
 - If the release includes a kernel update, make sure to add the corresponding grsecurity-patched kernel packages, including both `linux-image-*` and `linux-firmware-image-*` packages as appropriate.
13. Run the `tools/publish` script. This will create the Release file.
14. Commit the changes made by the `tools/publish` script.

15. Push your commits to the remote `release` branch. This will trigger an automatic upload of the packages to `apt-qa.freedom.press`, but the packages will not yet be installable.
16. Create a [draft PR](#) from `release` into `main`. Make sure to include a link to the build logs in the PR description.
17. A reviewer must verify the build logs, obtain and sign the generated Release file, and append the detached signature to the PR. The PR should remain in draft mode. The packages on `apt-qa.freedom.press` are now installable.
18. Coordinate with one or more team members to [confirm a successful clean install in production VMs](#) using the packages on `apt-qa.freedom.press`.
19. If no issues are discovered in final QA, promote the packaging PR out of draft mode.
20. A reviewer must merge the packaging PR. This will publish the packages on `apt.freedom.press`.
21. The reviewer must delete the `release` branch so that it can be re-created during the next release.
22. Update the [public documentation](#):
 - Review and merge the `securedrop-docs` PR that bumps the version and adds the upgrade documentation for this release.
 - Verify that there are no changes on the `main` branch of `securedrop-docs` that should not be released into the stable version of the documentation.

If necessary, you can create a branch from an earlier commit. Follow the `release/<major>.<minor>.<patch>` convention for the branch name in `securedrop-docs`, and cherry-pick at least the changes from the PR above onto it via a backport PR.
 - Create a tag signed with your developer key in the format `<major>.<minor>.<patch>` on the HEAD of the `main` branch or of the docs release branch you created in the previous step.

```
git tag -as <major>.<minor>.<patch>
git push origin <major>.<minor>.<patch>
```

This will update the stable version of the documentation.

- Subsequent changes to the stable version should be tagged with PEP-440 conformant [post-release separators](#) in the format `<major>.<minor>.<patch>-1`, `<major>.<minor>.<patch>-2`, and so on.
1. Verify that the public documentation has been updated. Inspecting or restarting builds requires Codefresh access; if you lack access, a tech lead or infra team member can do so on your behalf.
 2. Create a [release](#) on GitHub with a brief summary of the changes in this release.
 3. Make sure that release notes are written and posted on the SecureDrop blog.
 4. Make sure that the release is announced from the SecureDrop Twitter account.
 5. Make sure that members of [the support portal](#) are notified about the release.
 6. Make sure that version string monitored by FPF's Icinga monitoring system is updated by the infrastructure team.

25.3 Post-Release

1. Backport the changelog from the release branch into `develop`.
 - a. Collect the hashes of all the commits that modified `changelog.md` during the release:

```
git log --pretty=oneline changelog.md
```
 - b. From a new branch based on `develop`, cherry-pick each commit in the `git log` output from the previous step. Make sure to use the `-x` flag so that the original commit is appended to the new commit.
2. Bump the SecureDrop version so that it's ready for the next release.
 - a. Create a new minor release candidate. Only add a commit message and accept the default changes for everything else (it's fine to leave the changelog entries with empty bullets). For example, if the release is 1.3.0, then you'll run:

```
./update_version.sh 1.4.0~rc1
```
 - b. Disregard the script-generated `.tag` file since this is only used when we are making an actual release.
 - c. Sign the commit that was added by `update_version.sh`:

```
git commit --amend --gpg-sign
```
 - d. Make a PR to merge these changes into `develop`.
3. Monitor the [FPF support portal](#) and the [SecureDrop community support forum](#) for any new user issues related to the release.

BUILD CONTAINER

We use a Docker build container to build our debian packages for SecureDrop (via `make build-debs` in the `securedrop` Github repository root directory). We keep images of this our container in a Docker repository at <https://quay.io/freedomofpress>. The images are organized by Ubuntu release version. For instance, you can find the images for Focal at <https://quay.io/freedomofpress/sd-docker-builder-focal>.

Maintaining images of our build container for each release is our way of recording the exact version of each dependency used to build our production debian packages for SecureDrop.

26.1 Who can update the build container?

There are tight restrictions over who can make edits to our Docker repository. If you have permissions to do so, you'll need to make sure your local Docker client has credentials to push.

- First login into your quay.io account via the web-portal at <https://quay.io/>
- Drill into your **Account settings** via the upper right drop-down (where your username is)
- Click **Generate Encrypted Password**
- From a command-line prompt type **docker login quay.io** with your username and credentials obtained from the previous step.
- Proceed with update instructions

26.2 Updating the build container

We know the build container needs to be updated when `test_ensure_no_updates_avail` fails during `make build-debs` in the `securedrop` Github repository root directory. This test fails if any of the dependencies required to build the debian packages have security updates. If you have access rights to push to quay.io, then you can build and push a new container to the Quay repository by following the steps below.

Note: The reason we don't update the container at runtime is that we use the container image as a way of recording our build environment.

```
cd molecule/builder/  
# Build a new container  
make build-container
```

Once the container is built, you can push the container to the registry.

```
make push-container
```

You can now test the container by going back to the SecureDrop repository root:

```
cd ../../..
make build-debs
```

Assuming no errors here, commit the changes in `molecule/builder/image_hash` in a branch containing the prefix `update-builder-`.

UPDATING TOR

Given SecureDrop's significant reliance on Tor via Onion Services, we test new Tor versions to ensure they don't break SecureDrop before releasing them to users.

27.1 Identifying new releases

Announcements for new Tor releases are posted in the [Tor forum](#).

Our continuous integration automatically checks for new Tor packages every night and should commit them to the [securedrop-dev-packages-lfs](#) repository. Within 15 minutes they should be available for download via `apt-test.freedom.press`.

27.2 Testing

Use a staging environment to verify that with the new Tor release, SecureDrop functions properly as an Onion Service, both the Source Interface and protected Journalist Interface.

Then install the new Tor release on a production environment. Wait a day so it goes through the unattended-upgrades cycle, confirming that after the nightly reboot, Tor is still on the new version and running as expected.

27.3 Promoting

To promote a Tor release to production, copy the `*.deb` files over to the [securedrop-debian-packages-lfs](#) repository and follow those instructions.

DEVELOPING THE SECUREDROP CLIENT APPLICATION

As part of the ongoing work to make an integrated journalist-friendly workstation for SecureDrop we have created a native client application to be run within the Qubes operating system. It helps journalists with the most common activities associated with using SecureDrop in a user friendly manner.

Currently the client is alpha quality although work is ongoing in terms of improving features and the user interface.

The source code, and related issues are [hosted on GitHub](#).

28.1 Developer Setup

You may find developer setup instructions in the [SecureDrop Client README](#).

28.2 How to Find Help

If you would like to report a problem [submit a new issue](#).

If you'd like to chat with other developers working on the client drop into our [Gitter chat channel for the project](#).

Every non-public holiday weekday (except Fridays) at 10am (Pacific Time) we take part in a public daily stand-up, usually via a [meeting on Google Meet](#) (although the details of each daily meeting are published on the Gitter channel five minutes before the start of the meeting). All are welcome to contribute.

Otherwise, read on.

28.3 Client Architecture

The SecureDrop client is a PyQt application. It's written using Python 3.5 and the Python bindings for the Qt UI framework ([PyQt](#)).

In the root directory of the repository are two important directories: `securedrop_client` (containing the application code) and `tests` containing our unit tests. You'll also find a `Makefile` in the root directory which defines commands to run commonly needed activities. Type, `make` to find out what commands are available.

The code in the `securedrop_client` namespace is organised in the following way:

- `app.py` - starts and configures the application.
- `logic.py` - contains the application logic, encapsulated in the `Client` class.
- `db.py` - holds all the [SQLAlchemy ORM model definitions](#) for interacting with the local SQLite database.

- `storage.py` - contains the functions needed for interacting with a remote SecureDrop API and the local database.
- `utils.py` - generic utility functions needed throughout the application.
- `gui` - this namespace contains two modules: `main.py` (containing the `Window` class through which all interactions with the user interface should happen) and `widgets.py` (containing all the custom widgets used by the `Window` class to draw the user interface).

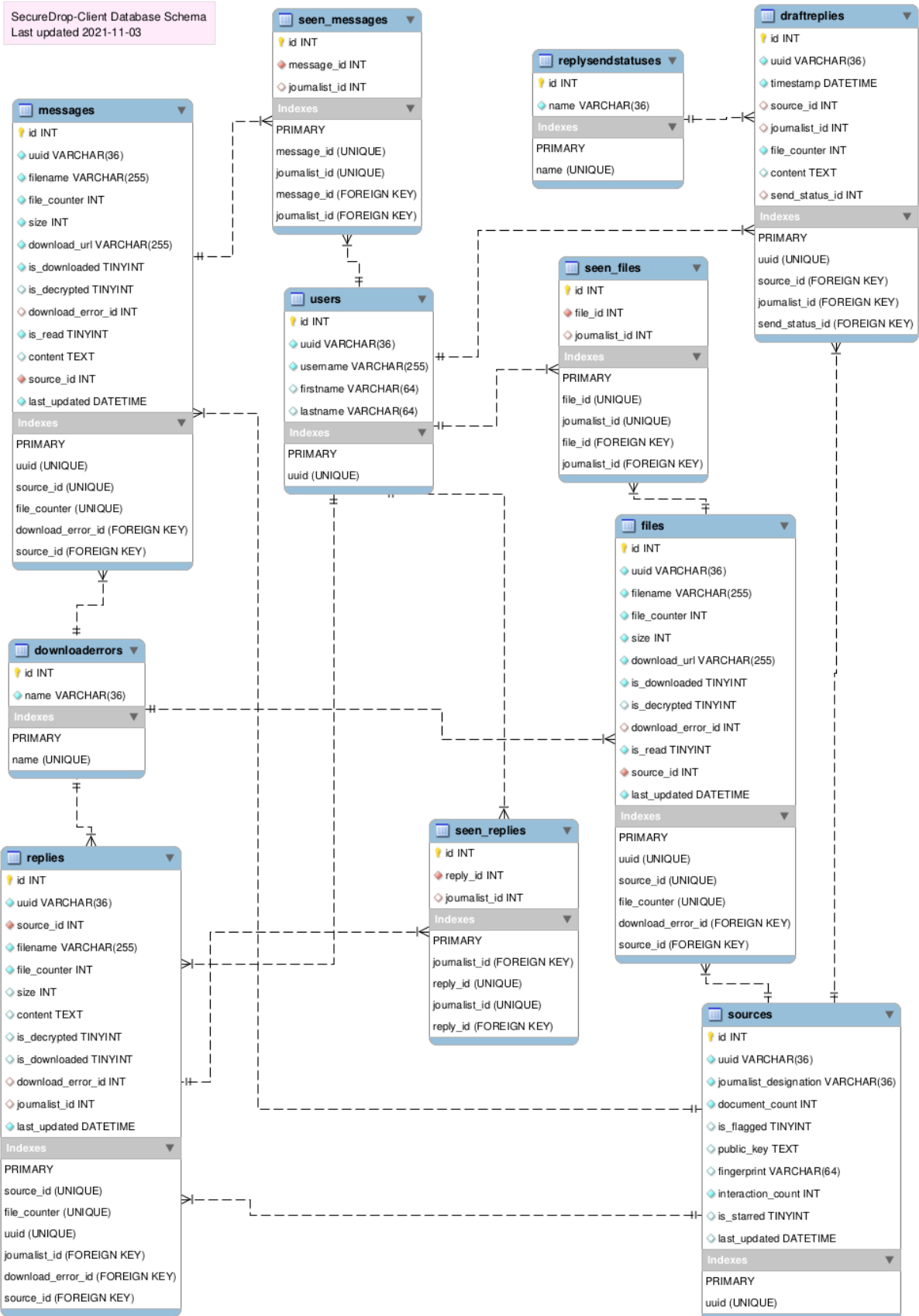
We try very hard to keep the application logic and UI code cleanly separated. Furthermore, we try equally hard to ensure the main GUI code always remains unblocked. For instance look at how the `APICallRunner` is used in `logic.py` to make unblocked network calls to the remote API.

We encourage developers to make sure all classes, methods and functions have docstrings describing the intention behind the code. Obviously, it's important that such docstrings **remain up to date** as the code evolves.

If possible, please use [Python type hints](#) for new code. We're going to transition the code base to this style in the not-too-distant future.

28.4 Client Database Structure

For a better understanding of the SecureDrop Client application architecture, a high-level view of its database structure has been provided:



28.5 Tests

The files and directory structure found within the `tests` directory mirrors that of the files and directories in `securedrop_client`. For instance, all the unit tests for the `securedrop_client/logic.py` module can be found in the `tests/test_logic.py` file.

To run the complete test suite simply type:

```
make check
```

Our code style checkers, full test suite and coverage checker will run and report any errors.

We use the [PyTest testing framework](#) for writing and running our unit tests. We expect every test to have an associated comment which describes the *intent* of the test. As far as possible, tests should be self contained with all the context needed to understand them within each individual unit test (this makes it easier to debug things when the test suite fails as the codebase evolves).

Take a look in any of the test files to see the sort of code we expect for unit tests.

We currently have, and expect to maintain, 100% unit test coverage of our code base. If you're unsure how to achieve this, please don't hesitate to get in touch via Gitter or mention this in your description of any pull requests you submit.

28.6 Contributing

Our open issues are [on GitHub](#).

Please remember that we have a [Code of Conduct](#) and expect all contributors to abide by it.

Before submitting a pull request, make sure the test suite passes (`make check`), because our CI tools will flag broken tests before we're able to merge your code into `main`.

Most of all, please don't hesitate to get in touch if you need help, advice or would like guidance.

Thank you for your support!