
SecureDrop Developer Documentation

Release latest

SecureDrop Team and Contributors

Aug 27, 2026

GETTING STARTED

1	Contributing to SecureDrop	3
1.1	Programmers	3
1.2	Technical Writers	4
1.3	UX Contributors	5
1.4	Translators	5
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	Weblate	18
4.3	How-To Guides	24
5	Contributing Guidelines	27
5.1	Signing Commits	27
5.2	Code Review	27
5.3	Branching Strategy	27
5.4	Automated Testing	27
5.5	Code Style	28
5.6	Type Hints in Python code	29
5.7	Git History	30
5.8	Privileges	30
5.9	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	Policy on Supported Languages	37
7.1	Definitions	37
7.2	Thresholds for Translation and Review Coverage	37
7.3	Granting Support for a Language	37

7.4	Revoking Support for a Language	38
7.5	Adding a New Language for Translation	38
7.6	Technical Limitations	38
8	Continuous Integration	41
8.1	Basics	41
8.2	Pull requests	41
8.3	Special branch prefixes	41
8.4	Nightlies	41
8.5	Workstation CI	42
9	Dependency specification and update policies	43
9.1	Adding a dependency	43
9.2	Updating dependencies	44
9.3	Specifying version constraints	45
9.4	Scanning updated dependencies with GuardDog	45
9.5	Scanning updated dependencies with Semgrep	46
9.6	Manual dependency diff review procedure	47
9.7	Auditing Rust dependencies	48
10	Reproducible builds	51
10.1	Goals	51
10.2	Build environment	51
10.3	Build metadata	51
10.4	Reproducible wheels	51
10.5	Not reproducible	52
10.6	Tools	52
11	End-of-Life Maintenance Guidelines	53
11.1	Overview	53
11.2	EOL Tracking and Monitoring	53
11.3	SecureDrop Server Components	54
11.4	SecureDrop Workstation Components	54
11.5	Communication Protocol	55
12	Language toolchains	57
12.1	Python	57
12.2	Rust	57
12.3	Node.js	58
13	Journalist Interface API	59
13.1	Versioning	59
13.2	Content Type	59
13.3	Authentication	59
13.4	Errors	60
13.5	Endpoints	60
13.6	Removed functionality	72
14	Development of Securedrop-Admin in the Admin Directory	73
15	Development of SecureDropUpdater in the journalist_gui Directory	75
15.1	Installing the Dependencies in a Virtual Environment	75
15.2	To Update the UI Design	75
15.3	Using Resources in the UI	76
15.4	Adding and Running Test Cases	76

16 Virtual Environments: Servers	77
16.1 Staging	77
16.2 Production	79
17 Virtual Environments: Admin Workstation	83
17.1 Linux	83
18 Virtual Environments: Using Qubes	85
18.1 Overview	85
18.2 Download Ubuntu server ISO	85
18.3 Create the base VM	85
18.4 Boot into installation media	86
18.5 Initial VM configuration	86
18.6 Clone VMs	87
18.7 SecureDrop Installation	88
18.8 Managing Qubes RPC for Admin API capability	88
18.9 Creating staging instance	89
18.10 Accessing the Journalist Interface (Staging) in Whonix-based VMs	89
19 Upgrade Testing using Molecule	91
19.1 Upgrade testing using locally-built packages	91
19.2 Upgrade testing using apt-test.freedom.press	92
20 Database Migrations	93
20.1 Migration Files	93
20.2 Deployment	93
20.3 Developer Workflow	93
21 Internationalization (i18n)	97
21.1 What languages are available where?	98
21.2 Development tasks	98
21.3 Release Management	100
21.4 Weblate administration	101
22 Documentation Guidelines	103
22.1 Documentation versions	103
22.2 Updating Documentation	103
22.3 Testing Documentation Changes	104
22.4 Pushing to a contributor fork	104
22.5 Updating Screenshots	104
22.6 Updating Upgrade Guides	105
22.7 Style Guide	105
23 Testing SecureDrop	109
24 Testing: Application Tests	111
24.1 Running the Application Tests	111
24.2 Updating the Application Tests	111
25 Testing: Configuration Tests	113
25.1 Installation	113
25.2 Running the Config Tests	113
25.3 Updating the Config Tests	113
25.4 Config Test Layout	114
25.5 Running the CI Staging Environment	114

26	demo.securedrop.org	117
26.1	How it works	117
26.2	Containers	117
26.3	Troubleshooting	117
27	Package repositories	119
27.1	Test repositories	119
27.2	QA repositories	119
27.3	Production repositories	120
27.4	How it works technically	120
28	Debian packages	121
28.1	About dbgsym packages	121
29	Updating OSSEC Rules	123
29.1	Alerting Strategy	123
29.2	Using ossec-logtest	123
29.3	Writing Automated Tests for OSSEC Rules	124
29.4	Adding new OSSEC rules	124
29.5	Deployment	126
30	Generating AppArmor Profiles for Tor and Apache	127
31	Portable SecureDrop Demo	129
31.1	Hardware	129
32	Release Management	131
32.1	Pre-Release	131
32.2	Release Process	134
32.3	Post-Release	135
32.4	Releases that only modify code on Tails workstations	136
33	Build metadata	137
33.1	Build logs	137
33.2	buildinfo	137
34	Linux kernel maintenance	139
34.1	Testing a new kernel manually	139
35	Updating Tor	141
35.1	Identifying new releases	141
35.2	Testing	141
35.3	Promoting	141
36	Setting up the SecureDrop Workstation	143
36.1	Install Qubes	143
36.2	Development Environment	144
36.3	Staging Environment	146
37	SecureDrop Workstation Development	149
37.1	Testing	149
37.2	Automatic updates	150
37.3	Manually updating dom0 code	150
37.4	Building workstation Debian packages	150
37.5	Building workstation RPM packages	150

38 SecureDrop Workstation Release Management	151
38.1 Communications Process	151
38.2 Release an RPM package	151
38.3 Post-Release tasks	153
39 SecureDrop Inbox Release Management	155
39.1 Development timeline	155
39.2 Workflow	155
39.3 Using a release branch	158
39.4 Signing procedures	158
40 SecureDrop Client Development	161
40.1 Developer Setup	161
40.2 How to Find Help	161
40.3 Client Architecture	161
40.4 Client Database Structure	162
40.5 Tests	164
40.6 Contributing	164
41 SecureDrop App Development	165
41.1 Repository Overview	165
41.2 Development Setup	165
41.3 Synchronization Behavior	165
42 “Seen By” Feature	169
42.1 Overview	169
42.2 Current Implementation	169
42.3 State Flow Diagrams	170
42.4 API Integration	170
42.5 Additional Implementation Considerations	170
Index	173

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; and
- *Translators*, to translate SecureDrop.

You can always find a regular project contributor to answer any questions you may have on the [SecureDrop instant messaging channel](#).

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

1.1.1 SecureDrop Workstation, Client, and supporting applications

The [SecureDrop Workstation](#) is a mixture of Python scripts and Salt configuration-as-code. See [SecureDrop Client Development](#) for more information.

The graphical [SecureDrop Client](#) app is written in Python, using the Qt GUI framework, and includes other supporting apps (Python/Rust), services (systemd), and packages (Debian). See [SecureDrop Workstation Development](#) for more information.

The “good first issue” tag is a good introduction to work available in both the [Workstation](#) and [Client](#) repositories.

1.1.2 SecureDrop Server

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](#).

The “good first issue” tag is a good introduction to work available in the SecureDrop Server.

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](#)

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 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 our [Gitter chat room](#) or through Localization Lab communications platforms.



SETTING UP THE DEVELOPMENT ENVIRONMENT

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 the necessary development tools:

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

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

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 13, 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-13-xfce 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 dev-tor                           # run development servers (onion_
```

(continues on next page)

(continued from previous page)

```
↪addresses)
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](#).

Credentials

The *Journalist Interface* requires a login. A test administrator (`journalist`) and non-admin user (`dellsberg`) are created by default when running `make dev`. In addition, sources and submissions are created.

The test users have the following credentials. Both 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`

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'
```

Persistent storage

By default the development environment doesn't store any server data once shut down.

If you set the `USE_PERSISTENT_STORE=true` environment variable, it will create a persistent `sd-store` volume and use whatever is stored in that. This is especially useful if you want to test against a large number of sources that take a while to initially generate.

You should be able to export, import, and delete the volume using the standard Docker/Podman commands.

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:

- 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 How is SecureDrop translated?

SecureDrop is translated using the *Weblate platform*.

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.

4.1.3 When does SecureDrop's translation happen?

All of SecureDrop's *components* can be translated continuously, as new code and *source strings* are merged into the project. Translations are then finalized during the release process.

This process is coordinated in collaboration with [Localization Lab](#). You can watch for the announcements published in *multiple locations*.

4.1.4 Workflow Diagram

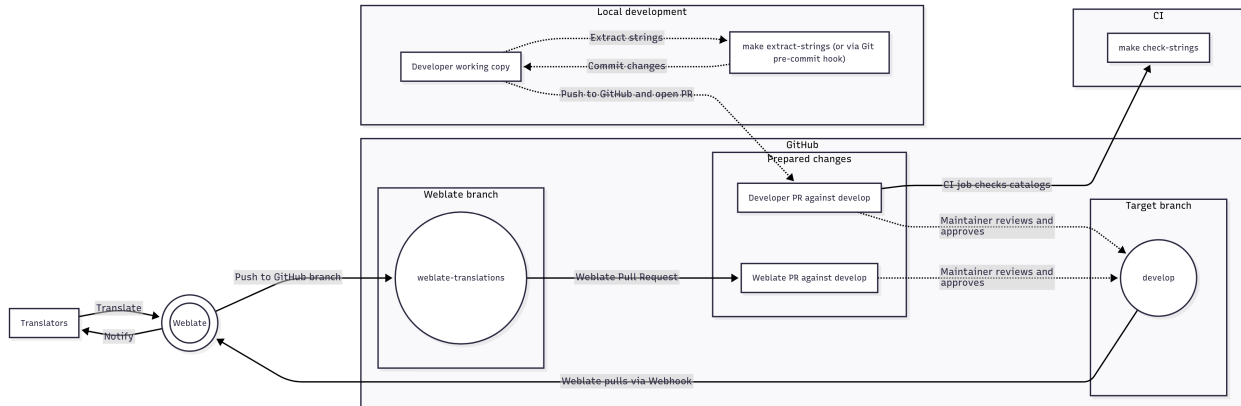


Fig. 1: A workflow diagram showing the continuous translations process for the SecureDrop server repository (click to enlarge).

4.1.5 Translation Responsibilities

1. **Developers** must use `make extract-strings` to keep the catalog template up to date with changes they've made to UI strings in the Python source code. Developers are encouraged to commit `make extract-strings` changes *along with* the source-code changes that caused them, for a cleaner Git history to review and (if necessary) `blame` and `revert`.
 - CI will enforce this requirement via `make check-strings` on branches pushed to this repository. Like the other linters, this check must pass for a pull request to be approved for merge into `develop`.
 - Developers can run `make check-strings` locally.
2. **Maintainers** should keep in mind that pull requests they review will include changes to the catalog template for any strings changed in the source code. They should consider the impact and timing of these changes on translators—just as they consider the impact and timing of code changes on other developers—as part of their review of a pull request prior to approving it for merge into `develop` and thereby into Weblate.
3. **Translators** can always see the latest strings available to translate in `weblate`. They can translate new and changed strings continuously, as development progresses, and they can give feedback on strings well in advance of the release period.
4. **Release and localization managers** do not need to do anything special outside of preflight testing! As summarized above, developers are responsible for keeping strings up to date; Weblate is responsible for keeping the per-language editable `.po` catalogs and loadable `.mo` machine objects up to date.

4.1.6 Tutorial for Developers

Let's say you've made the following string change in the Client's Python source code:

```
$ git diff
diff --git a/securedrop_client/app.py b/securedrop_client/app.py
index 6b95eda..13ee15d 100644
--- a/securedrop_client/app.py
+++ b/securedrop_client/app.py
@@ -175,7 +175,7 @@ def prevent_second_instance(app: QApplication, unique_name: str) ->
->None:
     if e.errno == ALREADY_BOUND_ERRNO:
```

(continues on next page)

(continued from previous page)

```

        err_dialog = QMessageBox()
        err_dialog.setText(
-         _("{application_name} is already running").format(
+         _("{application_name} is already running!").format(
            application_name=app.applicationName()
        )
    )

```

When you commit and push this change, you'll receive the following error in CI:

```

writing PO template file to securedrop_client/locale/messages.pot
Translation catalog is out of date. Please run "make extract-strings" and commit the
changes.
make: *** [check-strings] Error 1

```

As prompted, run `make extract-strings` and commit and push the changes:

```

$ make extract-strings
[...]
writing PO template file to securedrop_client/locale/messages.pot
$ git diff
diff --git a/securedrop_client/app.py b/securedrop_client/app.py
index 6b95eda..13ee15d 100644
--- a/securedrop_client/app.py
+++ b/securedrop_client/app.py
@@ -175,7 +175,7 @@ def prevent_second_instance(app: QApplication, unique_name: str) ->
None:
     if e.errno == ALREADY_BOUND_ERRNO:
         err_dialog = QMessageBox()
         err_dialog.setText(
-         _("{application_name} is already running").format(
+         _("{application_name} is already running!").format(
             application_name=app.applicationName()
         )
     )
diff --git a/securedrop_client/locale/messages.pot b/securedrop_client/locale/messages.
pot
index 51c95cc..524cdde 100644
--- a/securedrop_client/locale/messages.pot
+++ b/securedrop_client/locale/messages.pot
@@ -16,7 +16,7 @@ msgstr ""
"Content-Transfer-Encoding: 8bit\n"
"Generated-By: Babel 2.9.1\n"

-msgid "{application_name} is already running"
+msgid "{application_name} is already running!"
msgstr ""

msgid "The SecureDrop server cannot be reached. Trying to reconnect..."
$ git commit --all --message "changes a string"
[i18n 3637b3d] changes a string
2 files changed, 2 insertions(+), 2 deletions(-)

```

When you open a pull request for your branch, a maintainer will review your string changes for their translation impact.

4.2 Weblate

4.2.1 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](#).

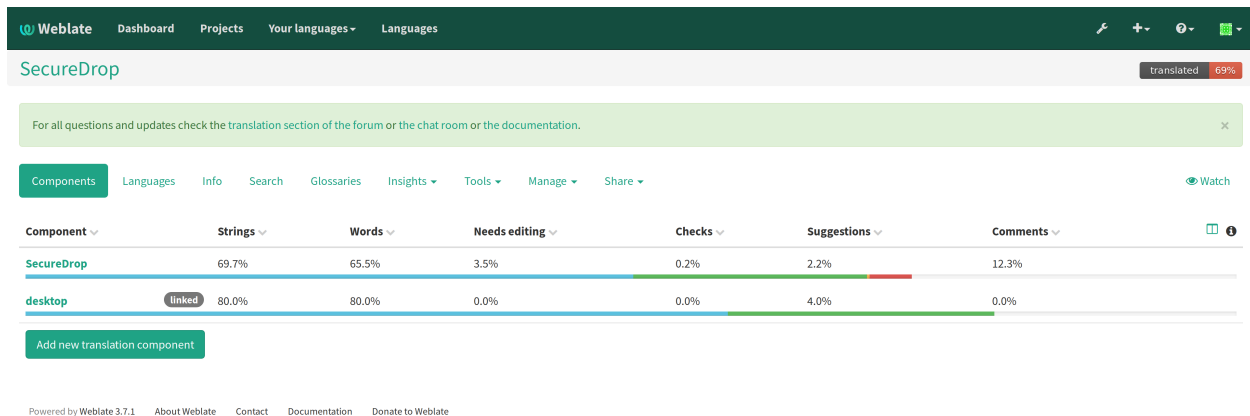


Fig. 2: Components of the SecureDrop project in Weblate. Click on the image to see it full-size.

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](#).

4.2.2 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.

By registering you agree to use your name and email in version control system commits and provide your contribution under license defined by each translated project.

Register using email

Email

Activation email will be sent here.

Username

Username may only contain letters, numbers or the following characters: @, +, -, _

Full name

What is 6 X 6?

Register

Third party registration

GitHub

Powered by Weblate 3.7.1 About Weblate Contact Documentation Donate to Weblate

Fig. 3: Weblate registration page. Click on the image to see it full-size.

4.2.3 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:

4.2.4 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.

4.2.5 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.*

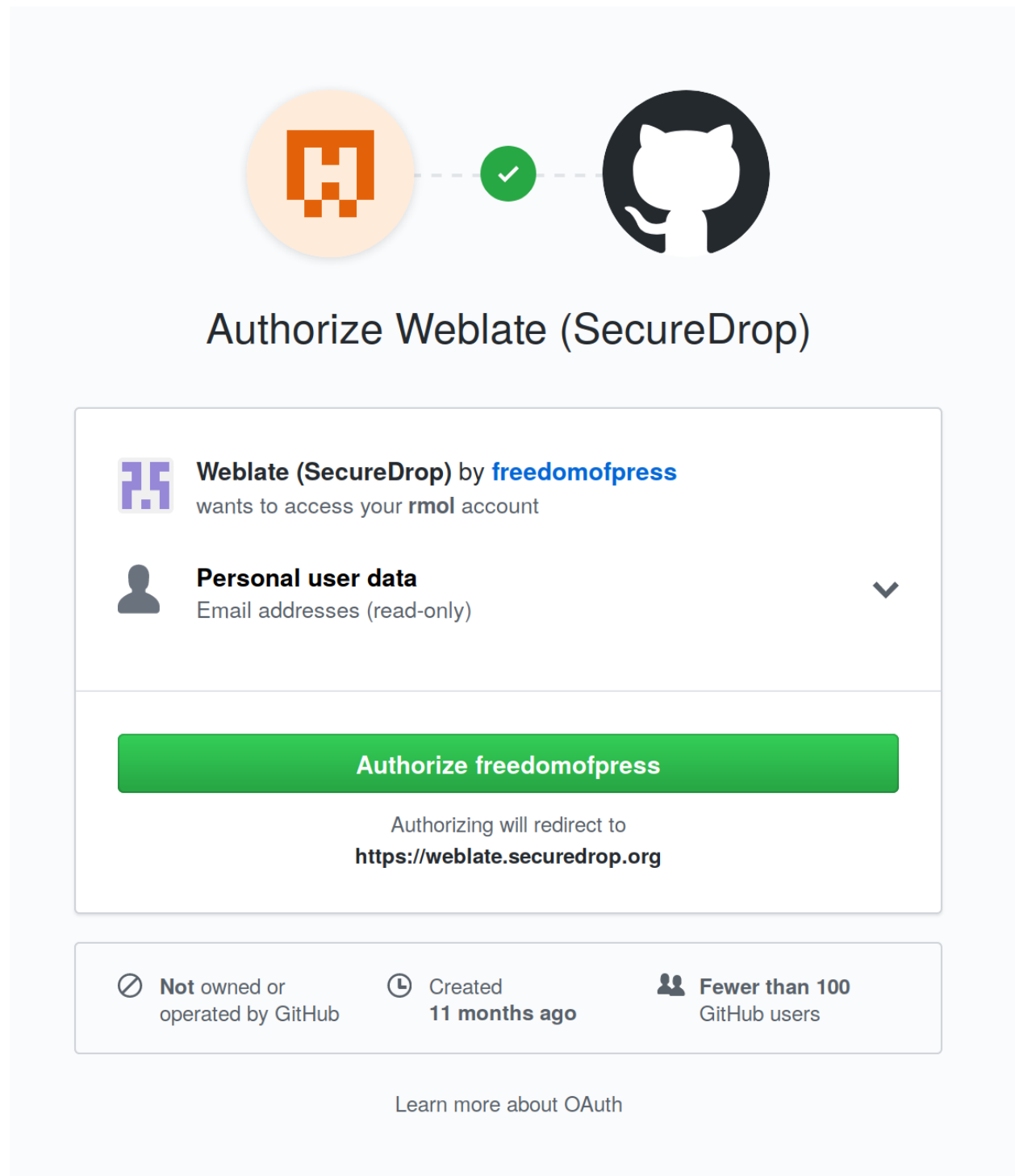


Fig. 4: GitHub authorization request. Click on the image to see it full-size.

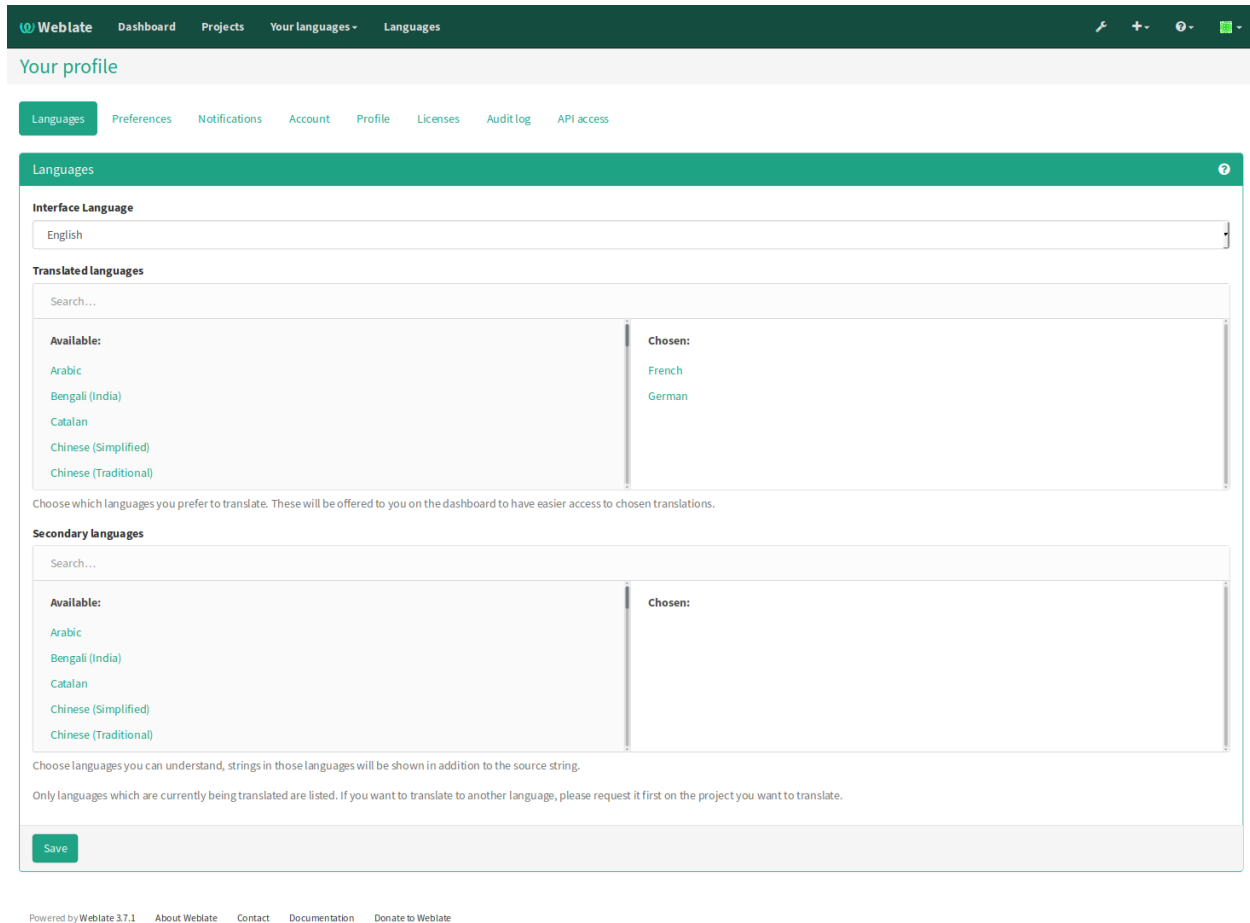


Fig. 5: Language preferences in Weblate. Click on the image to see it full-size.

SecureDrop / SecureDrop

translated 69%

For all questions and updates check the translation section of the forum or the chat room or the documentation.

Translations Info Search Glossaries Insights Tools Manage Share Watch

Language	Strings	Words	Needs editing	Checks	Suggestions	Comments	
Arabic	100.0%	100.0%	0.0%	2.8%	5.1%	17.3%	Translate
Bengali (India)	1.6%	1.3%	0.0%	0.0%	0.0%	13.0%	Translate
Catalan	100.0%	100.0%	0.0%	0.0%	0.0%	11.8%	Translate
Chinese (Simplified)	51.6%	40.7%	0.0%	0.0%	2.4%	13.0%	Translate
Chinese (Traditional)	100.0%	100.0%	0.0%	0.0%	0.0%	13.0%	Translate
Czech	37.0%	26.0%	0.0%	0.8%	6.7%	13.0%	Translate
Danish	8.7%	5.6%	0.0%	0.4%	2.8%	13.0%	Translate
Dutch	100.0%	100.0%	0.0%	0.0%	1.2%	14.2%	Translate
Finnish	80.7%	73.3%	6.7%	0.0%	8.7%	12.6%	Translate
French	100.0%	100.0%	0.0%	0.0%	0.0%	13.8%	Translate

Fig. 6: List of available languages in Weblate. Click on the image to see it full-size.

4.2.6 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.

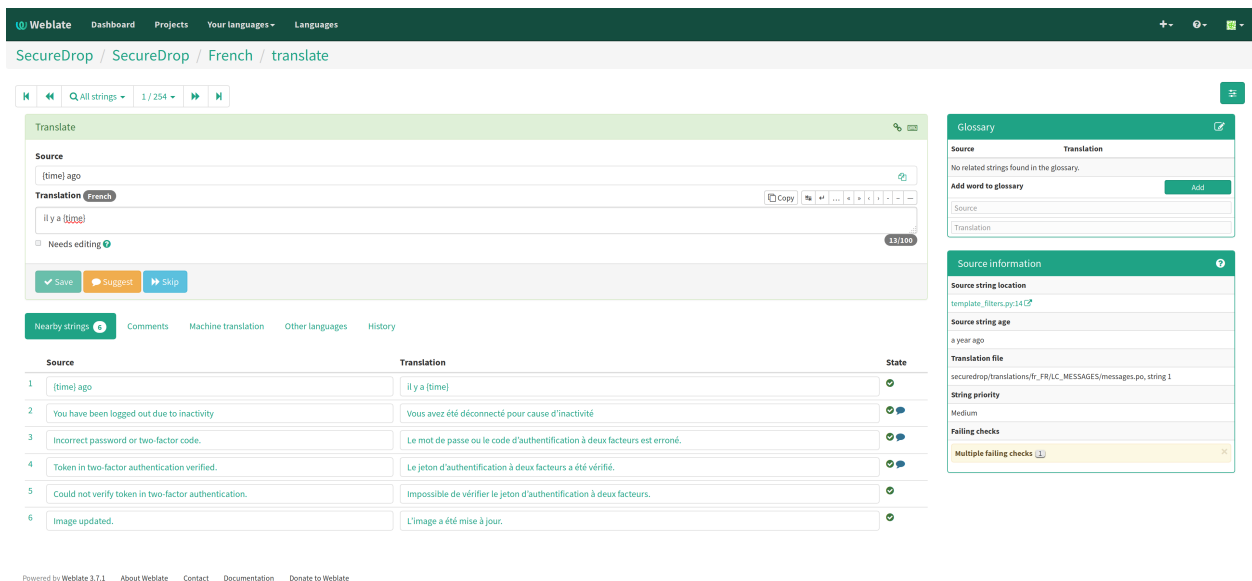


Fig. 7: Translating a phrase in Weblate. Click on the image to see it full-size.

- [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](#)

4.2.7 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.

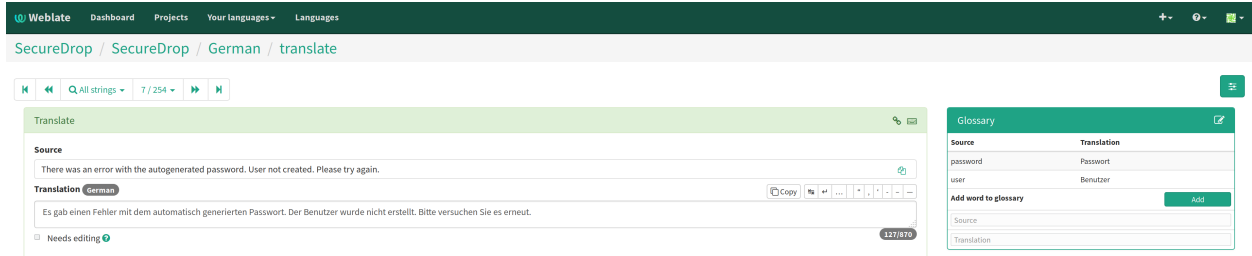


Fig. 8: Glossary sidebar in Weblate. Click on the image to see it full-size.

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.

4.2.8 Glossary

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

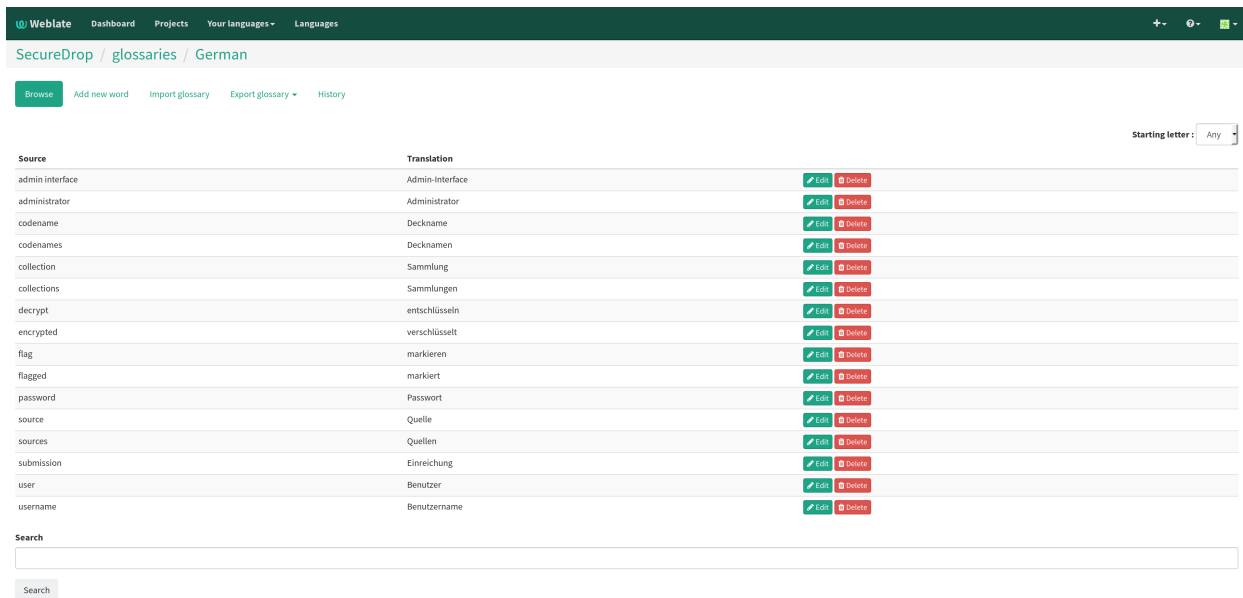


Fig. 9: A language glossary in Weblate. Click on the image to see it full-size.

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.

Reviewer

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

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](#).

4.3 How-To Guides

4.3.1 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.

4.3.2 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.
3. Click on the "Journalist Interface" link.
4. Input the **username**, **passphrase** and **current TOTP token** ("Two-factor Code") to log in.

4.3.3 How to become a reviewer

Contact [Localization Lab](#) to ask to join a [language team](#) as a reviewer.

4.3.4 How to add a new language to SecureDrop

We love seeing SecureDrop translated into new languages. Contact [Localization Lab](#) to ask to join a [language team](#) (or start a new one) and have the new language added to Weblate.

However, SecureDrop only supports a subset of all the languages being worked on in [Weblate](#). New languages are supported according to the [Policy on Supported Languages](#).

4.3.5 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.

4.3.6 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}
```

4.3.7 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: `<strong>`) 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 (`<img>`) 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 `<img>` 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.

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 Code Review

Changes to SecureDrop code or documentation (in any repository) must be reviewed by a SecureDrop maintainer. This is generally done in the form of GitHub pull requests (PRs), which are formally reviewed and merged.

If multiple individuals collaborate on a single PR, it is acceptable for cross-review by the authors to take place during development, but there must not be any unreviewed changes in the pull request at the time of approval.

When adding a new component to SecureDrop, it is the developer's responsibility to ensure it undergoes full review before integration. For example, if you create a new GitHub repository, it must not be added as a dependency of production code before the repository has been reviewed in its entirety.

If a component review is done by other means than a pull request, please ensure that a record of the review is kept in an issue that has "code review" in its subject, so that it is discoverable at a later date.

5.3 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](#).

5.4 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.

5.5 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 .githubhooks/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
```

5.5.1 Python

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

```
make flake8
```

5.5.2 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.

5.5.3 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.

5.5.4 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
```

5.6 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).

5.6.1 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

def main():
    # type: () -> None
    config = Config() # type: Config
```

(continues on next page)

(continued from previous page)

```
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.

5.6.2 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
```

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

5.8 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 on [Gitter](#) 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.

5.9 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:

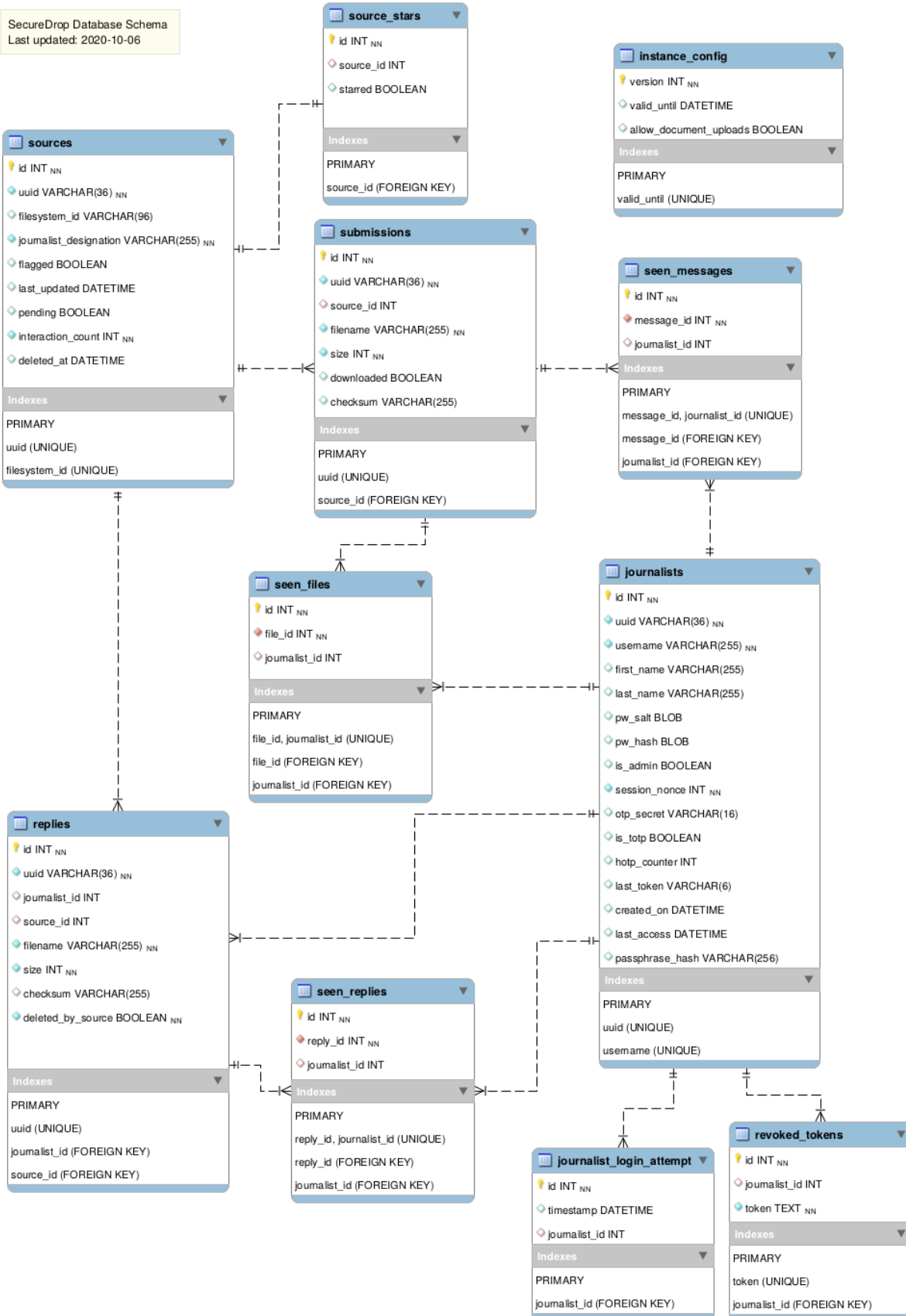
```
make update-pip-requirements
```

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:



POLICY ON SUPPORTED LANGUAGES

Version	Approved
5 (internal link)	8 March 2023

7.1 Definitions

Note

The key words *MUST*, *MUST NOT*, *REQUIRED*, *SHALL*, *SHALL NOT*, *SHOULD*, *SHOULD NOT*, *RECOMMENDED*, *MAY*, and *OPTIONAL* in this document are to be interpreted as described in [RFC 2119](#).

translation freeze

The deadline for translations to be reviewed and merged in order to be included in a given release. For SecureDrop, this is *release day*.

7.2 Thresholds for Translation and Review Coverage

	Translation Coverage	Review Coverage ¹
To grant support	100%	100%
To maintain support	80%	100%

In addition to these thresholds, the SecureDrop team will:

1. always prioritize the translation of source-facing strings, given their importance for sources' security; and
2. inform Localization Lab when particular strings should be prioritized or even considered blocking for a given release.

7.3 Granting Support for a Language

Granting support for a new language consists of adding an entry in the `supported_locales` object in `securedrop's i18n.json` and/or in the “Localization” section in `securedrop-client's MANIFEST.in`. Other steps, such as communication, are at the discretion of the Localization Manager.

¹ Machine translation (e.g., Google Translate) MAY be used to close gaps in review coverage for an otherwise well-supported language. (It MAY NOT be used to close gaps in translation coverage.) Because of the risk of low-quality machine translations especially from minority languages, machine translation SHOULD be considered a last resort, on a case-by-case basis in consultation with Localization Lab.

1. A language *L* that reaches coverage in time for a release version *V*'s *translation freeze* SHOULD be nominated for support in version *V*.
2. The Localization Manager SHOULD ask Localization Lab whether they believe *L*'s *language team* is likely to be able to maintain coverage for the foreseeable future.
 1. If so, the Localization Manager SHOULD grant support for *L*.
 2. If not, the Localization Manager MUST NOT grant support for *L*.

7.4 Revoking Support for a Language

Revoking support for a currently-supported language consists of removing the language's entries in `i18n.json` and/or `MANIFEST.in`. Other steps, such as communication, are at the discretion of the Localization Manager.

Consider an expected release timeline as follows:

Version	<i>Translation Freeze</i>
V1	January 1
V2	March 1
V3	May 1

Then:

1. A language *L* that misses coverage for a release version *V1*'s *translation freeze* MUST be considered on probation for up to the next two releases *V2* and *V3*. While on probation, a language is still considered supported until it has missed coverage for a total of 3 consecutive translation freezes.
 1. In consultation with Localization Lab, the Localization Manager MAY consult the *language census* (internal link) and reach out to administrators who may be able to contribute to translation and review.
2. If *L* misses coverage again for *V2*'s translation freeze and does not regain coverage for *V3*'s translation freeze, then the Localization Manager SHOULD revoke support for *L* for *V3*.
 1. In consultation with Localization Lab and the Release Manager, the Localization Manager MAY extend *L*'s probationary period, for example if the *language census* indicates that revoking support for *L* would jeopardize the default locale for many instances, for especially high-traffic or high-profile instances, etc.

7.5 Adding a New Language for Translation

Translators MUST ask Localization Lab to add a new language for translation in Weblate.

7.6 Technical Limitations

- Although Weblate, our localization platform, supports arbitrary languages for translation and review, at runtime SecureDrop components are generally limited to locales with ISO 639-1 codes registered in the Common Locale Data Repository (CLDR).²
- SecureDrop cannot be translated into the following constructed languages, because their ISO 639-1 codes are used in our automated tools and tests:
 - Esperanto (eo): used for *pseudolocale* generation
 - Interlingua (ia): used for testing

² For example, to be supported by securedrop, a new language's ISO 639-1 code must be registered in the CLDR release used by the version of Babel we install.

- Ido (io): reserved for future use

CONTINUOUS INTEGRATION

The SecureDrop project uses [GitHub Actions](#) for running automated continuous integration on code changes. You can get an overview of what each project does by reviewing the Makefile and files in the `.github/workflows` folder of the project's repository.

8.1 Basics

Our CI runs a mixture of linters, tests and build processes to validate code submissions.

Most tasks have a corresponding `make` target that will run the same command locally. Common targets across all our projects include:

- `make lint`: run linters
- `make fix`: apply automated fixes from formatters and linters
- `make test`: run automated tests

In CI, these are run in a container using the corresponding Linux distribution (e.g. Debian or Fedora), which can also be used to reproduce CI results locally. Some projects, like `securedrop` (server) and `securedrop-workstation`, automatically run commands in containers.

8.2 Pull requests

Most CI jobs are [triggered](#) by both `push` and `pull_request` events. The former is run against your branch, while the latter is run against your branch merged into `main` (or `develop`).

8.3 Special branch prefixes

In the `securedrop` repository, some slower jobs are only triggered if a specific branch prefix is used when creating the pull request. Currently these are:

- `stg-*`: runs a staging build in Google Cloud, see [Configuration Tests](#)
- `l10n-*`: runs localization tests across all 20+ supported languages

8.4 Nightlies

For `securedrop-workstation` and `securedrop-client`, packages are built for every merged commit as well as every night. These packages are published to the test yum and apt repositories.

A “nightlies” workflow runs in each repository that builds the respective packages. The workflow uses an authenticated token for the sdcibot GitHub account to push the packages and build metadata to `build-logs`, `securedrop-apt-test` and `securedrop-yum-test`.

The `securedrop-apt-test` and `securedrop-yum-test` repositories have a workflow that automatically prunes older packages, also using a token for sdcibot to push to themselves.

8.5 Workstation CI

For testing `securedrop-workstation`, we run a special CI job that virtualizes Qubes OS inside of VMWare. Documentation for this is available in the [securedrop-workstation-ci](#) repository.

DEPENDENCY SPECIFICATION AND UPDATE POLICIES

This document describes the process for updating SecureDrop dependencies. Since dependencies are run in the production environment, care should be taken when adding or updating dependencies to minimize risk. The following guidelines describe the process for adding or updating dependencies and specifying supported version constraints. They should be followed by the PR author or reviewer at the time of PR review.

Authors of PRs are encouraged to perform the investigations described below when updating or adding dependencies, and post the results of their investigation on the PR. Avoid if possible putting all the burden of dependency review on the PR reviewer. Please note that the following guidelines do not apply to dev, test, or deployment only (e.g. `s3transfer`, provided the deployment artifacts are signed, which they should be) dependencies.

We use tools that pin every dependency to a specific version and verify they match published checksums. Historically we used `pip-compile` for this purpose; we are currently in the process of migrating to `poetry`. For JavaScript, we use `pnpm`.

9.1 Adding a dependency

Before a new dependency should be added, a review should be performed. The following factors should be considered:

1. **Is this dependency well-maintained?** Are there recent commits or releases? Are high priority bugs on their bug tracker responded to and fixed?
2. **How secure is this dependency?** Have there been vulnerabilities reported in the project before? How have they responded? Do any of its dependencies have known CVEs? In lieu of a full code review (which might be a high burden), one might also run bandit static analysis on the Python dependency, are there high severity issues?
3. **How popular is this dependency?** How many GitHub stars does it have? Do other well-known projects depend on it? One can look at the GitHub dependency graph, e.g. Flask, in order to see the number of projects that use the dependency. By relying on well-known, widely-used dependencies, we benefit from the many eyes that should be evaluating it.

9.1.1 When to upgrade a dependency

In general dependencies should be upgraded whenever there is some specific need, whether it's for a new feature or to address a security vulnerability.

If the upgrade fixes security vulnerabilities, it should be prioritized based on the severity. Non-security releases should be upgraded to after a [7-day cooldown](#).

It may also be useful to upgrade dependencies to the latest major version so that any potential security fixes are easier to upgrade to.

Development-only dependencies can be upgraded on a regular cadence without needing a concrete reason, with the belief that we'll benefit from improved tooling and linters.

9.2 Updating dependencies

When updating a dependency, one should:

1. **Review the changelog:** were any high-risk areas of the code modified? Were bugs with security implications fixed?
2. **Review the diff:**
 - For routine updates of PyPi or NPM dependencies, use *GuardDog* to locally scan the updated version of the dependency. GuardDog is particularly focused on identifying common patterns used in supply chain attacks.
 - For dependencies with lower trust or otherwise requiring heightened scrutiny, use *Semgrep* to locally scan the updated version of the dependency. Semgrep is an all-purpose static code analysis tool.
 - For packages where the highest level of scrutiny is warranted, perform a *manual diff review* with the assistance of a tool like *diffoscope* locally to view the diffs in the source code.

See below for details on each of these techniques to review an updated dependency.

3. **Explain version specifiers:** Use comments in `.in`, `pyproject.toml`, or `package.json` files to explain why you are specifying certain versions or ranges.

9.2.1 dependabot automated updates

GitHub’s *dependabot* can be used to automatically propose pull requests for dependency updates.

In addition to the normal review process, the reviewer should verify the GitHub-provided checksums match those published for the package on pypi.org. To do so, locate the package on PyPI, select the correct version from the “Release history” page, and click “Download files”. Identify the files corresponding to the Dependabot diff, and click “view hashes” to compare the hashes.

You should see a dialog similar to this one:

Hashes for importlib_metadata-6.4.1.tar.gz
×

Algorithm	Hash digest	
SHA256	eb1a7933041f0f85c94cd130258df3fb0dec060ad8c1c9318892ef4192c47ce1	Copy
MD5	b852ac8a5bce8c89feb12dd2de1e3e0f	Copy
BLAKE2b-256	20fca0e728307f38609ef849d813c95dc974d344a3d395f62013ddcd8fbe0bfe	Copy

Close

Fig. 1: Example dialog for displaying hashes of a Python dependency published on PyPI

9.3 Specifying version constraints

For certain high-risk dependencies, we carefully control when and how far they're updated. As a general rule, we don't want to accept major version changes without substantial testing. For some projects, like Ansible before it was semver-compatible, we only want to permit patch changes, not major or minor. For example, in a `requirements.in` file:

```
# bad
ansible>=2.9.13
# good
# v2.10.0 is a breaking change, requires custom update logic
ansible>=2.9.13,<2.10.0
```

Make sure to provide a comment explaining the version constraints, so that future maintainers will have an easier time making sense of the controls. If no constraint is specified, then the latest version is assumed to be appropriate. Typically, you should set a lower bound on the target version you just upgraded to (for example, due to a `safety` alert):

```
Jinja2>=2.11.3
```

Doing so clearly indicates to other maintainers that no version less than 2.11.3 should be used. The next time the requirement is updated, the lower bound should be adjusted accordingly.

For projects using poetry, the `^` semver operator should be used, which only permits semver minor and patch updates:

```
[tool.poetry.dependencies]
sphinx = "^6.1.3"
```

9.3.1 Additional comments

These same processes should be followed for the dependencies of the dependency highlighted in the diff.

Some package updates will require new reproducible wheels to be published in the `securedrop-builder` repository; this should be done at the same time as the dependency update.

9.4 Scanning updated dependencies with GuardDog

GuardDog independently scans dependencies specifically for patterns indicative of supply-chain attacks.

You can install and run `GuardDog` locally using `pip` or `Docker` as you prefer. It's also important to keep GuardDog up-to-date.

Install using `pip`:

```
pip install guarddog
```

Update using `pip`:

```
pip install --upgrade guarddog
```

Install using `Docker`:

```
docker pull ghcr.io/datadog/guarddog
alias guarddog='docker run --rm ghcr.io/datadog/guarddog'
```

Update using `Docker`:

```
docker pull ghcr.io/datadog/guarddog:latest
```

Note

GuardDog fails quietly, and scans that did not run produce output similar to a successful scan with no findings. For this reason, you should pass `--log-level debug` with every invocation.

9.4.1 Scanning PyPi packages with GuardDog

GuardDog can scan any package directly from PyPi:

```
guarddog --log-level debug pypi scan cryptography --version 2.7
```

By default, GuardDog will grab wheels from PyPi. We typically pin python dependencies to tarballs rather than wheels. There are no flags to prioritize tarballs over wheels, so in these cases, you should download the target version's tarball off of PyPI and then run GuardDog against this local artifact:

```
guarddog --log-level debug pypi scan /path/to/TARBALL.tar.gz
```

9.4.2 Scanning NPM packages with GuardDog

GuardDog can also scan packages directly from *npm*:

```
guarddog --log-level debug npm scan react --version 19.2.0
```

To scan the updated version of a package locally, you can download the package tarball from *npm*:

```
pnpm view react@19.2.0
```

The output of `npm view` includes the integrity sha512 hash that should match the hash in the updated *pnpm-lock.yaml* file. You can compute this hash of the tarball locally:

```
cat ./react-19.2.0.tgz | openssl dgst -sha512 -binary | base64
```

To scan the package locally, decompress the tarball and run GuardDog:

```
guarddog --log-level debug npm scan ./package
```

9.5 Scanning updated dependencies with Semgrep

Semgrep performs detailed static code analysis. While not the same as a comprehensive diff review, Semgrep can be used to target specific changes for further examination.

You can install and run [Semgrep](#) locally using `pip` or `Docker`.

You will need to use a free Semgrep account to access the SAST scanning feature. You can use an existing GitHub account for SSO with Semgrep. Run `semgrep login` to launch a browser session where you can create an account or login and create a token to use in your terminal session.

Semgrep can scan any local file or code repository. For a dependency update we recommend cloning the repository of the package so that you can target the scan to the code that has changed in the update. Checkout the version/tag you are updating to, and use the `--baseline-commit` flag to indicate the hash of the commit of the version/tag you are updating *from*.

GuardDog uses a combination of [YARA](#) and Semgrep rules. You can thus instruct Semgrep to use GuardDog's Semgrep rules via the `--config` flag if you also have GuardDog installed.

Below is an example series of commands to clone a package repository, and scan the code changes between versions using the local GuardDog rules as well as Semgrep's default rules:

```
git clone https://github.com/remix-run/react-router/
cd react-router
git fetch --tags
git checkout v7.13.0 #the version we are upgrading to
semgrep scan --baseline-commit=4a5e333 --config=/home/user/guarddog-env/lib/
python3.13/site-packages/guarddog/analyzer/sourcecode ./
```

9.6 Manual dependency diff review procedure

1. Download the source tarball from pypi.org for both the version from which you are starting your diff review and the target version, example here for the diff review for cryptography 2.3 to 2.7:

```
$ pip download --no-binary :all: --no-deps cryptography==2.3
$ pip download --no-binary :all: --no-deps cryptography==2.7
```

2. Compute the sha256 hashes:

```
$ shasum -a 256 cryptography-{2.3,2.7}.tar.gz
c132bab45d4bd0fff1d3fe294d92b0a6eb8404e93337b3127bdec9f21de117e6  cryptography-2.3.
tar.gz
e6347742ac8f35ded4a46ff835c60e68c22a536a8ae5c4422966d06946b6d4c6  cryptography-2.7.
tar.gz
```

Verify that these hashes match what's in the requirements file (before and after).

3. Now perform a timeboxed review of the diff using diffoscope or your tool of choice, e.g.:

```
$ tar xvzf cryptography-2.3.tar.gz
$ tar xvzf cryptography-2.7.tar.gz
$ diff -r cryptography-2.3 cryptography-2.7 | more
```

(Fun tip: use `script` to write diffoscope output to a file while preserving colors, and review at your leisure with `less -R`.)

If you find issues, discuss with other team members and escalate upstream where necessary.

4. Else, make a signed document containing the source tarball hashes before/after, and sign it:

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA512

Diff reviewed from:

c132bab45d4bd0fff1d3fe294d92b0a6eb8404e93337b3127bdec9f21de117e6  cryptography-2.
3.tar.gz
e6347742ac8f35ded4a46ff835c60e68c22a536a8ae5c4422966d06946b6d4c6  cryptography-2.
7.tar.gz
-----BEGIN PGP SIGNATURE-----
```

(continues on next page)

(continued from previous page)

```
iQIzBAEBCgAdFiEEntsmvkbGwko38nhRsH6AZWrNlQEFA11a9swACgkQsH6AZWrN
lQGwbQ/+OwKgNKJuU44+zW8kBQ7l08oiLKf02kxBaGZYMBamd2/LMjATQGdQ8ocp
sIQ4YXq+ybInv3ZNP80k9tuFP3o+3PsXU1Mc0A3FZnH4wwxFUGckWV57drFIWuSR
pNOA03M8VaggMP7FaDFgSRa1BmjBIYGW/FL+nUUSQtKfWomGJEmVBo0Uxf8WXztP
lnio4BYYsQbhjGS1m1jXjNyrMkHefYluKQBs jcgU/dngTtOAD3GgAC6wczBztX
k7Afzmv9vJVRlEcZGkzfNNUJC5WCQjoH3y6DiyQYiIk9sIG0TbFvqNIG2azWY5b
AdlHgbS2qR01tIMzjpn5fiiXHeKJ8L8Y6tRTYGgkN9IIUawUhfkd0ExPB3OGOWz
4CItkwyrFUC6RtWH0oGhHNUaDeGWrh3TyHwjHE9kFQDDz+RXv1SpBkLmZof/UK0V
mK8TSs5LSa+WPTP8zbgjORcMmOZL44Hdnrxn0pfM+YhBDKp8bUC2wB9Rasew74y6
sM07lHyQQzDuPgvV/SrCVGJF8hDDA10VLXK3QasEdHnvlU5w4lM8uLb+rX2sC8Im
+HSpM3f4N7dbPTTft352+uSgD0vXGqzqwc0rovtEJcgb1T/IpN40QvnsrLQyMZ70
xPA9YoeZ0lHsgAazDPXfHRqsPmJslZp80uZqbf560ZPBBJKbuw=
=T0MH
-----END PGP SIGNATURE-----
```

Note that you generate an inline signature like this via: `gpg --clear-sign crypto-diff.txt`

- At this point, create a wiki page e.g. <https://github.com/freedomofpress/securedrop-debian-packaging/wiki/cryptography-2.3-to-2.7> containing the hashes before/after.
- Comment on the PR indicating that the diff review is approved.
- Send the same content from the wiki to diff-review@python.org.

9.7 Auditing Rust dependencies

We audit Rust crates using the [Cargo Vet](#) tool. To get started:

```
cargo install --locked cargo-vet
```

Then you can audit both new and updated crates:

```
cargo vet diff $CRATE $OLD $NEW      # $CRATE has been updated from $OLD to $NEW.
cargo vet inspect $CRATE $VERSION    # $CRATE is entirely new at $VERSION.
[...]
cargo vet certify
```

Running `cargo vet` suggest after updating or modifying dependencies will automatically provide you with the relevant `diff` and `inspect` commands to run.

Cargo Vet has two [default policies](#): *safe-to-deploy* and *safe-to-run*. We consider the lower *safe-to-run* policy to be equivalent to our standard practice for reviewing Python dependencies. Your own audits should certify *safe-to-run* unless you have the expertise, and have invested the time, to review up to the higher *safe-to-deploy* standard.

9.7.1 Trusting third-party audits

To benefit from work others have done in reviewing crates, we import and trust other organizations' audits. New organizations should be reviewed and discussed on a case-by-case basis amongst SecureDrop maintainers.

We also trust individual developers who are writing and releasing crates that we use. Currently we trust developers who are members of either the Rust Project or Sequoia-PGP. This trust is valid for 6 months and must be extended regularly.

9.7.2 Exemptions

SecureDrop only runs on the `x86_64-unknown-linux-gnu` [target](#), so we only need to audit code and crates that apply to it. For example, we can ignore all of the `windows-sys` crates.

Exemptions can be specified in `supply-chain/config.toml`:

```
[policy.windows-sys]
criteria = []
notes = "Windows-only"
```

Note that within crates do review, you should still review all the code, regardless of what platform it is targeting.

REPRODUCIBLE BUILDS

We have implemented [reproducible builds](#) for most of our projects (see the exceptions below).

10.1 Goals

A developer should be able to build Debian and RPM packages and get bit-for-bit identical results as another developer. This is primarily done at release time, to verify the packages were not altered in some way.

We also want third-parties to be able to take our code and reproduce the output to verify we are building from what we've published and not e.g. inserting secret backdoors.

We expect that any reproduction attempts will be done near the time of the release (on the scale of months). While we are supportive of efforts to reproducibly rebuild packages from years ago, we are not explicitly guaranteeing that.

10.2 Build environment

Our primary strategy for having reproducible builds is using a containerized build environment (docker or podman). By using containers, we ensure the build environment is:

- clean: no manual modifications have been made to it (aside from what is specified in a Dockerfile or equivalent)
- minimalist: only essential packages and build requirements are installed.
- up to date: all installed packages should be up to date and we should be able to verify this
- repeatable: it should be straightforward for another developer to set up an identical environment

10.3 Build metadata

We publish all the build metadata available to us, so that others have as much information as possible when attempting to reproduce our builds. See [build metadata](#) for more details.

10.4 Reproducible wheels

Python packages with C or other native dependencies are often not reproducible because the build process happens in a randomized temporary directory, which is then captured in the `.so`. We build wheels ourselves in a fixed path to ensure they're reproducible, see [securedrop-builder](#) for more details.

10.5 Not reproducible

Currently the `securedrop-app-code` package ([issue #5901](#)) and our Linux kernel builds ([issue #3](#)) are not reproducible.

10.6 Tools

We primarily use [diffoscope](#) to identify differences in everything; it can handle Debian packages, RPM packages, Python wheels, etc.

It also supports outputting markdown; if you pass `--markdown` - and pipe it to `gh gist create - -f diff.md` you can share and easily readable diffoscope report.

We have also used [reptest](#) in a few places, but recently have been phasing it out in favor of just repeating builds twice, in parallel, and diffoscoping the result.

END-OF-LIFE MAINTENANCE GUIDELINES

This document outlines the procedures for managing end-of-life (EOL) transitions for software components in the SecureDrop ecosystem, including both SecureDrop Server and SecureDrop Workstation environments.

11.1 Overview

End-of-life maintenance is a critical aspect of SecureDrop's security posture. When software components reach their EOL dates, they no longer receive security updates. This document provides guidelines for:

- Tracking EOL dates and timelines
- Planning and implementing upgrades
- Coordinating community communication

This document focuses on operating system EOLs. For additional component upgrades, see:

- *Updating Tor*
- *Linux kernel maintenance*
- *Dependency specification and update policies*
- *Release Management*
- *SecureDrop Workstation Release Management*
- *Package repositories* (relevant for EOL upgrades if new distributions have to be added)
- *Dependency specification and update policies*

Guidelines regarding hardware end-of-life are maintained in the documentation for SecureDrop administrators.

11.2 EOL Tracking and Monitoring

SecureDrop uses an automated EOL checker system located in the [securedrop-dev repository](#) to monitor component lifecycles.

Key Features:

- **Automated Issue Creation:** Opens GitHub issues at specified intervals before EOL dates
- **Calendar Integration:** Generates ICS files that developers can subscribe to for EOL date tracking
- **endoflife.date API:** Consumes data maintained by the <https://endoflife.date/> community

Note

The EOL checker requires manual updates for each new software release. Developers must update the YAML configuration when new versions are released to ensure proper monitoring.

11.3 SecureDrop Server Components

11.3.1 Ubuntu Operating System

The SecureDrop Server runs on Ubuntu LTS (Long Term Support) releases. The project follows a conservative upgrade strategy:

- **Upgrade Schedule:** Upgrade to every other LTS release (e.g., 16.04 → 20.04, skipping 18.04)
- **Timeline:** Plan upgrades well in advance of the current LTS reaching EOL
- **Testing:** Perform extensive testing in continuous integration and in developer-run staging environments.

Because newsroom admins typically have many other responsibilities besides SecureDrop, we want to keep the maintenance effort as low as possible. Starting with the Ubuntu 20.04 → 24.04 migration, we've pursued an in-place upgrade strategy.

Upgrade considerations include Python and Debian package compatibility across Ubuntu and Python versions, changes to configuration file formats, OS software components getting swapped out, and changes to the Ubuntu installer. This migration is therefore typically a heavy lift.

11.4 SecureDrop Workstation Components

The SecureDrop Workstation (SDW) operates on Qubes OS and requires management of multiple operating systems used in virtual machine templates.

11.4.1 Qubes OS

Qubes OS itself is updated on an irregular schedule:

- Qubes 4.0: March 2018
- Qubes 4.1: February 2022
- Qubes 4.2: December 2023

After each major release, the previous one is given 6 months until end-of-life. Given this short timeline, ideally, compatibility testing should begin no later than the first release candidate, after preliminary testing with nightly builds.

Note

Because the exact dates are not known ahead of time, the Qubes OS EOL is not tracked automatically.

11.4.2 Template VMs

Debian Templates

- **Usage:** Base template for all SDW virtual machines
- **Lifecycle:** Follow Debian stable release cycles (typically 2-year cycles) and availability of Qubes base template

- **Management:** Requires changes to SecureDrop Workstation provisioning logic. High-effort due to major package updates (e.g., Python version) that may require components to be rebuilt or upgraded, cause incompatibilities, etc.

Fedora Templates

- **Usage:** Base template for system VMs
- **Lifecycle:** Follow Fedora release schedule (typically 6-month cycles) and availability of Qubes base template
- **Management:** Requires updates to SecureDrop Workstation provisioning logic. Usually low-to-moderate effort. These templates are used by Qubes OS itself, so it is possible that future version of Qubes OS will provide more built-in automation here.

11.5 Communication Protocol

Low-risk upgrades that are performed automatically without user action can be announced through the relevant release blog post. A simple tracking issue is sufficient to plan this work.

For moderate to high-risk upgrades, we typically at least want to give admins the option to trigger them manually, so they can troubleshoot any unexpected issues. This involves additional advance notice, typically through blog posts, social media, and our support portal.

LANGUAGE TOOLCHAINS

12.1 Python

In nearly all cases we use the OS-provided version of Python, whether it's Ubuntu, Debian or Fedora. In some cases we may also install Python dependencies using OS packages, typically when it's something that's hard to build from scratch (e.g. Python Qt or GTK bindings).

We rely on the OS for security updates, so we only upgrade major versions when we upgrade OS versions.

With Python, we use a mix of `pip`, `poetry` and `uv` for package management. Older projects may still use `pip-tools`.

12.2 Rust

Unlike Python, which we get from Debian packages, we manage our own Rust toolchain in the SecureDrop server dev environment and package builder.

Rust releases new versions every 6 weeks. We aim to stay within 2-3 versions of the latest stable release, which allows us to update (at minimum) every 3-5 months.

12.2.1 Upgrading

The Rust version is specified in a number of files, including:

- `rust-toolchain.toml`
- Package builder's `Dockerfile`
- Dev environment's `Dockerfile`
- CI manifests

It is recommended to `grep` for the old version string to find any other places it might also be used.

As of this writing, Rust code is used by Sequoia-PGP redwood bridge and cryptography dependency. The following test plan can be used for smoke testing those:

```
* [ ] CI passes, including deb building and staging build
* [ ] Build new debs, deploy on a staging/prod instance:
    * [ ] Create a new source, upload a file.
    * [ ] Create new journalist, log in as them.
    * [ ] As the journalist, download the file and successfully decrypt it.
```

12.3 Node.js

Currently our primary usage of Node.js is with Electron in the SecureDrop App (still under development). Electron ships its own version of Node.js, so we match that version in CI and elsewhere.

For package management, we are using [pnpm](#) largely for its better control over build scripts, and because it's generally faster than npm.

JOURNALIST INTERFACE API

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

13.1 Versioning

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

```
/api/v1/
```

13.2 Content Type

Clients shall send the following headers:

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

13.3 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 eyJhbGciOiJIUzI1NiIsImV4cCI6MTUzMDU4NjU4MiwiYW0IjoxNTMwNTc5MzgyfQ.  
↪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.

13.3.1 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."  
}
```

13.4 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."  
}
```

13.5 Endpoints

13.5.1 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"  
}
```

13.5.2 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\
↪nmQINBFRfoABEACf5Y+6prky4JcWmKSsuh/52ZLw1FTCqrgAIK0QVFZ+cy2riFHv\
↪njQXYB4bPOCt7PmYbmMxxIWkXqJCaPvkLbpi7p5X2WkgH+qGgJjotq2Y9iPP6KQ3\nGvJdpG3rWwbOsr4rDh/
↪L/lStn+ty4io3cDr7l7IS0t0cmOPKeKv6eGxSmCAYsnJ\nKKsIWcSjfb82KhCzL/
↪BBAPqXt9uc6Jqjh1RPL3bGIG0tq37yX/zbFefDBDF8m8d6\nc7pvvYMa090PGViBVg6hh8+rPq/
↪rK7YyHOWZl7t6MXw7cm/GaH+DkGxGKe8Yuj92R\nOPNQFfpAI/tXldEcEvdG/
↪4mba7uxrEMe33tsnbQamFZtXFAIRsJXa904CEEwnRCz\nNE90u9FeM4bk/lModsr7gOrWbO6QwctVt/
↪YnvI7blUXzpMzDsbgvR89auKS9VHGZ\nY5L3yz0yVwRAIw3/
↪CwsJEYajKiPadcExhZhc8OCTTe8zPXxQ80WrvmfBA6x6cfvq\nSsqoH3NXrDVY/
↪6w9dCqVXitcYynATqm0Qkk81jXE3BEfx7AQPXHXGasvFM1mqeQU\n+WQPqUKheomy7/
↪7z3heasKub3MYLkuW6y7c31z6cmvt6h5fYcNPvQXCox4BJkVcK\
↪nPhbst612sbqhTQeEsDnVU1sPLxpfbxKfKuWQlEV8kfm4JsMbryqG9Z0RQARAQAB\
↪ntHxBdXRvZ2VuzXJhdGVkIEtleSA8UF1NR0IzRE9BNVFLVFozNjVPUTNQWUpDMk9a\
↪nQ0RXQjIyM1dFS1Q3V0o1NDI0QUZUT1ZFSjI0SEpaSFRYQTZTQjVGVkFBVjdHRVFQ\
↪nS01HQjQzUUxMVzNTRUxWFENYVWk1VRk5QWU2WT0+iQI3BBMBCgAhBQJRkX6AAhsv\
↪nBQsJCAcCBhUICQoLAgQWAgMBAh4BAheAAAoJEBcz2mVYVNsSQ88P/3e54noTBb/O\
↪nFVVNYw5oY9zIQPsoYUKCCvKCv26bi3qpfsDWjohyupKLth9AfFBT3oiNhzeFhiv\
↪nZ5RbLgJYAWuzWNdMCSd3RAqZbbzFx3255oR9t+/RNwjeOqKpo0313myAKsRR1z+N\nbRF0A1C8GiMOCrvV/
↪9p+rsTDrV+8fXkrQz55nGkt6JlI43EqLH0Eg7wxI+HMgTdz\
↪nsPWR63INNhkrR5Ln7YSh0BmnUWjpEjFYvZlAbzkMbbfzndZ2g7auRpT0S8vNgcG\
↪n9k9dG3gpMFnHiaE4Smd0Ib82qv9X6Q70wwxmz85JAe/P/CYsndUbRHSfXMP16igm\nj0RfcC7J0E/
↪SkwBY9jc+YtGCWfqqXa1a4uY03vN1YqqFWqb+exa/Qv14wwgcS17p\nn80/
↪X1y9gPV0qleikFgNt8sPd+a2lVdRSjh4Xh716eTHMqoDUJXtFu0evSg3oBFZj\nn80IXe8KZltJCYlXn+1/
↪xlVZjAVfmYT6kxOXYsPB3o3Z9Hemgsw2PnjI04ZMwTSyb\nn101xfgB1XBd1Hrv9WQ5PNoPwXRhx7/bfzQWTx/
↪uP8luT6yqEerLiF0m/ShvYvKQa\
↪nCLuwtW3Rlj1BD5CpdG+491jJ6cRXq8xfYmCd2MmBTtMAoq4DobYw75NKIssZ5gs6\
↪nu6NXuCW0sf8z1QNBKxkNpuohLlTef8n1y\nn=Zp4Z\nn-----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",
    }
  ]
}
```

(continues on next page)

(continued from previous page)

```

    "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+jhigpn1wZvDT8AyJqaEisUddREUw1CpvOdCfw1uIFfodz5GJmVXZnApN\n27BJKNnsJtL8lWrUvTY/
↪n4afXgMZ78ZH8aOkdmJ7wmVbIhrZlHu4UHJP6DbCm/+D\
↪n7o74ozWCv6si9bfbpG6UbCxVqaeRYjb1kGT0y36TLy8W6+JXw+yISgKT0RETTjQX\
↪nzzHP5gfLu8ZTJhSvMV+xpKxc0HaX6P80rQR40QfVYRg01uZ1Bfab+rPdUrQSPdnb\
↪ntN6RhrN0QfucupPYpiS8Ajl1Si9ztyIdkYLJTL/Cse06SWDc/krIj8mX4VbN0h0\
↪nYwECCbtv5uX8q3Jhkc8oTjpW+DRxfb1UW7us1nOoXVj9a0QaUM6QZtbVz0qQDJ9e\
↪nSOqIx2tv5qToTxKim8E9HjX+NCvZKDIqvaoDpreMHkFP/Fo0t0tnbHTZAWcUMaih\
↪n5WNqrFqpGYm1fdFYDIL9m3DPVaFhk3e07apxQXwDrckeRY7Bma+YLOXG4yVf/I6\
↪nKedgBz0Nxl9ZcU6c10Fy3Dn90jcjYtT0trEsVORdfe/1SVBKMA0jpYirnQARAQAB\
↪ntHxBdXRvZ2VuzZXJhdGvKIEtleSA8MldXQlhaRlo1Q1RYSkVCQzZYQUNZUVhMWlNN\
↪nNEdCuk0zUVlZWJjJmR0VPQUxQTEZKSjVCR1lPSzRZUzU0SkYSlQzTlhVTkpLQ0VH\
↪nTFU2RFVQUldGWEM1WlEzRk1UVFhDM0VSRlQzWT0+iQI3BBMBCgAhBQJRkX6AAhsv\
↪nBQsJCACBhUICQoLAgQWAgMBAh4BAheAAAoJEIhu6go2DZ/xLcgP/11EL1F7hoQr\nLQm8T/
↪DqjoExh0F8am9SKb2LH9HSBUJPY9b/oPjptxyg/3N1GXP/GJGcI6SVXtnq\nGU2D2+vMUUrnV/
↪AemAtBUIquIXMEUjbGdK0uWTBcntgj6PJL6/VNi2o+v9FxAtn1\n6hefcd0Ik7DMaK8y56BJA+aI/
↪7TnCr1ndHLUMXh0rKd8GS13vXtv2kuY8iSqiOmj\
↪nu0tW1w2lByFBglNLgnozdbudwwVqNvKX8j3oWJKsJ525Y3HsWka/l4GbkowveUYR\
↪nU66usAX6KS1zT01pLDmYFCL7LX8SPkZq97qHoFa1C9NIHW2gP+y8Q922E9QWBqy7\n/
↪g30ZF73MgZC0nFOChswH607LBvMGUyz+A2Qjpd7Zvf67G33inY7QlGkMI59Zz4T\nnXXv/
↪1U3G16LLkGWRTDhHgK2KA9+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/
↪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"
  }

```

(continues on next page)

(continued from previous page)

```
}
]
}
```

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+6prky4JcWmKSsuh/52ZLw1FTCqrgAIK0QVFZ+cy2riFHv\
    ↪njQXYB4bPOct7PmYbmMxxIWkXqJCaPVkLbpi7p5X2Wkgh+qGgjIjotq2Y9iPP6KQ3\nGvJdpG3rWwbOsrt4rDh/\
    ↪L/lStn+ty4io3cDr7l7IS0tOcmOPKeKv6eGxSmCAYsnJ\nKKsIWcSjfb82KhCzL/\
    ↪BBAppqXt9uc6Jqjh1RPL3bGIG0tq37yX/zbFefDBDF8m8d6\nc7pvvYMa090PGViBVg6hh8+rPq/\
    ↪rK7YyHOWZlt6MXw7cm/GaH+DkGxGKe8Yuj92R\nOPNQFfpAI/tXldEcEvdG/\
    ↪4mba7uxrEMe33tsnbQamFZtXFAIrSjXa904CEEwnRCz\nNE90u9FeM4bk/lModsr7gOrWbO6QwctVt/\
    ↪YnvI7blUXzpMzDsbgvR89auKS9VHGZ\nY5L3yz0yVwRAIw3/\
    ↪CwsJEYajKiPadcExhZhc8OCTTe8zPXxQ8OWrvmfBA6x6cfvq\nSsqH3NXrDVY/\
    ↪6w9dCqVXitcYynATqm0Qkkr81jXE3BEfx7AQPXHXGasvFM1mqeQU\n+WQPqUKheomy7/\
    ↪7z3heasKub3MYLkuW6y7c31z6cmvt6h5fYcNPvQXCox4BJkVcK\
    ↪nPBzst612sbqhTQEEssDnVU1sPLxpfbx fKuWQlEV8kfm4JsMbryqG9Z0RQARAQAB\
    ↪ntHxBdXRvZ2VuZXJhdGVkIETleSA8UF1NR0IzRE9BNVFLVfzNjVPUTNQWUpDMk9a\
    ↪nQ0RXQjIyM1dFS1Q3V0o1NDI0QUZUT1ZFSjI0SEpaSFRYQTZTQjVGVUkFBVjdHRVfQ\
    ↪nS01HQjQzUUxMVzNTRUxFWENYwklVRk5QWTU2WT0+iQI3BBMBCgAhBQJRkX6AAhsv\
    ↪nBQsJCACBhUICQoLAgQWAgMBAh4BAheAAAoJEBcz2mVYVNsSQ88P/3e54noTBb/O\
    ↪nFVVNYw5oY9zIQPsoYUkCCvKCv26bi3qpfsDWjohyupKLth9AfFBTk3oiNhzeFhiv\
    ↪nZ5RbLgJYAWuzWNdMCSd3RAqZbbzFx3255oR9t+/RNwje0qKpo0313myAKsRR1z+N\nbRF0A1C8GiMOCrvV/\
    ↪9p+rsTDrv+8fXkrQz55nGkt6JlI43EqLH0Eg7wxI+HMgTdz\
    ↪nsPWB6R3INNhrR5Ln7YShOBmnUWjpejFYvZlAbzkMbbfznDZ2g7auRpT0S8vNgcG\
    ↪n9k9dG3gpMFnHiaE4Smd0Ib82qv9X6Q7Owwxmz85JAe/P/CYsndUbRHSfXmp16igm\nj0RfcC7J0E/\
    ↪SkwBY9jc+YtGCWfqQXa1a4uY03vN1YqqFWqb+exa/Qv14wwgcS17p\n80/\
    ↪X1y9gPV0qleikFgNt8sPd+a21VdRSjh4Xh716eTHMQdUJXtFu0evSg3oBFZj\n80IXe8KZltJCylxN+1/\
    ↪xlvZjAVfmYT6kxOXYsPB3o3Z9Hemgsw2PnjI04ZMwTSyb\n101xfgB1XBd1Hrv9WQ5PNoPwXRhx7/bfzQWTx/\
    ↪uP8luT6yqEerLiF0m/ShvYvKQa\
    ↪ncLuwT3Rl1j1BD5CpdG+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,
  "number_of_messages": 2,
```

(continues on next page)

(continued from previous page)

```

    "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": [
        "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": "",
      "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,

```

(continues on next page)

(continued from previous page)

```

    "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"
  ],
  "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"
}
```

13.5.3 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",

```

(continues on next page)

(continued from previous page)

```

    "uuid": "00d24bed-8d13-4f90-b068-52341593a727"
  }
]
}

```

13.5.4 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"
    }
  ]
}

```

13.5.5 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.

For more information about the “seen by” feature implementation and behavior across different SecureDrop interfaces, see *“Seen By” Feature*.

```
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.

13.6 Removed functionality

13.6.1 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.

15.1 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.

15.2 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
```

15.3 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.

15.4 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.

16.1 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 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**.

16.2 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.

16.2.1 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 rmmod kvm_intel
sudo rmmod 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.

16.2.2 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.

17.1 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`.

17.1.1 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 R4.2 installation on a compatible laptop, with at least 32GB 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.

18.1 Overview

Follow the Qubes platform instructions in [Setting Up the Development Environment](#) to create a Debian 11 `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*

18.2 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.

18.3 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.

18.4 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.

18.5 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.

18.6 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`.

18.6.1 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.

18.6.2 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-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-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-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
```

(continues on next page)

(continued from previous page)

```
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-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.

18.6.3 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.

18.7 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
```

18.8 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](#).

18.9 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
```

18.10 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).

18.10.1 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.

18.10.2 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*.

19.1 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.

19.2 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.

20.1 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.

20.2 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.

20.3 Developer Workflow

20.3.1 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.

20.3.2 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
```

20.3.3 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):
        """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
```

(continues on next page)

(continued from previous page)

```
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.

20.3.4 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](#).

As string changes are merged into the `develop` branch in the [main SecureDrop repository](#), the changes will automatically appear in Weblate, and translation can begin. Translation and review can take place continuously, at any time. Translations for *supported languages* reviewed and finalized during the release process.

21.1 What languages are available where?

- All languages *translated* in Weblate are *present* in the `securedrop/translations` directory.
- *Supported languages* are listed in the `supported_locales` object in the `i18n.json` file.
- Those languages that are both *present* and *supported* are *available* for administrators to *configure* in `securedrop-admin sdconfig`.
- Those languages that are both *configured* and *available* on the *Application Server* are *usable* for users to select.

21.2 Development tasks

21.2.1 Add a new language

See *How to add a new language to SecureDrop*.

However, SecureDrop only supports a subset of all the languages being worked on in [Weblate](#). New languages are supported according to the *Policy on Supported Languages*.

21.2.2 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 extract-strings` in the root of the SecureDrop working copy.

The `extract-strings` target gathers source strings, then updates the `.pot` files for the SecureDrop server code and the desktop icons. (This step is enforced by CI, which will fail if you skip it.)

After running `make extract-strings`, 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. There is no need to review multiple languages' `.po` files because they are processed in the same way.

Commit and push these changes for review along with your source changes. See *Translation Responsibilities* for the complete workflow.

Verify translations

Content review is the responsibility of Localization Lab and their reviewers, in Weblate's review process. *Security review* of translations, primarily checking for malicious HTML or interpolation not caught by Weblate's checks, is the responsibility of SecureDrop maintainers, especially the localization manager and release manager for a given release.

SecureDrop web interfaces (`securedrop/securedrop`)

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/functional/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 (securedrop/desktop)

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.

If there have been new changes to the `securedrop/desktop` component, CI will fail on the pull request from Weblate with a warning to run `make update-desktop-files`. Run this command locally and push the changes to the `weblate-fpf/securedrop` fork, and CI should pass.

21.2.3 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](#).

21.3 Release Management

21.3.1 Two weeks before the release: string freeze

Note

If both a Localization Manager and a deputy are assigned for this release, consider pairing on this ceremony, both for knowledge-sharing and so that the intermediate pull requests can be reviewed and merged promptly.

When features for a new SecureDrop release are frozen, so are the source strings. The localization manager should apply any source-string suggestions that have been made by translators *before* the release branch has been cut, since only translations will be updated for release candidates and the final release.

Then, the localization manager for the release will:

- *Update Weblate screenshots* so translators can see new or modified source strings in context.
- 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://developers.securedrop.org/en/latest/translations.html> has instructions on how to get started.
- Update Localization Lab via the *SecureDrop Coordination* channel in the *TCU 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.

21.3.2 During QA

Review, merge, and backport `Translations update` from Weblate pull requests at most once before each release candidate is cut. Coordinate with the release manager.

21.3.3 Release day

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

- Review, merge, and backport the final `Translations update` from Weblate pull request.
- *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 [tracking spreadsheet](#) with supported languages’ current translation and review coverage. File a ticket for each new language due either (a) consideration for new support, (b) probation for dropping coverage, or (c) revocation of support.

21.3.4 Translator credits

Correct acknowledgment of translators’ contributions is important, so [Weblate makes it easy to list the translators](#) who have contributed in a specific period or at any point in the project’s history. For example, the Communications Manager for a release can generate a “Credits” report since the date of the last minor (X.Y.0) release and copy-paste it into the release notes.

21.4 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 [on Gitter](#). 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.

21.4.1 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://github.com/freedomofpress/securedrop-i18n> (make sure that the user has commit access)

21.4.2 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.

21.4.3 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 for end users (sources, journalists and administrators) 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.

Developer documentation is maintained separately via the SecureDrop developer documentation repository and published to <https://developers.securedrop.org>.

22.1 Documentation versions

SecureDrop maintains two versions of its end user documentation: `stable` (appropriate for end users) and `latest` (appropriate for developers). `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.

The developer documentation at <https://developers.securedrop.org/> is published only in a single version. When changes are merged into the main branch of the `securedrop-dev-docs` repository, a new version of the documentation is built immediately.

22.2 Updating Documentation

To get started editing the docs:

1. Clone the SecureDrop documentation repository.

For the end user documentation:

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

For the developer documentation:

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

2. Install Poetry, which is used to manage the Python dependencies of the project.

If you are not already using Poetry, follow the [installation guide](#) to set up Poetry on your operating system.

3. Install the Python dependencies of the project.

```
poetry install
```

We use the version of Python included with the most recent stable release of the Debian GNU/Linux distribution. The project configuration therefore specifies that this version of Python is required.

If you receive a warning that the required version of Python cannot be found, you may need to install it. If necessary, you can use `pyenv` to manage multiple versions of Python alongside each other.

4. 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> or <https://developers.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 appropriate repository.

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

22.4 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`).

22.5 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.

22.6 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 you are not also the release manager, check with them about any other pertinent release-specific instructions that should be included.
6. 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.

22.7 Style Guide

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

22.7.1 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
```

22.7.2 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
```

22.7.3 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`.

22.7.4 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

22.7.5 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.)

22.7.6 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`, and other Python tools to comprehensively test the SecureDrop server.

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.

24.1 Running the Application Tests

The tests are written to be run inside the development container:

```
make test
```

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

```
make test-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
```

24.1.1 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.

To run just these tests:

```
make test-pageslayout
```

24.2 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
```

(continues on next page)

(continued from previous page)

```
├── 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 a full SecureDrop server, whether on physical hardware or in staging VMs. Any changes to the Ansible configuration should have a corresponding test.

25.1 Installation

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

25.2 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.

25.3 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.

25.4 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.

25.5 Running the CI Staging Environment

The staging environment can also run via CI in Google Cloud (GCE). These tests are run every night or if a member of the `freedomofpress` Github Organization pushes to a branch that starts with `stg-`. Please ask in your PR if you'd like someone to run the tests for you.

The tests can also be run manually with a Google Cloud Platform account and Docker installed locally:

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. Some parameters are specific to FPF's GCE setup and may need adjusting if you are running elsewhere.

Then 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.

DEMO.SECUREDROP.ORG

There is a deployment of SecureDrop server running at <https://demo.securedrop.org/> that is very similar to `make dev`. Changes pushed to the `develop` branch automatically get deployed there.

26.1 How it works

Deployment is managed in Codefresh, which we use for other Kubernetes-related CI/CD; if you want to look at build logs or (re)start pipelines and don't have access, ask someone from the infra team. This is not a requirement to be able to update and deploy the demo; you just need to be able to push to the SecureDrop repo's `develop` branch.

26.2 Containers

There are two containers in the `securedrop` repo:

- `securedrop/dockerfiles/focal/python3/DemoDockerfile`
- `devops/demo/landing-page/Dockerfile`

If you want to try these locally, build them from the root of the SecureDrop repository, using `docker build -f FILE ..`. The “landing page” container is what you see at <https://demo.securedrop.org/> and the main container serves both <https://demo-source.securedrop.org/> and <https://demo-journalist.securedrop.org/>

26.3 Troubleshooting

The deployment is not highly available, and everything in Redis is thrown away when it restarts. When restarting, the source and journalist interfaces will both be unavailable for a few seconds and you'll see a 503.

If you need to cause a restart without pushing a new deployment, or roll back to an earlier version, ask the infra team. These can be done quickly but require direct Kubernetes access.

PACKAGE REPOSITORIES

SecureDrop publishes `.deb` and `.rpm` packages via apt and yum repositories, respectively.

Each package repository is maintained in a specific Git LFS repository that is published to [Cloudflare's R2 static hosting product](#). The Git repository contains the `.deb` and `.rpm` files, as well as the repository metadata. When a new commit is pushed, GitHub Actions (GHA) publishes the contents of the “public” folder to Cloudflare. We use `rclone` for this process to make it mostly vendor-neutral in case we need to switch providers in the future.

For historical reasons, the apt repositories are on `*.freedom.press` while the yum repositories are on `*.securedrop.org`.

There are three levels of package repositories, which correspond to different stages of the development process.

27.1 Test repositories

- apt: `apt-test.freedom.press`, via `securedrop-apt-test`
- yum: `yum-test.securedrop.org`, via `securedrop-yum-test`

Test repositories serve two primary functions. First, during the release process, release candidate packages are published here to enable developers to perform QA, including testing upgrades.

Second, nightly package builds are automatically pushed to test repositories by CI to enable developers to test integrated systems with code straight from *main*. Nightlies are stored in a separate component: “nightlies” in apt, “fXX-nightlies” in yum.

The signing key for these test repositories is lower-security and stored in GHA. Packages are automatically signed by a GHA workflow before being uploaded to Cloudflare.

27.2 QA repositories

- apt: `apt-qa.freedom.press`, via `securedrop-apt-prod's release branch`
- yum: `yum-qa.securedrop.org`, via `securedrop-yum-prod's release branch`

QA repositories are used as the final QA step before a new version is fully released. Developers upload candidate packages (using a non-release candidate version) to the *release* branch, and sign the repository using the high-security, offline, SecureDrop signing key.

Once the new packages have been QA'd and approved, the *release* branch is merged into *main*, which publishes the packages on the production repositories.

27.3 Production repositories

- apt: apt.freedom.press, via [securedrop-apt-prod's main branch](#)
- yum: yum.securedrop.org, via [securedrop-yum-prod's main branch](#)

Production repositories are used by real deployments of SecureDrop. SecureDrop server is configured to automatically fetch and install updates every 24 hours while SecureDrop Workstation requires a manual updater run.

This repository is signed using the high-security, offline, SecureDrop signing key.

27.4 How it works technically

This is an overview of the workflow, for step-by-step instructions, see the [server release management](#) and [workstation release management](#) docs.

1. New packages are committed to the relevant Git LFS repository:
 - Nightlies: by GitHub Actions
 - Test/QA/Prod: by a maintainer
2. If this is a yum repository containing RPMs, the individual RPMs are signed:
 - Test: by GitHub Actions, using the low-security test key
 - QA/Prod: by a maintainer, using the offline SecureDrop release key
3. Repository metadata is updated, including generation of `index.html`:
 - Test: by the `.github/workflows/sign.yml` GitHub Actions workflow
 - QA/Prod: by a maintainer, by running `./tools/publish` locally (this script is a misnomer as it doesn't actually publish the packages)
4. If this is an apt repository, the Release files are signed:
 - Test: by the `.github/workflows/sign.yml` GitHub Actions workflow
 - QA/Prod: by a maintainer, using the offline SecureDrop release key
5. Once pushed to the correct branch, a GitHub Actions workflow publishes the “public” (yum) or “repo/public” (apt) folder to Cloudflare R2.

We use rclone for this purpose, and in theory are entirely vendor neutral and can switch to any another S3-like service.

DEBIAN PACKAGES

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-*-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](#).

All SecureDrop Client component packages.

See [SecureDrop Workstation Release Management](#) for more information.

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](#).

28.1 About dbgsym packages

A [debug symbols package](#) is a Debian package that includes static debug symbols and allows for generating a backtrace or other diagnostic information in the event of a crash, for example [with gdb](#). These packages have a `-dbgsym.deb` suffix on Debian, and a `-dbgsym.ddeb` suffix (which we [rename](#) to to `-dbgsym.deb` for consistency) on Ubuntu. These packages are generated during the build process for components that include compiled binaries, such as for SecureDrop components with Rust or C code, and they do not make any other changes (i.e, they do not enable debug logs).

When building non-kernel production Debian packages, follow the relevant Release Management documentation to commit the dbgsym packages along with the regular production packages. These packages are automatically placed in a separate repo component (`main-debug`) in [securedrop-apt-prod](#).

Currently, we publish dbgsym packages for: `securedrop-app-code` (core), `securedrop-client`, `securedrop-proxy` (workstation). Kernel builds also generate dbgsym packages, but they are not published due to their prohibitive size.

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.

29.1 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.

29.2 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'
   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/
```

(continues on next page)

(continued from previous page)

```
↪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.

29.3 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.

29.4 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.

29.4.1 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.
```

29.4.2 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>
  <options>no_email_alert</options>
</rule>
```

(continues on next page)

(continued from previous page)

</group>

29.4.3 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.

29.4.4 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.

29.5 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.

31.1 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.

32.1 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. If applicable, consult the *specific considerations <tails_only_releases>* for releases that only modify code running on Tails-based workstations.
2. If this is a regular release, work with the localization manager for this release cycle to *review and merge translations*.
3. Copy a link of the latest release or release candidate from the [Tails apt repo](#) and include it in the issue. 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.
4. 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
```

5. For each release candidate, update the version files, code repo changelog, and Debian package changelog.
 1. If there have been new translations since the release branch or the last release candidate was cut, ask the localization manager to review them for merge into `develop` and then backport them into the release branch.
 2. 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.

3. 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.

4. Disregard the script-generated `.tag` file since this is only used when we need to sign the final release tag (see [Release Process](#) section).
 5. Sign the commit that was added by `update_version.sh`:

```
git commit --amend --gpg-sign
```

6. Push the branch:

```
git push origin release/<major>.<minor>.<patch>
```

7. Push the unsigned tag (only the final release tag needs to be signed, see [Release Process](#) section):

```
git push origin <major>.<minor>.<patch>-rcN
```

8. Once the tag is pushed, notify the Localization Manager so that the localization team can get started on translations.

6. Build Debian packages:
 - a. Check out the tag for the release candidate.
 - b. Build the packages with `make build-debs`

- c. Save and publish *build metadata*.
- d. Open a PR on `securedrop-apt-test` 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.

7. 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.
8. 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.
9. 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 `utils/backport.py`, which uses `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.*

32.2 Release Process

1. Prepare the final release commit and tag. Do not push the tag file.
2. Step through the signing ceremony for the tag file. If you do not have permissions to do so, coordinate with someone that does.
3. Once the tag is signed, append the detached signature to the unsigned tag:

```
cat 1.x.y.tag.sig >> 1.x.y.tag
```

4. Delete the original unsigned tag:

```
git tag -d 1.x.y
```

5. Make the signed tag:

```
git mktag < 1.x.y.tag > .git/refs/tags/1.x.y
```

6. Verify the signed tag:

```
git tag -v 1.x.y
```

7. Push the signed tag:

```
git push origin 1.x.y
```

8. 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
```

9. Build Debian packages:
 - a. Verify and check out the signed tag for the release.
 - b. Build the packages with `make build-debs`.
 - c. Save and publish *build metadata*.
10. In a clone of the private `securedrop-apt-prod` repository, create a branch from `main` called `release`.
11. In your local branch, commit the built packages to the `core/focal` directory. Include all `.deb` packages, including `-dbgsym` packages. `-dbgsym` packages belong in the `main-debug` component repo. See *Notes on dbgsym-packages* for more information.
12. Run the `tools/publish` script. This will create the Release file.
13. Commit the changes made by the `tools/publish` script.
14. 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.

15. Create a [draft PR](#) from `release` into `main`. Make sure to include a link to the build logs in the PR description.
16. 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.
17. Coordinate with one or more team members to [confirm a successful clean install in production VMs](#) using the packages on `apt-qa.freedom.press`.
18. If no issues are discovered in final QA, promote the packaging PR out of draft mode.
19. A reviewer must merge the packaging PR. This will publish the packages on `apt.freedom.press`.
20. The reviewer must delete the `release` branch so that it can be re-created during the next release.
21. 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.

32.3 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](#) for any new user issues related to the release.

32.4 Releases that only modify code on Tails workstations

On occasion, a point release may only modify code that is deployed to Tails-based *Admin Workstations* and *Journalist Workstations*. Even in those cases, it is generally preferred to issue a release that also updates the server packages (bumping the version number):

- This ensures that users attentive to version numbers are not confused by the discrepancy between the version shown on their workstations compared with the version number shown in other parts of the SecureDrop user interface.
- It also mitigates the risk of any unexpected side effects. Notably, our uptime monitoring of known SecureDrop instances checks for differences between the version number returned by a server's metadata endpoint, and the latest GitHub release object.

If, because of time sensitivity and team availability, a release manager decides to proceed with a workstation-only release, they should observe the following:

- As with regular releases, create tags for any release candidates and test the expected behavior on *Admin* and *Journalist Workstations* as appropriate.
- Coordinate with the infrastructure team to ensure that uptime monitoring will not alert on the discrepancy between server-side version numbers and the latest release object on GitHub.
- After the release passes QA, push a signed tag. This will enable the graphical updater on Tails workstations to detect the new release.
- Follow the standard release communications process, including publication of a release object on GitHub. Make note of the fact that this is a workstation-only release (see the [SecureDrop 2.6.1 release communications](#) as an example).

BUILD METADATA

When we build packages to ship to users, we save and publish build metadata. Currently this happens in the form of build logs and `.buildinfo` files, both of which are published to the [build-logs](#) repository.

33.1 Build logs

When you build a package for release, you should save your terminal output, including:

- Checking out the build tag and verifying that it is signed with the release key
- `make build-debs` (or equivalent) output
- SHA256 checksums of the built packages

These should be committed into the corresponding folder in the build-logs repository.

The goal with these build logs is to have a clear record of what happened during the build process for the purpose of retrospectives. This can help us determine if mistakes are made during the build (since some of the process is manual) and for incident response.

33.2 buildinfo

`.buildinfo` files record information about the environment used to build the package so that an external user can recreate that environment and reproduce the package. See the [Debian documentation](#) for more details.

When produced by a build, these `.buildinfo` files should be committed into the `buildinfo/` folder. As these files also contain SHA256 checksums of the packages, checksums can be omitted from the build log.

These are not yet [generated for RPM packages](#).

LINUX KERNEL MAINTENANCE

We build and publish our own Linux kernels with additional [grsecurity hardening patches](#). This process is automated in the [kernel-builder](#) repository.

1. Follow the instructions in [kernel-builder](#) for building and uploading new kernel packages.
2. Once the new packages have been reviewed and merged in the [securedrop-apt-test](#) repository, they will be [automatically tested](#) on the hardware we maintain in our kernel test farm.
3. Wait for sdcibot to file a New Linux kernel ticket ([example](#)) in the [securedrop](#) repository with its test results.
4. The packages can then be promoted to [securedrop-apt-prod](#).

34.1 Testing a new kernel manually

These are the steps sdcibot performs in its [automatic testing](#) of new kernel packages on all of our [recommended hardware](#):

1. Install the new kernel packages on your *Monitor Server* using unattended-upgrades, e.g. `sudo apt update && sudo unattended-upgrades --debug` or wait for the automatic nightly upgrade.
2. Reboot. Verify with `uname -r` that you are using the new kernel.
3. If it doesn't boot, see the [Troubleshooting Kernel Updates](#) documentation.
4. Install the `paxtest` package, run with `sudo paxtest blackhat`, and verify it doesn't return any new errors nor warnings.
5. Install [spectre-meltdown-checker](#) and the `binutils` package, run with `sudo ./meltdown-checker`, and verify it doesn't return any errors nor warnings.
6. Upgrade your *Application Server* to the new kernel and reboot.

You may optionally also:

7. Run basic smoke tests of SecureDrop by verifying you can send a submission and a journalist can reply.

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.

35.1 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-apt-test](#) repository. Within 15 minutes they should be available for download via `apt-test.freedom.press`.

35.2 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.

35.3 Promoting

To promote a Tor release to production, copy the `*.deb` files over to the [securedrop-apt-prod](#) repository and follow those instructions.

SETTING UP THE SECUREDROP WORKSTATION

The SecureDrop Workstation based on [Qubes OS](#) is a project currently in the beta stages of software development which aims to improve journalists' experience working with SecureDrop while retaining the current security and privacy features SecureDrop provides.

Installing the project requires an up-to-date Qubes 4.2 installation running on a machine with at least 16GB of RAM (32 GB recommended).

The project is currently in open beta. See our [blog post](#) for more information. If you are interested in using SecureDrop Workstation, please reach out to us via the [support portal](#). Documentation for end users is being developed [here](#). The instructions below are intended for developers.

36.1 Install Qubes

Before trying to use this project, install [Qubes 4.2.1](#) on your development machine. Accept the default VM configuration during the install process.

After installing Qubes, you must update both dom0 and the base templates to include the latest versions of apt packages. Open a terminal in dom0 by clicking on the Qubes menu top-right of the screen and left-clicking on Terminal Emulator and run:

```
sudo qubes-dom0-update
```

After dom0 updates complete, reboot your computer to ensure the updates have been properly applied. Finally, update all existing TemplateVMs:

```
qubes-update-gui
```

Select all VMs marked as **updates available**, then click **Next**. Once all updates have been applied, you're ready to proceed. Choose the environment that you wish to set up and then follow the applicable instructions:

- The staging environment uses the `yum-test.securedrop.org` and `apt-test.freedom.press` repositories, and is configured to use the `main` component for apt packages. It will typically install the most recent release candidate packages (which could be more recent than the production packages if a release is underway).
- The development environment uses the `yum-test.securedrop.org` and `apt-test.freedom.press` repositories, and is configured to use the `nightly` and `main` components for apt packages. This means it will install nightly packages, unless a package is only available in `main`, or a version with a higher version number has been published there.

This configuration does not alter your management settings on your laptop to prevent suspension to disk (a security measure for production environments, which the staging environment preserves to be more faithful to prod-like settings).

- The production environment uses `yum.securedrop.org` and `apt.freedom.press` repositories, verified using the production signing key. Its setup is not covered below; see our [production install docs](#) for details.

36.2 Development Environment

36.2.1 Download, Configure, Copy to dom0

This repository contains the specification for an RPM package, which contains the provisioning logic. By following the instructions below, you will build this RPM package locally from a `git` checkout in your development VM, copy it to `dom0`, install it, and run the provisioning code to set up a SecureDrop Workstation in the development environment configuration.

Decide on a VM to use for development. We recommend creating a standalone VM called `sd-dev` by following [these instructions](#).

Clone the *securedrop-workstation* repo to your preferred location on that VM.

Qubes provisioning is handled by Salt on `dom0`, so this project must be copied there from your development VM.

Note

Understand that [copying data to dom0](#) goes against the grain of the Qubes security philosophy, and should only be done with trusted code and for very specific purposes, such as Qubes-related development tasks. Still, be aware of the risks, especially if you rely on your Qubes installation for other sensitive work.

That process is a little tricky, but here's one way to do it: assuming this code is checked out in your `sd-dev` VM at `/home/user/projects/securedrop-workstation`, run the following in `dom0`:

```
qvm-run --pass-io sd-dev 'tar -c -C /home/user/projects/ securedrop-workstation' | tar -xvf -
```

(Be sure to include the space after `/home/user/projects/`.) After that initial manual step, the code in your development VM may be copied into place on `dom0` by setting the `SECUREDROP_DEV_VM` and `SECUREDROP_DEV_DIR` environmental variables to reflect the VM and directory to which you've cloned this repo, and running `make clone` from the root of the project on `dom0`:

```
export SECUREDROP_DEV_VM=sd-dev      # set to your dev VM
export SECUREDROP_DEV_DIR=/home/user/projects/securedrop-workstation # set to your
working directory
cd ~/securedrop-workstation/
make clone      # build RPM package and copy repo to dom0
```

NOTE: The destination directory on `dom0` is not customizable; it must be `securedrop-workstation` in your home directory.

If you plan to work on the [SecureDrop Client](#) code, also run this command in `dom0`:

```
qvm-tags sd-dev add sd-client
```

Doing so will permit the `sd-dev` AppVM to make RPC calls with the same privileges as the `sd-app` AppVM.

36.2.2 Run Development SecureDrop Server

Here, you will setup a development version of the SecureDrop server to which your workstation will connect, though the Tor Network. Alternatively, you can setup *virtualized staging environments on Qubes OS*, which is slightly more involved.

1. Setup a *SecureDrop (server) development environment* on Qubes.
2. Start the securedrop server in sd-dev qube with use `make dev-tor`

Note

You will need to run this step every time that you want to login on SecureDrop client, in the Workstation.

The specific *.onion* address and its discovery key will be stored on your system. If you ever need to change it, you can run *make clean*, which will also delete any other server state.

36.2.3 Configure the Workstation

In the output of the *make dev-tor* command ran in the previous section, there should be a section that looks like this:

```
{
  "submission_key_fpr": "65A1B5FF195B56353CC63DFFCC40EF1228271441",
  "hidserv": {
    "hostname": "jpweqok4r43xp4si5pattodglw2btdqlpz2utvn4mkwnx2iwbmp4v2id.onion",
    "key": "DAZHRYKWHQCIRUMEVIRSOUZA4MKU4C7WPDWLIVB3TMZWZH2V5MA"
  },
  "environment": "prod",
  "vm_sizes": {
    "sd_app": 10,
    "sd_log": 5
  }
}
```

Save this text in the file `securedrop-workstation/config.json`.

Next, set the default encryption key (for development purposes only):

```
cd securedrop-workstation
cp sd-journalist.sec.example sd-journalist.sec
```

Then, in `dom0`, clone the workstation again, to obtain these new files:

```
cd ~/securedrop-workstation/
make clone
```

36.2.4 Provision the VMs

Once the configuration is done and this directory is copied to `dom0`, you must update existing Qubes templates and use *make* to handle all provisioning and configuration by your unprivileged user. Before you do so, you may wish to increase the scrollbar in the `dom0` terminal from 1000 (the default) to 100000 or unlimited, to ensure you can review any errors in the verbose output.

Then run the following command to set up a development environment:

```
make dev
```

Note that this target automatically sets the `environment` variable in `config.json` to `dev`, regardless of its current value, before provisioning. It identifies the latest RPM you have built (using `scripts/prep-dev`), installs it, and runs the `sdw-admin --apply` command to provision the SecureDrop Workstation.

The build process takes quite a while. You will be presented with a dialog asking how to connect to Tor: you should be able to select the default option and continue. If you want to refer back to the provisioning log for a given VM, go to `/var/log/qubes/mgmt-<vm name>.log` in `dom0`. You can also monitor logs as they're being written via `journalctl -ef`. This will display logs across the entire system so it can be noisy. It's best used when you know what to look for, at least somewhat, or if you're provisioning one VM at a time.

When the installation process completes, a number of new VMs will be available on your machine, all prefixed with `sd-`.

36.2.5 Editing the configuration

When developing on the Workstation, make sure to edit files in `sd-dev`, then copy them to `dom0` via `make clone` && `make dev` to reinstall them. Any changes that you make to the `~/securedrop-workstation` folder in `dom0` will be overwritten during `make clone`. Similarly, any changes you make to e.g. `/srv/salt/` in `dom0` will be overwritten by `make dev`.

36.3 Staging Environment

36.3.1 Update `dom0`, `fedora-40-xfce`, `whonix-gateway-17` and `whonix-workstation-17` templates

Updates to these VMs will be performed by the installer and updater, but updating them prior to install makes it easier to debug any errors.

Before proceeding to updates, we must ensure that `sys-whonix` can bootstrap to the Tor network. In the Qubes menu, navigate to `sys-whonix` and click on `Anon Connection Wizard` and click `Next` and ensure the Tor Bootstrap process completes successfully.

In the Qubes Menu, select the cog icon to access the Settings submenu, navigate to `Qubes Tools` and click on `Qubes Update`. In the updater, select all VMs in the list, then click `Next` and wait for updates to complete.

36.3.2 Choose your installation method

You can install the staging environment in two ways:

- If you have an up-to-date clone of this repo with a valid configuration in `dom0`, you can use the `make staging` target to provision a staging environment. Prior to provisioning, `make staging` will set your `config.json` environment to `staging`.
- If you want to download a specific version of the RPM, and follow a verification procedure similar to that used in a production install, follow the process in the following sections.

36.3.3 Download and install `securedrop-workstation-dom0-config` package

Since `dom0` does not have network access, we will need to download the `securedrop-workstation-dom0-config` package in a Fedora-based VM. We can use the default Qubes-provisioned `work` VM. If you perform these changes in the `work` VM or another AppVM, they won't persist across reboots (recommended).

In a terminal in `work`, run the following commands:

1. Import the test signing key:

```
wget https://raw.githubusercontent.com/freedomofpress/securedrop-workstation/master/sd-
↳ workstation/apt-test-pubkey.asc
sudo rpmkeys --import apt-test-pubkey.asc
```

2. Configure the test repository

Populate `/etc/yum.repos.d/securedrop-temp.repo` with the following contents:

```
[securedrop-workstation-temporary]
enabled=1
baseurl=https://yum-test.securedrop.org/workstation/dom0/f37
name=SecureDrop Workstation Qubes initial install bootstrap
```

3. Download the RPM package

```
dnf download securedrop-workstation-dom0-config
```

The RPM file will be downloaded to your current working directory.

4. Verify RPM package signature

```
rpm -Kv securedrop-workstation-dom0-config-x.y.z-1.fc37.noarch.rpm
```

The output should match the following, and return OK for all lines as follows:

```
securedrop-workstation-dom0-config-x.y.z-1.fc37.noarch.rpm:
  Header V4 RSA/SHA256 Signature, key ID 2211b03c: OK
  Header SHA1 digest: OK
  V4 RSA/SHA256 Signature, key ID 2211b03c: OK
  MD5 digest: OK
```

5. Transfer and install RPM package in dom0

Note

Understand that copying data to dom0 goes against the grain of the Qubes security philosophy, and should only be done with trusted code and for very specific purposes, such as Qubes-related development tasks. Still, be aware of the risks, especially if you rely on your Qubes installation for other sensitive work.

In dom0, run the following commands (changing the version number to its current value):

```
qvm-run --pass-io work 'cat /home/user/securedrop-workstation-dom0-config-x.y.z-1.fc37.
↳ noarch.rpm' > securedrop-workstation.rpm
sudo dnf install securedrop-workstation.rpm
```

The provisioning scripts and tools should now be in place, and you can proceed to the workstation configuration step.

36.3.4 Configure the Workstation

Your workstation configuration will reside in `/usr/share/securedrop-workstation-dom0-config/` and will contain configuration information specific to your SecureDrop instance:

1. Populate `config.json` with your instance-specific variables. Set `environment` to `staging`
2. Move your submission private key to `sd-journalist.sec`

36.3.5 Provision the VMs

In a terminal in dom0, run the following commands:

```
sdw-admin --apply
```

SECUREDROP WORKSTATION DEVELOPMENT

This project's development requires different workflows for working on provisioning components and working on submission-handling scripts.

For developing salt states and other provisioning components, work is done in a development VM and changes are made to individual state and top files there. In the `dom0` copy of this project:

- `make clone` is used to build a new version of the RPM and copy the contents of your working directory (including the RPM) from your development VM to `dom0`
- `make dev` installs the latest locally present RPM and performs the full installation.

Note that `make clone` requires two environment variables to be set: `SECUREDROP_DEV_VM` must be set to the name of the VM where you've been working on the code, the `SECUREDROP_DEV_DIR` should be set to the directory where the code is checked out on your development VM.

For work on components such as the SecureDrop Client, see their individual readmes in the [securedrop-client](#) repository:

37.1 Testing

Tests should cover two broad domains. First, we should assert that all the expected VMs exist and are configured as we expect (with the correct NetVM, with the expected files in the correct place). Second, we should end-to-end test the document handling scripts, asserting that files present in the `sd-proxy` VM correctly make their way to the `sd-app` AppVM, and are opened correctly in disposable VMs.

37.1.1 Configuration Tests

These tests assert that expected scripts and configuration files are in the correct places across the VMs. These tests can be found in the `tests/` directory. They can be run from the project's root directory on `dom0` with:

```
make test
```

Note that since tests confirm the states of provisioned VMs, they should be run *after* all the VMs have been built with `make dev`.

Individual tests can be run with `make <test-name>`, where `test-name` is one of `test-base`, `test-app`, `test-proxy`, `test-whonix` or `test-gpg`.

Be aware that running tests *will* power down running SecureDrop VMs, and may result in *data loss*. Only run tests in a development / testing environment.

37.2 Automatic updates

Double-clicking the “SecureDrop” desktop icon will launch a preflight updater that applies any necessary updates to VMs, and may prompt a reboot. In a development environment, this will install the latest nightly packages (unless a package with a higher version number is available on `apt-test` in `main`), and the latest RPM published to `yum-test`.

37.3 Manually updating dom0 code

To update code in `dom0` manually, e.g., to a specific branch or tag of this repository, use the `sd-dev` AppVM that was created during the install. For example, to build a specific tag, from your checkout directory, run the following commands (replace `<tag>` with the tag of the release you are working with):

```
git fetch --tags
git tag -v <tag>
git checkout <tag>
```

In `dom0`:

```
make clone
make dev
```

The `make clone` command will build a new version of the RPM package that contains the provisioning logic in your development VM (e.g., `sd-dev`) and copy it to `dom0`.

37.4 Building workstation Debian packages

Debian packages for the SecureDrop Workstation components are maintained in a separate repository: <https://github.com/freedomofpress/securedrop-client/>

37.5 Building workstation RPM packages

```
make build-rpm
```

The build runs in a container and requires either Docker or Podman to be installed.

SECUREDROP WORKSTATION RELEASE MANAGEMENT

38.1 Communications Process

As with SecureDrop server releases, the release manager should work with the Communications Manager assigned for the release to prepare announcements that will be shared on the SecureDrop blog and on social media after the release is live.

Typically, the kick-off of the QA period is a good time to begin the process. Please note that review by FPF's editorial team is required and should only be skipped in case of urgent release-specific considerations, e.g., to get a hotfix release out as quickly as possible.

Once the release is live:

1. Make sure that release notes are written and posted on the SecureDrop blog.
2. Make sure that the release is announced on social media.
3. If the release warrants announcements beyond that (e.g., via Signal group), make them now.

38.2 Release an RPM package

38.2.1 Release `securedrop-workstation-dom0-config`

1. Verify the tag of the project you wish to build: `git tag -v VERSION` and ensure the tag is signed with the official release key.
2. `git checkout VERSION`
3. Now you are ready to build. Build RPMs following the documentation in an environment sufficient for building production artifacts. For `securedrop-workstation` you run `make build-rpm` to build the RPM.
4. `sha256sum` the built RPM (and store hash in the build logs/commit message).
5. Commit the (unsigned) version of this RPM to the `release` branch in the `securedrop-yum-prod` repository.
6. Copy the RPM to the signing environment.
7. Verify integrity of RPM prior to signing (use `sha256sums` to compare). **Note for reviewers:** Using `rpm --delsign` on a signed artifact (for example, a release candidate) in order to verify the checksum of the unsigned `.rpm` file must be done in the same type of build environment (Linux distribution and `rpm` version) as the `.rpm` was built in, or the checksums may not match.
8. Sign RPM in place (see Signing section below).
9. Move the signed RPM back to the environment for committing to the `lfs` repository.
10. Save and publish *build metadata*.
11. Commit the RPM in a second commit on the `release` branch in `securedrop-yum-prod`.

12. Run the `/tools/publish` script to update repository metadata and commit the result.
13. Create a PR to merge release into main. At this point, the package will be available on yum-qa.securedrop.org.
14. Once the PR is merged, the changes will be available on yum.securedrop.org.

38.2.2 Signing procedures

38.2.3 Sign the tag with the SecureDrop release key

1. If the tag does not already exist, create a new annotated and unsigned tag: `git tag -a VERSION`.
2. Output the tag to a file: `git cat-file tag VERSION > VERSION.tag`.
3. Copy the tag file into your signing environment and then verify the tag commit hash.
4. Sign the tag with the SecureDrop release key: `gpg --armor --detach-sign VERSION.tag`.
5. Append ASCII-armored signature to tag file (ensure there are no blank lines): `cat VERSION.tag.sig >> VERSION.tag`.
6. Move tag file with signature appended back to the release environment.
7. Delete old unsigned tag: `git tag -d VERSION`.
8. Create new signed tag: `git mktag < VERSION.tag > .git/refs/tags/VERSION`.
9. Verify the tag's signature: `git tag -v VERSION`.
10. Push the tag to the shared remote: `git push origin VERSION`.

38.2.4 Sign the RPM package

The entire RPM must be signed. This process also requires a Fedora machine/VM on which the GPG signing key (either in GPG keyring or in qubes-split-gpg) is setup. You will need to add the public key to RPM for verification (see below).

`rpm -Kv` indicates if digests and sigs are OK. Before signature it should not return signature, and `rpm -qi <file>`. `rpm` will indicate an empty Signature field. Set up your environment (for prod you can use the `~/rpmmacros` example file at the bottom of this section):

```
sudo dnf install rpm-build rpm-sign # install required packages
echo "vault" | sudo tee /rw/config/gpg-split-domain # edit 'vault' as required
cat << EOF > ~/.rpmmacros
%_signature gpg
%_gpg_name <gpg_key_id>
%__gpg /usr/bin/qubes-gpg-client-wrapper
%__gpg_sign_cmd %{__gpg} --no-verbose -u %{_gpg_name} --detach-sign %{__plaintext_
↪filename} --output %{__signature_filename}
EOF
```

Now we'll sign the RPM:

```
rpm --resign <name>.rpm # --addsign would allow us to apply multiple signatures to the
↪RPM
rpm -qi <name>.rpm # should now show that the file is signed
rpm -Kv <name>.rpm # should contain NOKEY errors in the lines containing Signature
# This is because the (public) key of the RPM signing key is not present,
# and must be added to the RPM client config to verify the signature:
sudo rpm --import <publicKey>.asc
rpm -Kv <name>.rpm # Signature lines will now contain OK instead of NOKEY
```

You can then proceed with distributing the package, via the “test” or “prod” repo, as appropriate.

38.3 Post-Release tasks

1. If you’ve not done so already as part of the release, ensure release communications are published.
2. Run the updater on a production setup once packages are live, and conduct a smoketest (successful updater run, and basic functionality if updating client packages).
3. Backport changelog commit(s) with `git cherry-pick -x` from the release branch into the main development branch, and sign the commit(s). In a separate commit, run the `update_version.sh` script to bump the version on main to the next minor version’s rc1. Open a PR with these commits; this PR can close the release tracking issue.

SECUREDROP INBOX RELEASE MANAGEMENT

39.1 Development timeline

The SecureDrop Team works on a 2-week sprint cycle. The schedule for a sprint that includes a release should look like:

- Week 1: Development work
- Week 2: Release week
 - Monday: Major features list finalized by standup; start Comms draft and send to FPF Editorial by the end of the day.
 - Tuesday: All remaining PRs that will be included in the release are identified at standup. By the end of the day, bump the version to an RC to begin QA. Merges to `main` are temporarily blocked until the release is complete.
 - Wednesday: Finish QA by standup and discuss any potential blockers. If there are no blockers, proceed with the full release, otherwise work on fixes/reverts and repeat the QA cycle.
 - Thursday: backup release day in case we needed extra time because of extra RCs.

If a release takes longer than this, then it becomes an all-hands-on-deck situation to resolve whatever is necessary. That could also include backing out the problematic change and retrying it in the next sprint.

39.2 Workflow

39.2.1 Step 0: Tracking and planning

Create a tracking issue titled `Release SecureDrop Inbox <version>`. It should contain assigned roles and estimated timelines. Pin the issue for ease of access and visibility.

Review the in-flight issues and PRs, tagging them with the appropriate milestone. Ensure you explicitly communicate with other developers if you are expecting to include their work in the upcoming release.

39.2.2 Step 1: Create a release candidate (RC) tag

1. Increment the version using `./update_version.sh <version>-rc1`, and update `changelog.md`, opening a PR with your changes.
2. Once that PR is merged, create a tag with the version, e.g. `git tag -s <version>-rc1` and push it. The tag should be signed with your local developer key, not the SecureDrop Release Key.

39.2.3 Step 2: Build and deploy the package to apt-test

Note

This step will largely be automated in the future.

1. Clone `securedrop-client` and `securedrop-builder`.

```
git clone git@github.com:freedomofpress/securedrop-client.git
git clone git@github.com:freedomofpress/securedrop-builder.git
```

2. Check out the newly pushed tag and then build the packages.

```
cd securedrop-client
git checkout <version>-rc1
make build-debs
```

3. Save and publish *build metadata*.
4. Open a PR to <https://github.com/freedomofpress/securedrop-apt-test> with the packages you want to deploy. Once merged, the packages will be deployed to <https://apt-test.freedom.press>.

39.2.4 Step 3: Begin QA

QA testers should begin working through the test plan, identifying any issues they run into. File issues for any unexpected behavior, and determine if they should be fixed or can be deferred to a future release.

If fixes are needed, create and merge PRs into `main`, issue another RC and repeat the QA process.

Once no fixes are needed, then you can begin preparing the production release.

39.2.5 Step 4: Create a production release tag

1. Submit a PR that updates the changelog and version. Remove any references to the RC versions from the changelogs.
2. Generate a release tag named `<version>` (same as the previous tags, without the `-rcN` part).
3. *Sign the tag with the SecureDrop release key* or ask another maintainer to do this and push the signed tag
4. Ask another maintainer to verify the tag signature and that it points to the correct commit.

39.2.6 Step 5: Build and deploy the packages to apt-prod

Note

This step will largely be automated in the future.

1. Clone `securedrop-client` and `securedrop-builder`.

```
git clone git@github.com:freedomofpress/securedrop-client.git
git clone git@github.com:freedomofpress/securedrop-builder.git
```

2. Check out the newly pushed tag and then build the packages.

```
cd securedrop-client
git tag -v <version> # Signed by SecureDrop Release Key
git checkout <major>.<minor>.<patch>
make build-debs
```

3. Save and publish *build metadata*.
4. Add your packages to a new branch called **release** in <https://github.com/freedomofpress/securedrop-apt-prod>. Include all built .deb packages, including -dbgSYM packages. (-dbgSYM packages will be kept in the main-debug component. See *Notes on dbgSYM-packages* for more information.)
5. Update the apt repo distribution files by running `./tools/publish` and push those changes to the **release** branch as well.
6. *Regenerate and sign the apt release file* or ask another maintainer to do this. The packages will now be installable from <https://apt-qa.freedom.press>.
7. Open a PR to merge the **release** branch into **main**.
8. Another maintainer should also build the packages (following the same steps as earlier) and verify their newly built packages are **bit-for-bit identical** to those pushed to apt-qa.

39.2.7 Step 6: Perform the apt-qa preflight check

First, provision a production workstation from the most recently-released `securedrop-workstation-dom0-config` production package. Ensure your machine has been updated (either via Qubes native updater or SDW GUI updater).

At minimum, perform the full test. Additional QAers may perform smoketest to save time if there is already full test coverage.

Note

These instructions are out of date.

Full test (includes updater)

1. As root, edit `/srv/salt/sd-default-config.yml` so that the `prod apt_repo_url` points to <https://apt-qa.freedom.press>.
2. Run the SDW GUI updater. To force an updater run, invoke the updater via `/opt/securedrop/launcher/sdw-launcher.py --skip-delta 0`.
3. Start the Client application, and observe the updated version string, indicating the required packages were installed. Perform testing according to the test plan.

Smoketest (no updater run)

1. Start the Template VMs.
2. In each template VM, edit `/etc/apt/sources.list.d/securedrop_workstation.list` file to point to <https://apt-qa.freedom.press>.
3. Update the package system and install the new packages via `apt update && apt upgrade -y`.
4. Verify that the updated packages were installed in the templates. Shut down template VMs and all VMs associated with SecureDrop Workstation.
5. Start the Client application and perform testing according to test plan.

39.2.8 Step 7: Deploy to apt-prod

1. In `securedrop-apt-prod`, merge the PR into `main` to deploy the new version to <https://apt.freedom.press>.
2. Once you see the packages land on <https://apt.freedom.press>, run the updater to install it in a production environment and ensure that it works as expected.

39.2.9 Step 8: Communications

1. Publish a blog post for the release on securedrop.org and promote it on our social media accounts.
2. If appropriate, send a blast out to Signal groups announcing the new release and any notable features or bug fixes.

39.2.10 Step 9: Cleanup

1. In the `securedrop-client` repository, update the version to `<major>.<minor+1>.0-alpha` to prepare for the next development cycle.
2. Close the release issue and milestone.
3. Celebrate!

39.3 Using a release branch

Historically each release was prepared on a separate `release/<version>` branch, with changes needing to be back-ported into the release branch, and then commits forward-ported into `main` post-release. In mid-2026 we moved away from this being the default workflow, however it remains an option for when a release takes longer than normal.

This can be implemented at the beginning of the release process, if we know that the release will take longer than normal and don't want to keep `main` frozen for that entire time, or if it ends up taking longer than anticipated.

To use a release branch, create a new branch from wherever `main` is. All future version bump, changelog updates and tags will take place on this branch. Any changes in `main` that should go into the release will need to manually be cherry-picked into the release branch.

Once the release is complete, review the release branch for any changes that were solely made in it, which should normally be just version bumps and the changelog, and cherry-pick those back into `main`.

39.4 Signing procedures

39.4.1 Sign the tag with the SecureDrop release key

1. If the tag does not already exist, create a new annotated and unsigned tag: `git tag -a VERSION`.
2. Output the tag to a file: `git cat-file tag VERSION > VERSION.tag`.
3. Copy the tag file into your signing environment and then verify the tag commit hash.
4. Sign the tag with the SecureDrop release key: `gpg --armor --detach-sign VERSION.tag`.
5. Append ASCII-armored signature to tag file (ensure there are no blank lines): `cat VERSION.tag.sig >> VERSION.tag`.
6. Move tag file with signature appended back to the release environment.
7. Delete old unsigned tag: `git tag -d VERSION`.
8. Create new signed tag: `git mktag < VERSION.tag > .git/refs/tags/VERSION`.
9. Verify the tag's signature: `git tag -v VERSION`.
10. Push the tag to the shared remote: `git push origin VERSION`.

39.4.2 Regenerate and sign the apt release file

1. From the release branch containing the new package, update the apt repository distribution files.

```
git clone https://github.com/freedomofpress/securedrop-apt-prod
cd securedrop-apt-prod
git checkout -b release
./tools/publish
```

2. Copy the regenerated file called `Release` into your signing environment and then verify the hash to ensure the file transfer was successful.
3. Sign the `Release` file with the SecureDrop release key.

```
gpg --armor --detach-sign Release
```

4. Copy the `Release.gpg` file into your release environment and move it to `repo/public/dists/<debian-codename>/` on your release branch.
5. Verify that the release file was signed with the production key.

```
gpg --verify ./repo/public/dists/<debian-codename>/Release{.gpg,}
```


SECUREDROP CLIENT DEVELOPMENT

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.

Note

We are actively rewriting the SecureDrop Client from a PyQt-based application to a modern Electron-based application. The new implementation can be found in the [app directory](#) of the `securedrop-client` repository. This rewrite aims to provide better performance, maintainability, and user experience while preserving all security properties.

See *SecureDrop App Development* for development documentation of the new implementation.

The source code, and related issues are [hosted on GitHub](#).

40.1 Developer Setup

You may find developer setup instructions in the [SecureDrop Client README](#).

40.2 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. Connection information for standups is published periodically on the Gitter channel. All are welcome to contribute.

Otherwise, read on.

40.3 Client Architecture

The SecureDrop client is a PyQt application. It's written using Python 3.11 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).

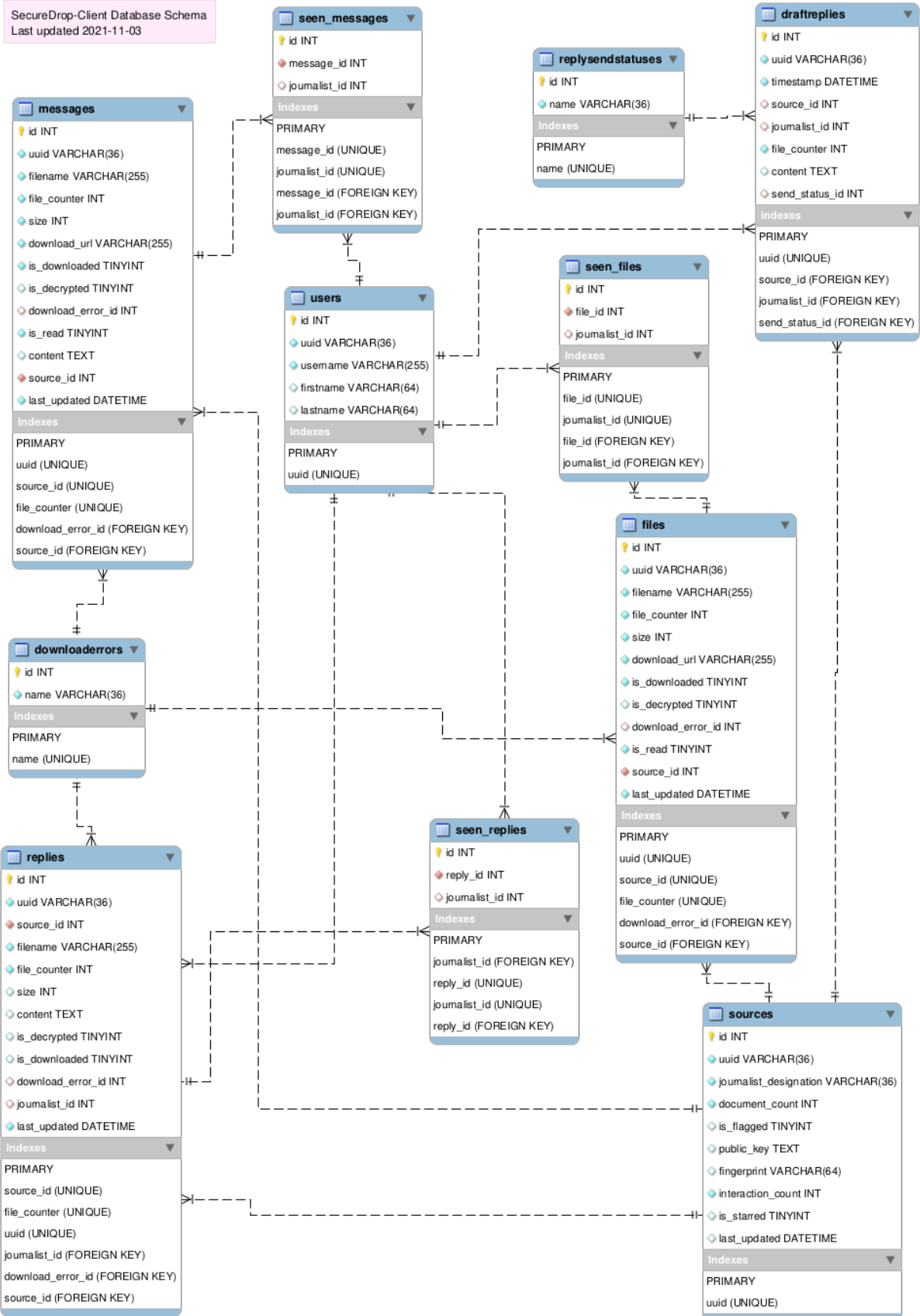
We try very hard to keep the application logic and UI code cleanly separated. Furthermore, we try equally hard to ensure the main GUI code always remains unblocked. For instance look at how the `APICallRunner` is used in `logic.py` to make unblocked network calls to the remote API.

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.

We make use of *ruff*, *mypy*, and other linting tools to standardize code style. Please run *make lint* locally from both the project root and the client directories before submitting a PR.

40.4 Client Database Structure

For a better understanding of the SecureDrop Client application architecture, a high-level view of its database structure has been provided:



40.5 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.

40.6 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!

SECUREDROP APP DEVELOPMENT

The SecureDrop App is a pre-alpha re-implementation of the SecureDrop Client for Qubes, built with modern web technologies.

The source code and issues are hosted on the [securedrop-client repository](#) in the `app` directory.

Warning

This project is not yet ready for production use. Please see the *SecureDrop Client Development* documentation for the implementation currently in use in production.

41.1 Repository Overview

The SecureDrop App is an Electron-based desktop application built with React and TypeScript. It provides a secure interface for SecureDrop journalists to communicate with sources, featuring a modern UI with Ant Design components and Tailwind CSS.

41.1.1 Requirements

- Node.js v24
- Rust toolchain
- pnpm package manager
- Additional system packages for building the Rust proxy

Linux is strongly recommended, though it should be possible to also develop the App using macOS.

41.2 Development Setup

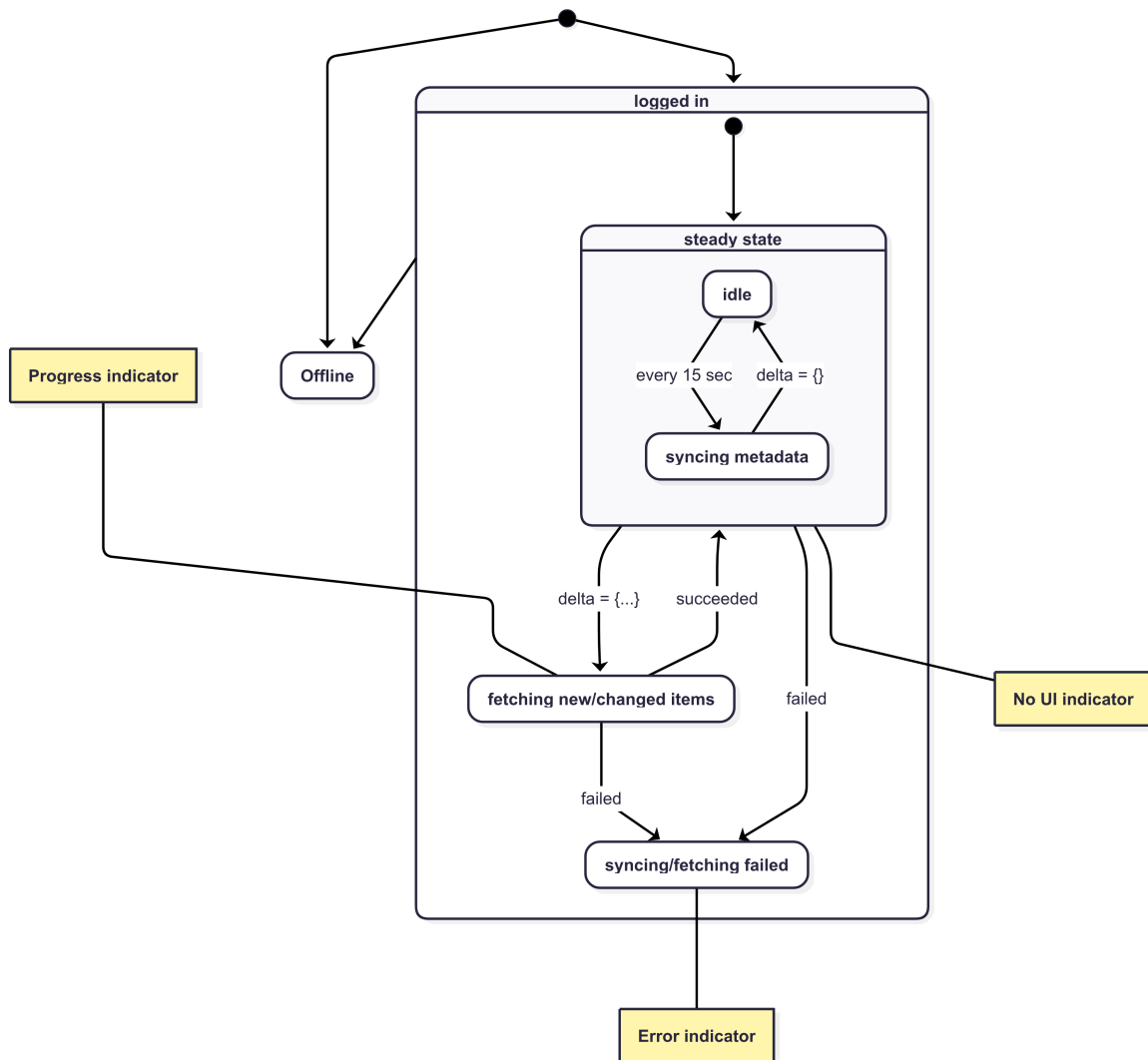
For complete setup instructions, development workflows, and available scripts, please refer to the [app directory README](#).

41.3 Synchronization Behavior

Note

The synchronization behavior described below represents the specification being implemented in both the new SecureDrop App and the v2 journalist API. This design addresses significant performance issues present in the legacy SecureDrop Client's synchronization approach.

The SecureDrop App maintains synchronization with the SecureDrop server through a combination of periodic metadata updates and content fetching. The following diagram illustrates the planned sync state machine:



41.3.1 Synchronization Overview

Syncing (Metadata Updates)

Synchronization refers to one-directional metadata updates that bring the SecureDrop App up-to-date with information about source messages, journalist replies, and files on the SecureDrop server. This process determines what new content is available for download but does not include the actual message or file content.

The metadata synchronization includes:

- Source starring status
- Message and reply read status by journalists

- Journalist reply authorship information
- Source deletion status of replies (visible to journalists as “read by source” indicators)
- Timestamps of last source activity
- Journalist user account information for reply attribution

Fetching (Content Download)

Fetching encompasses the download and decryption of actual source messages and journalist replies. Messages are typically downloaded and decrypted immediately upon discovery to enable a chat-like user experience. Files, however, are never downloaded automatically and require explicit user action.

The server is the source of truth. Deletions on the server (a source the app knows about, but the server does not) trigger corresponding local deletions in the application.

41.3.2 Sync Loop Operation

The application implements a continuous sync loop to provide near real-time updates. By default, the application queries the server every 15 seconds when logged in. This interval balances responsiveness with server load considerations.

Steady State Behavior

Even on busy servers, most synchronization requests result in an empty delta (no changes), representing the steady state. During these periods, no user interface indicators are displayed for routine background syncing.

Active Synchronization

When the server reports changes (non-empty delta), the application transitions to active fetching:

- A progress indicator appears to inform users of ongoing download activity
- Messages and replies are downloaded and decrypted
- Local data is updated or deleted to match server state

The interface reflects new content once processing completes.

41.3.3 Online and Offline States

Offline Mode

When not logged in or when server connectivity is unavailable, the application operates in offline mode. Users can still access previously synchronized messages, replies, and files stored locally. This design accounts for potential server downtime or Tor network connectivity issues.

Online Transition

Upon successful login, the application immediately initiates synchronization to catch up with any changes that occurred during the offline period.

41.3.4 Error Handling

Synchronization failures are categorized as either recoverable or fatal. Some non-exhaustive examples:

Recoverable Errors

- Network connectivity issues
- Temporary Tor routing problems
- Transient server unavailability

The application may display error indicators, and will automatically retry these operations.

Fatal Errors

- Account deletion or suspension
- Incompatibility with the server (e.g., API mismatch)

These errors require user intervention and cannot be resolved through automatic retry mechanisms.

“SEEN BY” FEATURE

The “seen by” feature tracks which journalists are aware of items associated with a source in SecureDrop’s shared inbox model. This document describes the technical implementation and behavior across different SecureDrop interfaces.

42.1 Overview

Purpose

The feature supports multi-journalist review of content by providing visibility into which team members have viewed specific items. This enables coordination and prevents duplicate work.

Definitions

- **Item:** A source message, submitted file, or journalist reply
- **Seen:** An item has been viewed by at least one journalist
- **Shared inbox model:** Source content appears as “new” until *any* journalist has seen it

42.2 Current Implementation

The feature is currently implemented in the web-based *Journalist Interface* and in the Python/Qt-based SecureDrop Client.

The behavior is subtly different across these implementations, as summarized in the following table, and explained in more detail below.

Aspect	Web Interface	SecureDrop Client
Trigger	Item download	Source click (viewing conversation)
Granularity	Per-item as downloaded	All source items at once
Name Display	Not implemented	Tooltip on checkmark per item
Source Styling	Bold for sources with unseen items	Bold for sources with unseen items

42.2.1 Journalist Interface Behavior

In the web-based Journalist Interface:

- **Trigger:** Item status is set to “seen” when downloaded by a journalist
- **Granularity:** Per-item basis (as files are downloaded)
- **Source List UI:** Sources with unseen items are styled bold in the source list
- **Per-Item UI:** Individual items are styled as bold (unseen) or normal (seen) with closed/opened envelope icons

- **Limitations:** Does not display which specific journalists have seen items; does not convey whether any items have been copied to the *Secure Viewing Station*, decrypted or viewed

42.2.2 SecureDrop Client Behavior

In the Python/Qt-based SecureDrop Client:

- **Trigger:** All source items marked “seen” when conversation is viewed
- **Granularity:** All items in a conversation marked seen simultaneously
- **Source List UI:** Sources with unseen items are styled bold in the source list
- **Per-Item UI:** Journalist names displayed as tooltips on checkmarks for each item
- **Enhancement:** Shows specific journalist names who have seen each item
- **Limitation:** Does not convey whether submitted files have been downloaded, decrypted, or viewed

42.2.3 SecureDrop App Behavior

The implementation is being planned via [this tracking issue](#).

42.3 State Flow Diagrams

42.3.1 Journalist Interface State Flow

42.3.2 SecureDrop Client State Flow

42.4 API Integration

The feature integrates with the `/api/v1/seen` endpoint. For complete API specification including request/response formats and error handling, see the *Journalist Interface API* documentation.

Items marked as seen will include the journalist’s UUID in the `seen_by` field of API responses. These UUIDs can be resolved to journalist names using the `/api/v1/users` endpoint.

42.5 Additional Implementation Considerations

Special Cases

- **Offline Mode:** SecureDrop Client displays “seen by” information but does not record new views
- **Deleted Users:** Displayed as “deleted” in SecureDrop Client tooltips
- **Own Replies:** Journalist’s own replies are immediately marked as seen by the sending journalist—in other words, replying to a source does not change the source’s seen/unseen state (for any journalist, in any interface)

Security Properties

- Only tracks journalist viewing actions, never source actions
- Conveys limited information about whether items have been fully viewed (note differences between SecureDrop Client and Journalist Interface enumerated above)
- Viewing status is available to all journalists

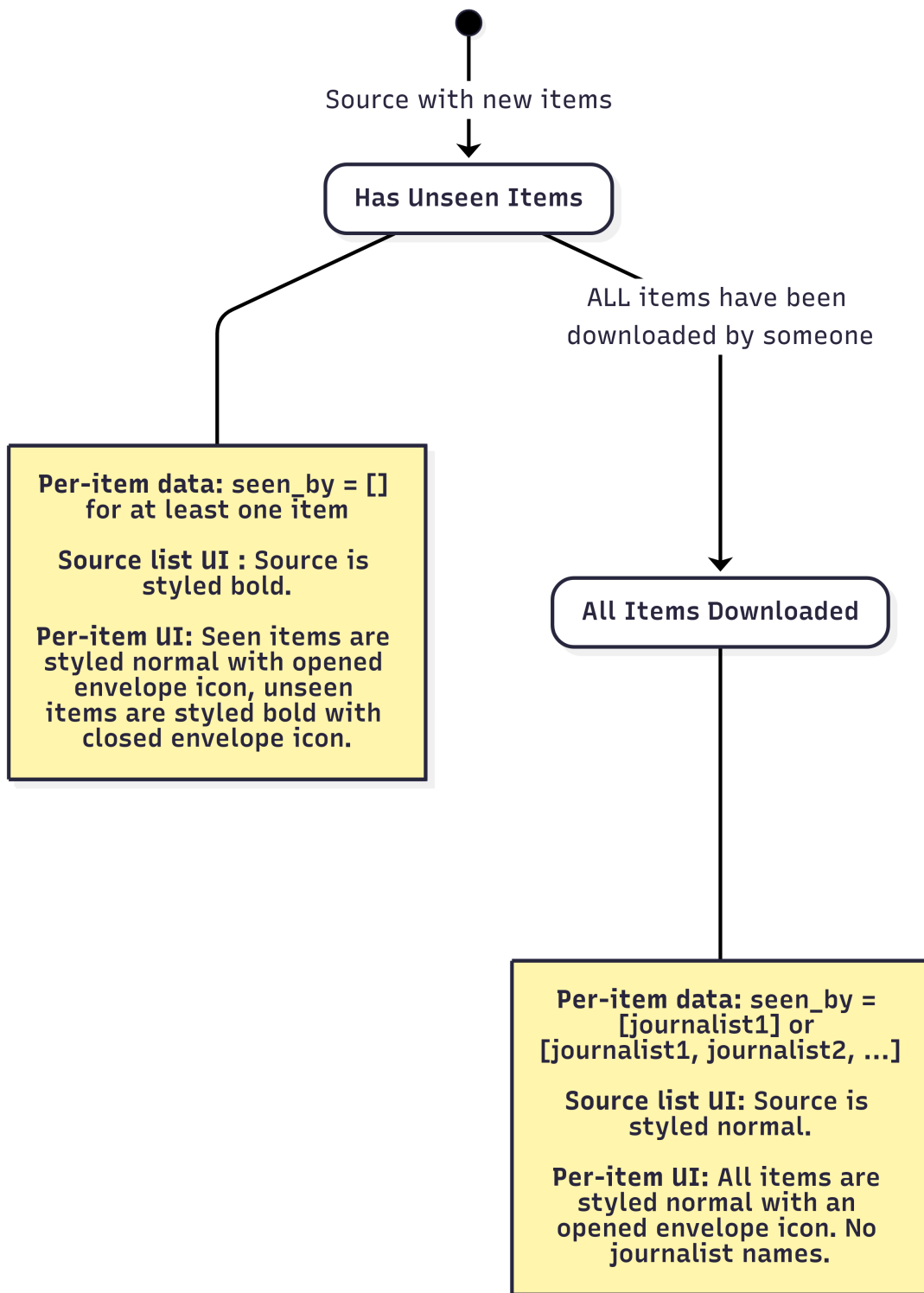


Fig. 1: A diagram showing how “seen” state is applied and displayed in the web-based *Journalist Interface* that is part of the SecureDrop Server.

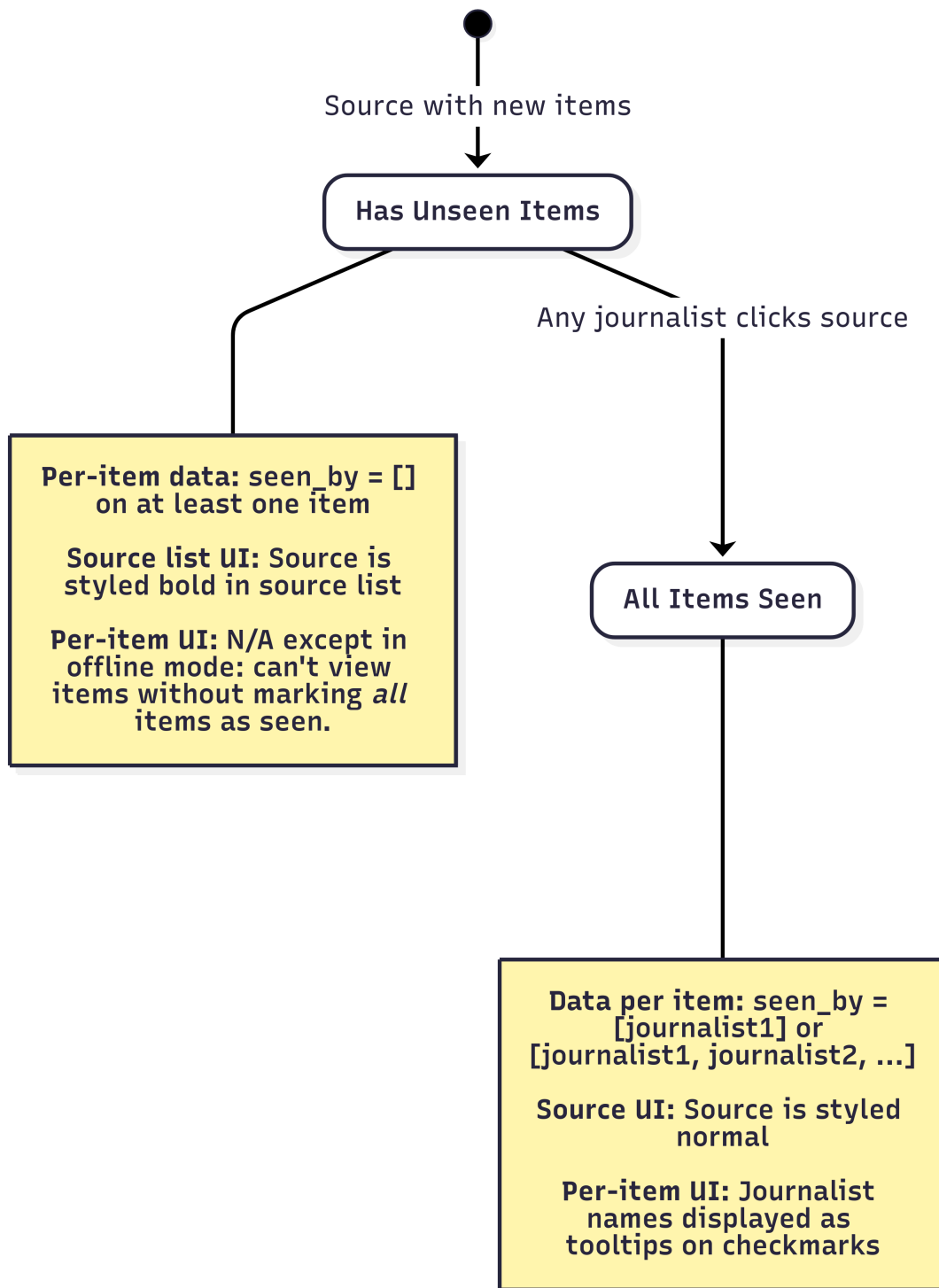


Fig. 2: A diagram showing how “seen” state is applied in the Python/Qt-based SecureDrop Client. Note that the entire conversation is marked as “seen” at once.

INDEX

T

translation freeze, [37](#)