RADICALLY OPEN SECURITY

# Penetration Test Report

## Open Tech Fund

V 1.1
Diemen, December 3rd, 2021
Public

## Document Properties

| | |
|---|---|
| Client | Open Tech Fund |
| Title | Penetration Test Report |
| Target | The Onionshare desktop and cli implementation (Release 2.4 https://github.com/onionshare/onionshare) |
| Version | 1.1 |
| Pentesters | Tillmann Weidinger, Philipp Koppe |
| Authors | Philipp Koppe, Tillmann Weidinger, Patricia Piolon, Tillmann Weidinger |
| Reviewed by | Patricia Piolon |
| Approved by | Melanie Rieback |

## Version control

| Version | Date | Author | Description |
|---|---|---|---|
| 0.1 | October 10th, 2021 | Philipp Koppe, Tillmann Weidinger | Initial draft |
| 1.0 | October 23rd, 2021 | Patricia Piolon | Review |
| 1.1 | December 3rd, 2021 | Tillmann Weidinger | Retest of the findings |

## Contact

For more information about this document and its contents please contact Radically Open Security B.V.


| | |
|---|---|
| Name | Melanie Rieback |
| Address | Overdiemerweg 28<br>1111 PP Diemen<br>The Netherlands |
| Phone | +31 (0)20 2621 255 |
| Email | info@radicallyopensecurity.com |

# Table of Contents

# 1 Executive Summary

## 1.1 Introduction

Between September 26, 2021 and October 8, 2021, Radically Open Security B.V. carried out a penetration test for Open Tech Fund

This report contains our findings as well as detailed explanations of exactly how ROS performed the penetration test.

## 1.2 Scope of work

The scope of the penetration test was limited to the following target(s):

- The Onionshare desktop and cli implementation (Release 2.4 https://github.com/onionshare/onionshare)

The scoped services are broken down as follows:

- Pentest on the desktop client implementation: 4-5 days
- Partial Code audit of shared code parts in the cli package: 5-6 days
- Retest/fixes: 0-2 days
- Reporting: 1.5 days
- Review & projectmanagement: 1 days
- **Total effort: 11.5 - 15.5 days**

## 1.3 Project objectives

ROS will perform a penetration test of the Onionshare application and components with OTF in order to assess the security of the application against adversaries. To do so ROS will access the Onionshare repositories and guide OTF in attempting to find vulnerabilities, exploiting any such found to try and gain further access and elevated privileges.

## 1.4 Timeline

The Security Audit took place between September 26, 2021 and October 8, 2021.

## 1.5 Results In A Nutshell

The most impactful finding we discovered is relevant for users of the desktop application and allows to render arbitrary HTML inside the Onionshare desktop application. This was possible due to missing input sanitization in the history

element OTF-001 (page 26) . While testing the capabilities of an adversary with such a primitive we discovered an out of bounds read in the QT image rendering component OTF-014 (page 12) . This can be abused to create a denial of service attack against the machine which runs the Onionshare desktop application.

Another denial of service issue was found in the receive mode OTF-012 (page 14) which blocked upload of files and could be reproduced over the tor network.

Most issues were found in the chat component, where adversaries with access could impersonate users OTF-005 (page 18) and OTF-003 (page 23) in the web interface. It was also possible to spoof a chat leave notification without actually leaving the chat, leading to a hidden eavesdropper OTF-004 (page 20) . Another method, which only allowed for hidden write access was found, which could combined with the impersonation bug to spoof messages from legitimate chat participants OTF-009 (page 16) .

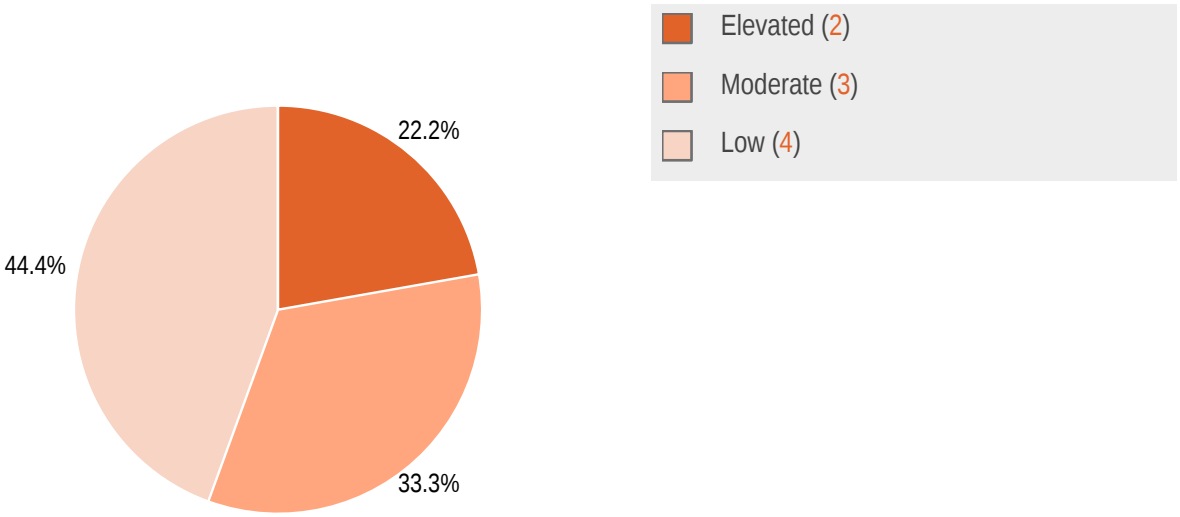
A low severity issue was found in the deployed applications and allowed for adversaries with existing exploits to facilitate these without common binary restrictions like Stack-Canaries OTF-013 (page 13) . Additionally, the webserver choice and configuration did not allow for CSP configuration, which would leave applications dependent on external resources without this common hardening OTF-006 (page 17) .

## 1.6 Summary of Findings

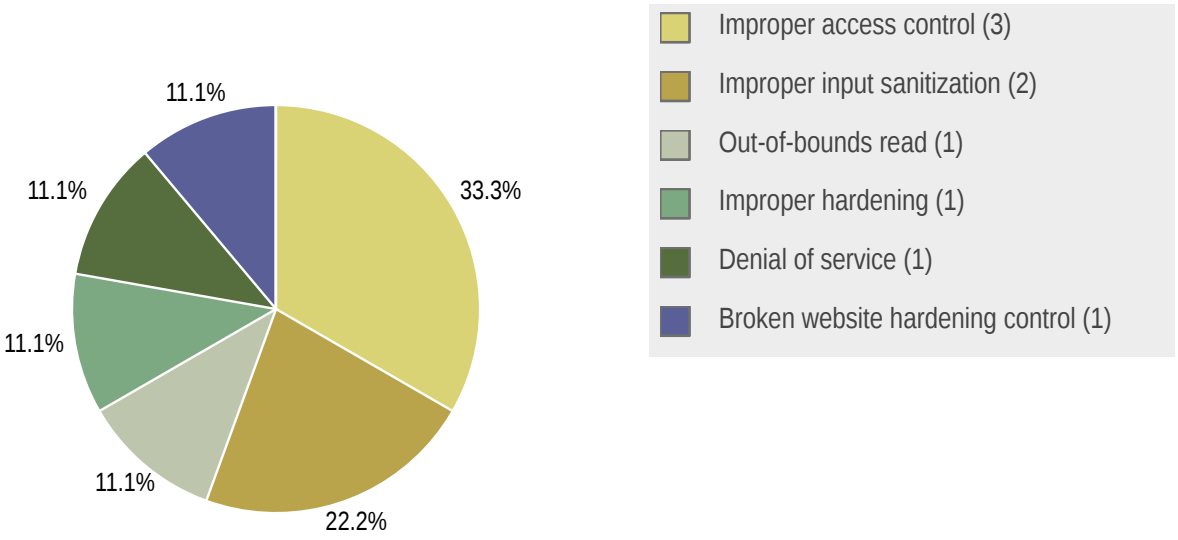
| ID | Type | Description | Threat level |
|---|---|---|---|
| OTF-014 | Out-of-bounds Read | The desktop application was found to be vulnerable to denial of service via an undisclosed vulnerability in the QT image parsing. | Elevated |
| OTF-001 | Improper Input Sanitization | The path parameter of the requested URL is not sanitized before being passed to the QT frontend. | Elevated |
| OTF-012 | Denial of Service | The receive mode limits concurrent uploads to 100 per second and blocks other uploads in the same second, which can be triggered by a simple script. | Moderate |
| OTF-004 | Improper Access Control | Chat participants can spoof their channel leave message, tricking others into assuming they left the chatroom. | Moderate |
| OTF-003 | Improper Access Control | Anyone with access to the chat environment can write messages disguised as another chat participant. | Moderate |
| OTF-013 | Improper Hardening | The filesystem restriction could be hardened and should only allow for pre-defined subfolders. | Low |
| OTF-009 | Improper Access Control | Authenticated users (or unauthenticated in public mode) can send messages without being visible in the list of chat participants. | Low |
| OTF-006 | Broken Website Hardening Control | The CSP can be turned on or off but not configured for the specific needs of the website. | Low |

| | | | |
|---|---|---|---|
| OTF-005 | Improper Input Sanitization | It is possible to change the username to that of another chat participant with an additional space character at the end of the name string. | Low |

### 1.6.1 Findings by Threat Level



### 1.6.2 Findings by Type




## 1.7 Summary of Recommendations

| ID | Type | Recommendation |
|---|---|---|
| OTF-014 | Out-of-bounds Read | • Monitor for upstream fix<br>• Fix OTF-001 (page 26) as a workaround |
| OTF-013 | Improper Hardening | • Reduce read access in Flatpak configuration. |
| OTF-012 | Denial of Service | • Remove this limitation<br>or<br>• Derive directory name from milliseconds |
| OTF-009 | Improper Access Control | • Allow chat access only after emission of the join event.<br>• Implement proper session handling. |
| OTF-006 | Broken Website Hardening Control | • Consider offering a configurable webserver choice<br>• Consider configurable CSP |
| OTF-005 | Improper Input Sanitization | • Remove non-visible characters from the username |
| OTF-004 | Improper Access Control | • Implement proper session handling |
| OTF-003 | Improper Access Control | • Implement proper session handling |

| | | |
|---|---|---|
| OTF-001 | Improper Input Sanitization | • Manually define the text format of the QLabel via `setTextFormat()` |

# 2 Methodology

## 2.1 Planning

Our general approach during penetration tests is as follows:

1. **Reconnaissance**
   We attempt to gather as much information as possible about the target. Reconnaissance can take two forms: active and passive. A passive attack is always the best starting point as this would normally defeat intrusion detection systems and other forms of protection afforded to the app or network. This usually involves trying to discover publicly available information by visiting websites, newsgroups, etc. An active form would be more intrusive, could possibly show up in audit logs and might take the form of a social engineering type of attack.
2. **Enumeration**
   We use various fingerprinting tools to determine what hosts are visible on the target network and, more importantly, try to ascertain what services and operating systems they are running. Visible services are researched further to tailor subsequent tests to match.
3. **Scanning**
   Vulnerability scanners are used to scan all discovered hosts for known vulnerabilities or weaknesses. The results are analyzed to determine if there are any vulnerabilities that could be exploited to gain access or enhance privileges to target hosts.
4. **Obtaining Access**
   We use the results of the scans to assist in attempting to obtain access to target systems and services, or to escalate privileges where access has been obtained (either legitimately though provided credentials, or via vulnerabilities). This may be done surreptitiously (for example to try to evade intrusion detection systems or rate limits) or by more aggressive brute-force methods. This step also consist of manually testing the application against the latest (2017) list of OWASP Top 10 risks. The discovered vulnerabilities from scanning and manual testing are moreover used to further elevate access on the application.

## 2.2 Risk Classification

Throughout the report, vulnerabilities or risks are labeled and categorized according to the Penetration Testing Execution Standard (PTES). For more information, see: http://www.pentest-standard.org/index.php/Reporting

These categories are:

- **Extreme**
  Extreme risk of security controls being compromised with the possibility of catastrophic financial/reputational losses occurring as a result.

- **High**

  High risk of security controls being compromised with the potential for significant financial/reputational losses occurring as a result.
- **Elevated**

  Elevated risk of security controls being compromised with the potential for material financial/reputational losses occurring as a result.
- **Moderate**

  Moderate risk of security controls being compromised with the potential for limited financial/reputational losses occurring as a result.
- **Low**

  Low risk of security controls being compromised with measurable negative impacts as a result.

# 3 Reconnaissance and Fingerprinting

We were able to gain information about the software and infrastructure through the following automated scans. Any relevant scan output will be referred to in the findings.

- nmap – https://nmap.org
- Burp Suite Professional – https://portswigger.net/burp/pro
- GoBuster – https://github.com/OJ/gobuster

# 4 Findings

We have identified the following issues:

## 4.1 OTF-014 — [Desktop] The QT application is Vulnerable to Out-of-Bounds Read of Uninitialized Heap Memory

| | |
|---|---|
| **Vulnerability ID:** OTF-014 | **Status:** Resolved |
| **Vulnerability type:** Out-of-bounds Read | |
| **Threat level:** Elevated | |

### Description:

The desktop application was found to be vulnerable to denial of service via an undisclosed vulnerability in the QT image parsing.

### Technical description:

Prerequisites:

- Onion address is known
- Public service or authentication is valid
- Desktop application is used
- History is displayed

The rendering of images found in OTF-001 (page 26) could be elevated to a Denial of Service, which requires only very few bytes to be sent as a path parameter to any of the Onionshare functions. Roughly 20 bytes lead to 4GB memory consumption and this can be triggered multiple times. To be abused, this vulnerability requires rendering in the history tab, so some user interaction is required. The issue is in the process of disclosure to the QT security mailing list. More details will be provided after a fixed QT build has been deployed.

**Retest:**

The issue was partially resolved. The path value is now sanitized in OTF-001 (page 26), but the upstream issues are not resolved at the time of retest.

### Impact:

An adversary with knowledge of the Onion service address in public mode or with authentication in private mode can perform a Denial of Service attack, which quickly results in out-of-memory for the server. This requires the desktop application with rendered history, therefore the impact is only elevated.

### Recommendation:

- Monitor for upstream fix
- Fix OTF-001 (page 26) as a workaround

## 4.2 OTF-013 — [CLI] Flatpak and Snap Configurations Allow for Read Access on the Entire Homefolder

| | |
|---|---|
| **Vulnerability ID:** OTF-013 | **Status:** Resolved |
| **Vulnerability type:** Improper Hardening | |
| **Threat level:** Low | |

### Description:

The filesystem restriction could be hardened and should only allow for pre-defined subfolders.

### Technical description:

The Flatpak and Snap configurations allow for read-only access on the whole home folder. The relevant lines in the configuration files are `onionshare/snap/snapcraft.yaml#L20` and `onionshare/flatpak/org.onionshare.OnionShare.yaml#L19`, respectively.

The encapsulation of filesystem access via these mechanisms should be restricted to pre-defined folders and not allow for access to (configuration) files outside the Onionshare-specific folders.

Sadly Snap does not allow for further restriction to specific folders and therefore cannot be further hardened. By default both frameworks disallow access to hidden folders and therefore reduce the potential impact.

**Retest:**

The issue was resolved. The configuration is not allowing read access to the home folder aside from proper file dialogs.

### Impact:

An adversary with a primitive that allows for filesystem access from the context of the Onionshare process can access sensitive files in the entire user home folder. This could lead to the leaking of sensitive data. Due to the automatic exclusion of hidden folders, the impact is reduced.

### Recommendation:

- Reduce read access in Flatpak configuration.

## 4.3 OTF-012 — [Receive] The Receive Mode is Vulnerable to Denial of Service

**Vulnerability ID:** OTF-012 **Status:** Resolved

**Vulnerability type:** Denial of Service

**Threat level:** Moderate

### Description:

The receive mode limits concurrent uploads to 100 per second and blocks other uploads in the same second, which can be triggered by a simple script.

### Technical description:

The following script uses GNU parallel and curl with around 6000 requests in parallel to send 10000 requests. A change in the `ulimit -n` configuration is required for it to work. This is sufficient to block file upload on a (public) receive instance.

```
seq 10000 | parallel --max-args 0 --jobs 6000 "curl -i -s -x socks5h://localhost:9150 -
k -X $'POST' -H $'Host: csqrp3qciewvj5axph4o62jnr6aevhmpxfkydmi3256bprhbusr2ltid.onion'
 -H $'Accept-Encoding: gzip, deflate' -H $'Content-Type: multipart/form-data;
 boundary=---------------------------191823767039180748733753870 42' -H $'Content-Length: 329' -H
 $'Connection: close' --data-binary $'---------------------------191823767039180748733753870 42\x0d
\x0aContent-Disposition: form-data; name=\"file[]\"; filename=\"poc.txt\"\x0d\x0aContent-Type:
 text/plain\x0d\x0a\x0d\x0aA\x0d\x0a---------------------------191823767039180748733753870 42\x0d
\x0aContent-Disposition: form-data; name=\"text\"\x0d\x0a\x0d\x0a\x0d
\x0a---------------------------191823767039180748733753870 42--\x0d\x0a' $'http://
csqrp3qciewvj5axph4o62jnr6aevhmpxfkydmi3256bprhbusr2ltid.onion/upload-ajax'"
```

Attack duration was around 80 seconds.

Cases where over 99 requests were sent per second:

```
Every 0.1s: ls | grep...  onionvm: Tue Oct  5 12:17:00 2021

78
```

Cases where files were successfully written to disk:

```
Every 0.1s: ls | wc -w    onionvm: Tue Oct  5 12:17:00 2021

8399
```

This means that during the attack time 1601 requests of 10000 were dropped. We tried to upload multiple files in the web interface during the attack and were not successful.

The failsafe is used to prevent creating more than 100 directories per second:

```
# Create that directory, which shouldn't exist yet
            try:
                os.makedirs(self.receive_mode_dir, 0o700, exist_ok=False)
            except OSError:
                # If this directory already exists, maybe someone else is uploading files at
                # the same second, so use a different name in that case
                if os.path.exists(self.receive_mode_dir):
                    # Keep going until we find a directory name that's available
                    i = 1
                    while True:
                        new_receive_mode_dir = f"{self.receive_mode_dir}-{i}"
                        try:
                            os.makedirs(new_receive_mode_dir, 0o700, exist_ok=False)
                            self.receive_mode_dir = new_receive_mode_dir
                            break
                        except OSError:
                            pass
                        i += 1
                        # Failsafe
                        if i == 100:
                            self.web.common.log(
                                "ReceiveModeRequest",
                                "__init__",
                                "Error finding available receive mode directory",
                            )
                            self.upload_error = True
                            break
            except PermissionError:
                self.web.add_request(
                    self.web.REQUEST_ERROR_DATA_DIR_CANNOT_CREATE,
                    request.path,
                    {"receive_mode_dir": self.receive_mode_dir},
                )
                print(
                    f"Could not create OnionShare data folder: {self.receive_mode_dir}"
                )
                self.web.common.log(
                    "ReceiveModeRequest",
                    "__init__",
                    "Permission denied creating receive mode directory",
                )
                self.upload_error = True
```

The limit of 100 requests/second is significantly lower than the possible network bandwidth and greatly reduces the attack complexity for denial of service. Our test was conducted over the tor network, which showed no limitation for the required bandwidth.

**Retest:**

The issue was resolved. The folder name contains the nanosecond of the request time.

### Impact:

An adversary with access to the receive mode can block file upload for others. There is no way to block this attack in public mode due to the anonymity properties of the tor network.

### Recommendation:

- Remove this limitation
  or
- Derive directory name from milliseconds

## 4.4 OTF-009 — [Chat] Users Can Send Messages Without Presence in the User List

**Vulnerability ID:** OTF-009 **Status:** Resolved

**Vulnerability type:** Improper Access Control

**Threat level:** Low

### Description:

Authenticated users (or unauthenticated in public mode) can send messages without being visible in the list of chat participants.

### Technical description:

Prerequisites:

- Existing chatroom
- Access to the chatroom (Public or known Private Key)

- Either a modified frontend client or manual requests from burp/curl

If a user opens the chatroom without emitting the join message he will not be present in `session.users[x]` list.

Therefore there is no listing in the frontend and no chat participant knows another party joined the chat. It is still possible to send messages in the chatroom.

If a user decides to abuse OTF-003 (page 23) he can impersonate messages from existing users; others would not be able to distinguish original and faked messages. This is also a prerequisite for OTF-004 (page 20).

**Retest:**

The issue was resolved. The client side is no longer announcing their presence and this is tracked on the server side.

### Impact:

An adversary with access to the chat environment can send messages to the chat without being visible in the list of chat participants.

### Recommendation:

- Allow chat access only after emission of the join event.
- Implement proper session handling.

## 4.5 OTF-006 — [Website] CSP Cannot be Configured

**Vulnerability ID:** OTF-006 **Status:** Resolved

**Vulnerability type:** Broken Website Hardening Control

**Threat level:** Low

### Description:

The CSP can be turned on or off but not configured for the specific needs of the website.

### Technical description:

The website mode of the application allows to use a hardened CSP, which will block any scripts and external resources. It is not possible to configure this CSP for individual pages and therefore the security enhancement cannot be used for websites using javascript or external resources like fonts or images.

If CSP were configurable, the website creator could harden it accordingly to the needs of the application.

As this issue correlates with the Github issue for exposing the flask application directly (https://github.com/onionshare/onionshare/issues/1389), it can be assumed that this can be solved by either changing to a well-known webserver, which supports this kind of configuration, or enhancing the status quo by making the CSP a configurable part of each website.

We believe that bundling the nginx or apache webserver would add complexity and dependencies to the application that could result in a larger attack surface - as these packages receive regular security updates. On the other hand it is not recommended to directly expose the flask webserver, due to lack of hardening. This is a trade-off which needs to be evaluated by the Onionshare developers, as multiple features are involved. Ideally the application user could choose between the built-in flask webserver or a system webserver of choice.

**Retest:**

The issue was resolved. The CSP can be configured and passed to the application via command line.

### Impact:

As this is a general weakness and not a direct vulnerability in the Onionshare application, the direct impact of this issue is rather low.

### Recommendation:

- Consider offering a configurable webserver choice
- Consider configurable CSP

## 4.6 OTF-005 — [Chat] Users can be Impersonated with Similar Usernames

**Vulnerability ID:** OTF-005 **Status:** Resolved

**Vulnerability type:** Improper Input Sanitization

**Threat level:** Low

### Description:

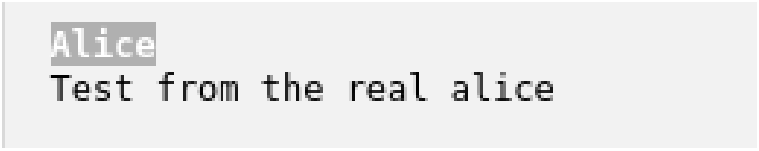
It is possible to change the username to that of another chat participant with an additional space character at the end of the name string.

## Technical description:

Assumed users in Chat:

- Alice
- Bob
- Mallory

1. Mallory renames to `Alice `.
2. Mallory sends message as `Alice `.
3. Alice and Bob receive a message from Mallory disguised as `Alice `, which is hard to distinguish from the `Alice` in the web interface.


Other (invisible) whitespace characters were found to be working as well.

**Retest:**

The issue was resolved. User names are stripped from leading or trailing spaces and the user name value is now limited to ASCII only, due to invisible non-whitespace characters like `U+3164 HANGUL FILLER`.

## Impact:

An adversary with access to the chat environment can use the rename feature to impersonate other participants by adding whitespace characters at the end of the username.

## Recommendation:

- Remove non-visible characters from the username

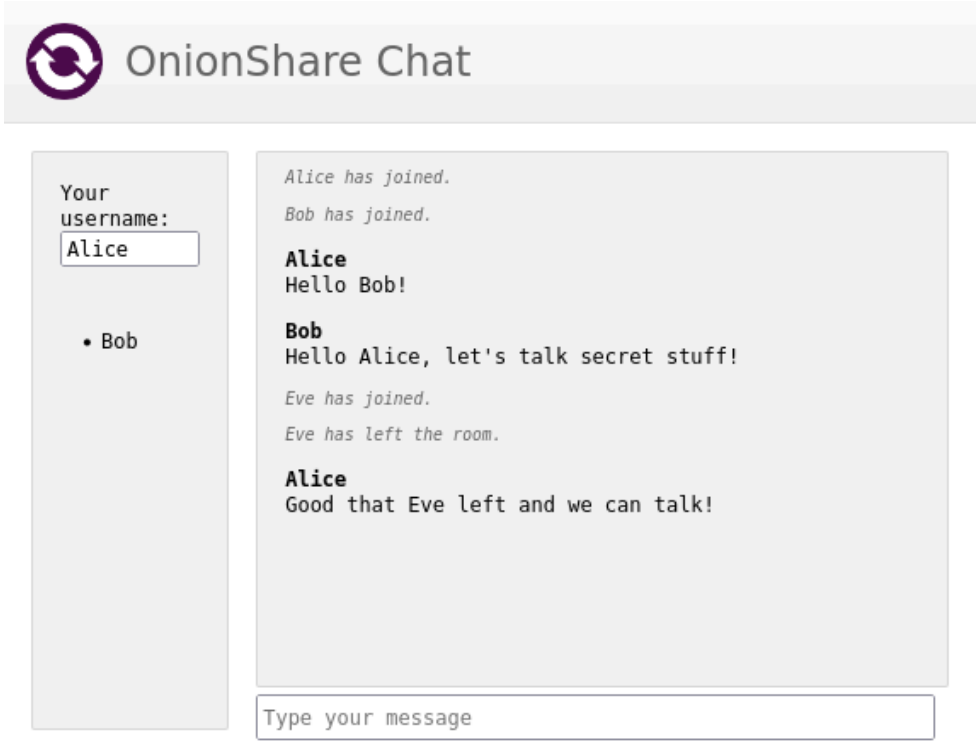
## 4.7 OTF-004 — [Chat] Users Can Spoof Leaving a Chatroom

| | |
|---|---|
| **Vulnerability ID:** OTF-004 | **Status:** Resolved |
| **Vulnerability type:** Improper Access Control | |
| **Threat level:** Moderate | |

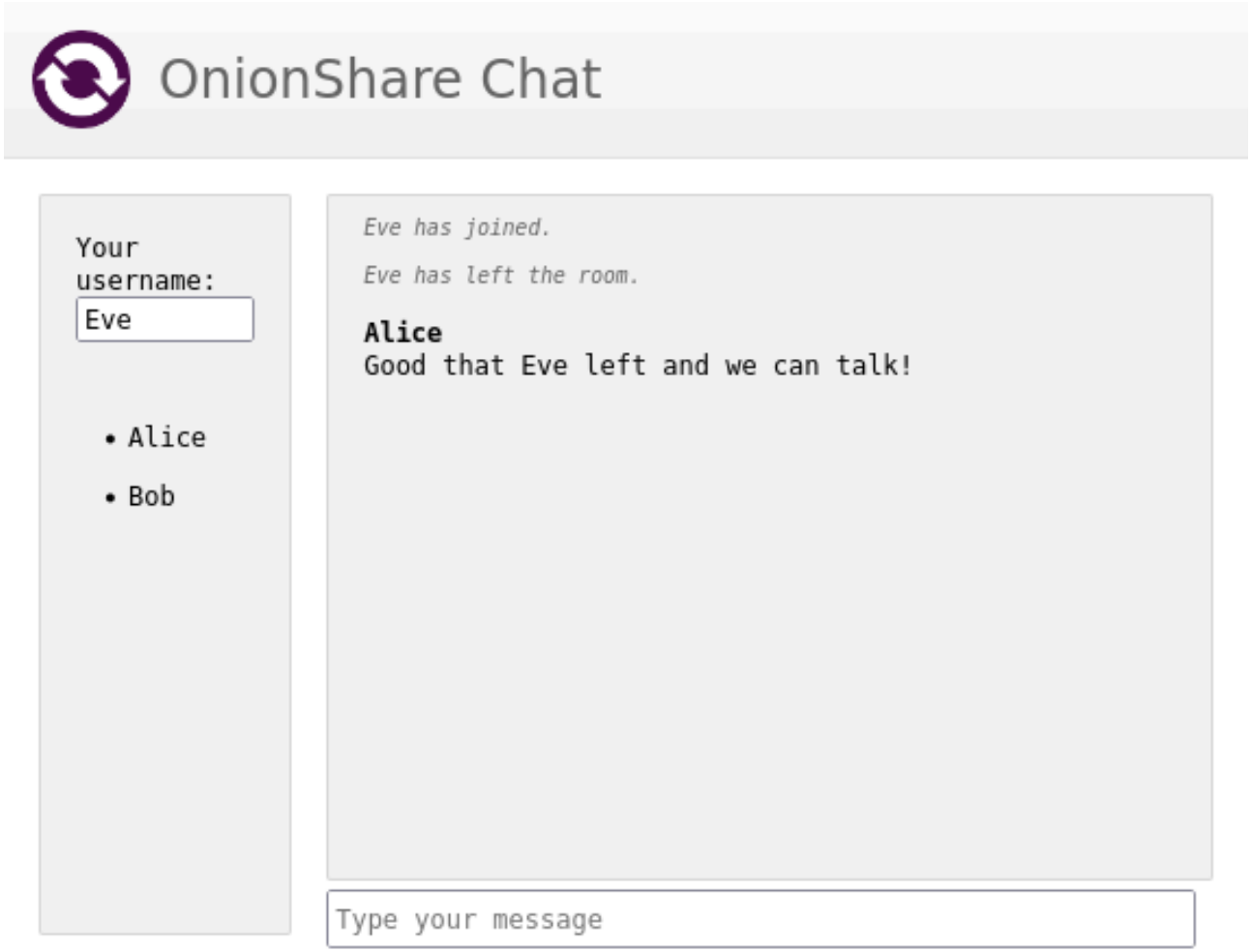
### Description:

Chat participants can spoof their channel leave message, tricking others into assuming they left the chatroom.

### Technical description:




OnionShare Chat
Your username:
Bob
• Alice
Bob has joined.
Alice
Hello Bob!
Bob
Hello Alice, let's talk secret stuff!
Eve has joined.
Eve has left the room.
Alice
Good that Eve left and we can talk!
Type your message




This series of screenshots show Alice, Bob and Eve joined a chatroom and are the only participants in the chatroom. Eve seemingly leaves the chatroom, which leads Bob and Alice to believe they are having a private chat. The last screenshot shows that Eve only emitted the leave message and is still able to read the chat and possibly write messages.

This can be reproduced by joining the chat with two different instances, where one instance has slightly modified the client-side JavaScript code similar to OTF-003 (page 23). The `joined` emit needs to be removed from the `connect` event handler. Therefore the modified client is not listed in the userlist and has no active session. The modified non-listed user also needs to change their username to Eve, which is not shown in the chatroom. The modified client then emits the `disconnect` event and their connection is no longer usable.

This results in the leave message for Eve and the removal from the user-list but not in removal of the original session of the Eve who announced to join the chat.

**Retest:**

The issue was resolved. The client side is no longer announcing their presence and this is tracked on the server side.

### Impact:

An adversary with access to the chat environment can spoof his leave event but still persist in the chat with access to all sent messages and the possibility to write in the chat using OTF-003 (page 23).

### Recommendation:

- Implement proper session handling

## 4.8 OTF-003 — [Chat] Message Sender Can Be Spoofed

| | |
|---|---|
| **Vulnerability ID:** OTF-003 | **Status:** Resolved |
| **Vulnerability type:** Improper Access Control | |
| **Threat level:** Moderate | |

### Description:

Anyone with access to the chat environment can write messages disguised as another chat participant.

### Technical description:

Prerequisites:

- Alice and Bob are legitimate users
- A third user has access to the chat environment


This screenshot shows Alice (`glimpse-depress`) and Bob (`blinker-doorpost`) joined a chatroom and are the only participants in the chatroom. Then the non-listed user `squad-nursing` writes a message in the chatroom without being visible in the list of users. The sending of the message itself is not required but was done here to show the initial access. The non-listed participant now renames himself to Bob and writes another message, seemingly coming from Bob.

This can be reproduced by slightly modifying the client-side JavaScript. The `joined` emit needs to be removed from the `socket.on(connect)` event handler. Therefore a client is not listed in the userlist and has no active session.

`onionshare/cli/onionshare_cli/resources/static/js/chat.js#L16`

```
socket.on('connect', function () {
      // socket.emit('joined', {});
    });
```

This can be done either via a crafted client or runtime modification of the `chat.js` script in the browser's internal debugger.

It is still possible to call the `text` method and send text to the chat via websocket.

`cli/onionshare_cli/web/chat_mode.py#L131`

```
@self.web.socketio.on("text", namespace="/chat")
        def text(message):
            """Sent by a client when the user entered a new message.
            The message is sent to all people in the room."""
            emit(
                "message",
                {"username": session.get("name"), "msg": message["msg"]},
                room=session.get("room"),
            )
```

It is also possible to call the `update_username` function and choose an existing username from the chat.

`cli/onionshare_cli/web/chat_mode.py#L141`

```
@self.web.socketio.on("update_username", namespace="/chat")
        def update_username(message):
            """Sent by a client when the user updates their username.
            The message is sent to all people in the room."""
            current_name = session.get("name")
            if message.get("username", ""):
                session["name"] = message["username"]
                self.connected_users[
                    self.connected_users.index(current_name)
                ] = session.get("name")
            emit(
                "status",
                {
                    "msg": "{} has updated their username to: {}".format(
                        current_name, session.get("name")
                    ),
                    "connected_users": self.connected_users,
                    "old_name": current_name,
                    "new_name": session.get("name"),
                },
                room=session.get("room"),
            )
```

Afterwards the hidden user can send messages that are displayed as coming from the impersonated user. There is no way to distinguish between the fake and original message.

**Retest:**

The issue was resolved. The authentication process no longer allows for multiple rooms and the session is bound to a unique user name. User names are also sanitized see OTF-005 (page 18) to prevent non-unique chat usernames.

## Impact:

An adversary with access to the chat environment can impersonate existing chat participants and write messages but not read the conversation. The similar exploit described in OTF-004 (page 20) has only slightly more requirements but also allows for reading.

## Recommendation:

- Implement proper session handling

## 4.9 OTF-001 — [Desktop] The Path Parameter of the History Element is not Sanitized

| | |
|---|---|
| **Vulnerability ID:** OTF-001 | **Status:** Resolved |
| **Vulnerability type:** Improper Input Sanitization | |
| **Threat level:** Elevated | |

### Description:

The `path` parameter of the requested URL is not sanitized before being passed to the QT frontend.

### Technical description:

The `path` parameter is not sanitized before being passed to the constructor of the `QLabel` .

`onionshare/desktop/src/onionshare/tab/mode/__init__.py#L499`

```
def handle_request_individual_file_started(self, event):
        """
        Handle REQUEST_INDVIDIDUAL_FILES_STARTED event.
        Used in both Share and Website modes, so implemented here.
        """
        self.toggle_history.update_indicator(True)
        self.history.requests_count += 1
        self.history.update_requests()

        item = IndividualFileHistoryItem(self.common, event["data"], event["path"])
        self.history.add(event["data"]["id"], item)
```

`onionshare/desktop/src/onionshare/tab/mode/history.py#L483`

```
class IndividualFileHistoryItem(HistoryItem):
    def __init__(self, common, data, path):
        super(IndividualFileHistoryItem, self).__init__()
        self.status = HistoryItem.STATUS_STARTED
        self.common = common

        self.id = id
        self.path = path

        self.path_label = QtWidgets.QLabel(self.path)
```

https://doc.qt.io/qt-5/qlabel.html#details

```
Warning: When passing a QString to the constructor or calling setText(),
make sure to sanitize your input, as QLabel tries to guess whether it displays the text
as plain text or as rich text, a subset of HTML 4 markup. You may want to call setTextFormat()
explicitly, e.g. in case you expect the text to be in plain format but cannot control the text
 source
```

```
(for instance when displaying data loaded from the Web).
```

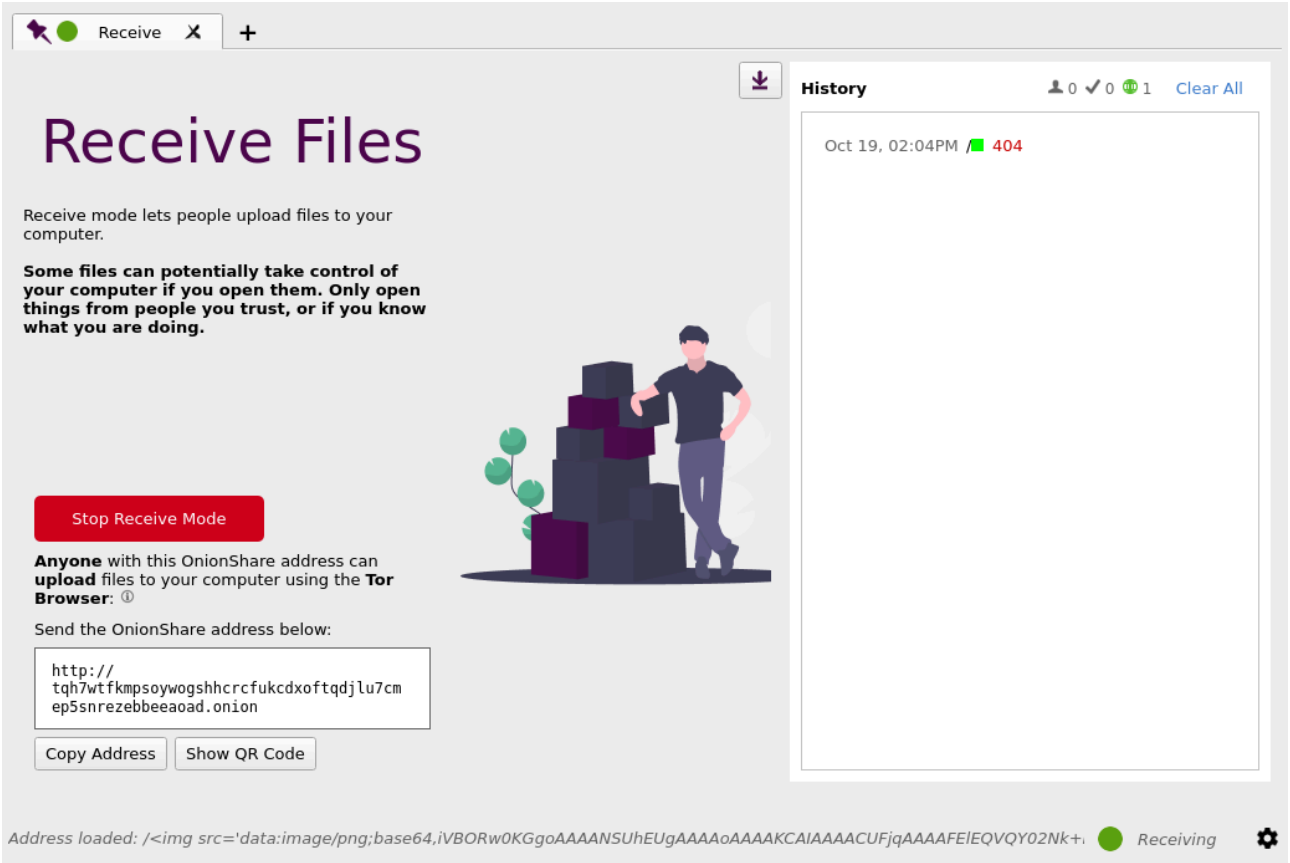
This path is used in all components for displaying the server access history. This leads to a rendered HTML4 Subset (QT RichText editor) in the Onionshare frontend.

In the following example an adversary injects a crafted image file into an Onionshare instance with receive mode and renders it in the history component of the Onionshare application.

The only requirement is another visit to the shared site with the following parameter attached to the path of the URL:

```
<img src='
AAAAFElEQVQY02Nk+M+ABzAxMIxKYwIAQC0BEwZFOw4AAAAASUVORK5CYII=' />
```

This will be rendered as a green square in the history tab where the path value is supposed to be (the value itself is shown at the bottom of the page).




Possible scenarios where this could lead to remote code execution would be a 0-day in libpng or other internal image rendering (OTF-014 (page 12)) of the QT framework.

The QT documentation indicates that external files could be rendered, but we were unable to find a QT code path allowing for it.

**Retest:**

The issue was resolved. The content type is now plain text only.

### Impact:

An adversary with knowledge of the Onion service address in public mode or with authentication in private mode can render arbitrary HTML (QT-HTML4 Subset) in the server desktop application. This requires the desktop application with rendered history, therefore the impact is only elevated.

### Recommendation:

- Manually define the text format of the QLabel via `setTextFormat()`

# 5 Non-Findings

In this section we list some of the things that were tried but turned out to be dead ends.

## 5.1 NF-015 — [CLI] The File Names and Folder Names Are Properly Sanitized

The usage of `werkzeug.secure_filename` and subsequent usage of safe file name shows no weakness against bogus, user-supplied file names. The folder names are not based on user-supplied input.

## 5.2 NF-010 — [CLI] The Frontend Is Not Vulnerable to Cross-Site Scripting

The frontend code found in `chat.js`, `receive.js` and `chat.js` properly escapes user-supplied content and properly constructs dynamic HTML tags.

# 6 Future Work

- **Retest of findings**
  When mitigations for the vulnerabilities described in this report have been deployed, a repeat test should be performed to ensure that they are effective and have not introduced other security problems.
- **Regular security assessments**
  Security is an ongoing process and not a product, so we advise undertaking regular security assessments and penetration tests, ideally prior to every major release or every quarter.

# 7 Conclusion

We discovered 2 Elevated, 4 Low and 3 Moderate -severity issues during this penetration test.

The penetration test goals were the de-anonymization of users and code execution on any of the involved parties, which was not found possible in the time allocated for the engagement. This is most likely due to the choice of offloading the client interaction and authentication fully on the Tor-browser and relying on the security assumptions of a recent and well maintained browser. Additionally, the usage of stable third party libraries for file and network handling, as well as the separation of logic and user interface exposed only a minimal attack surface. User-controlled input is minimal and in most cases sanitized or validated.

The direct exposure of the flask web server is one of the few issues that require more in-depth consideration and further work to harden the application. Using the system's temporary storage for storing compressed versions of files to serve is not the most streamlined way to approach fileserving, as it introduces file size limits and adds additional complexity, but we did not find any security-relevant issues in this component.

The implemented QT frontend did have some potential high-impact issues, as arbitrary HTML (QT Subset) could be rendered. This lead to discovery of a previously unknown vulnerability in the QT image handling process, and we do believe capable and motivated adversaries could achieve code execution. This assumption is based on the well known history of security-relevant bugs (CVE-2015-1860,CVE-2011-3194,CVE-2018-19873) in this component of the QT framework.

Our general impression is that the Onionshare project has no major security vulnerabilities and can be used within the properly documented boundaries. Sane default configurations were chosen and inexperienced users are warned about the consequences of sensitive configuration changes.

We recommend fixing all of the issues found and then performing a retest in order to ensure that mitigations are effective and that no new vulnerabilities have been introduced.

Finally, we want to emphasize that security is a process – this penetration test is just a one-time snapshot. Security posture must be continuously evaluated and improved. Regular audits and ongoing improvements are essential in order to maintain control of your corporate information security. We hope that this pentest report (and the detailed explanations of our findings) will contribute meaningfully towards that end.

Please don't hesitate to let us know if you have any further questions, or need further clarification on anything in this report.

# Appendix 1 Testing team

| | |
|---|---|
| Tillmann Weidinger | Tillmann is a trained full-stack developer with a strong emphasis on security. He started tinkering with computers in his early teens. Due to this he has multiple years of experience in (reverse-)engineering hard- and software, software architecture and breaking things. His main interests evolve around Secure Coding, Automation, Web Applications, WiFi, DMA attacks and other topics between hard- and software with a focus on red-teaming. He enjoys programming in multiple languages and recently has chosen rust as his new favorite. He started studying IT-Security at Ruhr University Bochum and switched to Computer Science at FH Bochum and will graduate in 2021. Due to his broad experience in application development and system's security he can quickly adapt to new IT-Security related topics and is always happy to learn lesser known facts. |
| Philipp Koppe | Philipp is a PhD candidate in IT-Security at Ruhr University Bochum, focusing on reverse engineering and mitigating the exploitation of software vulnerabilities. He also loves to build software security and anti-piracy solutions for embedded systems. He has many years of experience in the security and engineering domain and is always happy with building or breaking systems. |
| Melanie Rieback | Melanie Rieback is a former Asst. Prof. of Computer Science from the VU, who is also the co-founder/CEO of Radically Open Security. |