



## Audit Report

### Sniffnet

V 1.0  
Amsterdam, March 28th, 2025  
Public

## Document Properties

Client	Sniffnet
Title	Audit Report
Target	<ul style="list-style-type: none"><li>Sniffnet (macos, Windows, Linux, FreeBSD)</li></ul>
Version	1.0
Pentester	Morgan Hill
Authors	Morgan Hill, Marcus Bointon
Reviewed by	Marcus Bointon
Approved by	Melanie Rieback

## Version control

Version	Date	Author	Description
0.1	March 26th, 2025	Morgan Hill	Initial draft
0.2	March 28th, 2025	Marcus Bointon	Review
1.0	March 28th, 2025	Marcus Bointon	1.0

## Contact

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

Name	Melanie Rieback
Address	Science Park 608 1098 XH Amsterdam The Netherlands
Phone	+31 (0)20 2621 255
Email	info@radicallyopensecurity.com

Radically Open Security B.V. is registered at the trade register of the Dutch chamber of commerce under number 60628081.

# Table of Contents

<b>1</b>	<b>Executive Summary</b>	<b>4</b>
1.1	Introduction	4
1.2	Scope of work	4
1.3	Project objectives	4
1.4	Timeline	4
1.5	Results In A Nutshell	4
1.6	Summary of Findings	5
1.6.1	Findings by Threat Level	5
1.6.2	Findings by Type	6
1.7	Summary of Recommendations	6
<b>2</b>	<b>Methodology</b>	<b>7</b>
2.1	Planning	7
2.2	Risk Classification	7
<b>3</b>	<b>Findings</b>	<b>9</b>
3.1	CLN-002 — Directory traversal through file name field	9
3.2	CLN-004 — CAP_NET_ADMIN not required on Linux	10
3.3	CLN-001 — Unmaintained dependencies	12
<b>4</b>	<b>Non-Findings</b>	<b>15</b>
4.1	NF-006 — MacOS privilege reduction	15
4.2	NF-005 — Popular Linux distros don't supply ambient ptrace	16
4.3	NF-007 — FreeBSD privilege reduction	17
4.4	NF-008 — Update checker just checks	18
4.5	NF-011 — Minimal UI rendering attack surface	18
<b>5</b>	<b>Future Work</b>	<b>19</b>
<b>6</b>	<b>Conclusion</b>	<b>20</b>
<b>Appendix 1</b>	<b>Testing team</b>	<b>21</b>

# 1 Executive Summary

## 1.1 Introduction

Between March 8, 2025 and March 30, 2025, Radically Open Security B.V. carried out an audit for Sniffnet.

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 targets:

- Sniffnet (macos, Windows, Linux, FreeBSD)

The scoped services are broken down as follows:

- Static analysis and dependency checking: 0.5 days
- Review operating system interaction: 2.5 days
- Interactive testing for crashes: 1 days
- Review: 1 days
- **Total effort: 5 days**

## 1.3 Project objectives

ROS will perform an audit of Sniffnet with its developers in order to assess the security of the desktop application. To do so ROS will access Sniffnet source code and build artifacts and guide Sniffnet 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 March 8, 2025 and March 30, 2025.

## 1.5 Results In A Nutshell

This audit discovered one low-severity vulnerability and two N/A findings.

The low-severity issue [CLN-002](#) (page 9) relates to directory traversal in the file name field when selecting a location to save a PCAP export of the Sniffnet session. When combined with Sniffnet's enhanced privileges, this could present a

convoluted path to privilege escalation or remote code execution requiring several user interactions. We did not discover a way to complete the privilege escalation or remote code execution chains through partial control of a PCAP file during this audit.

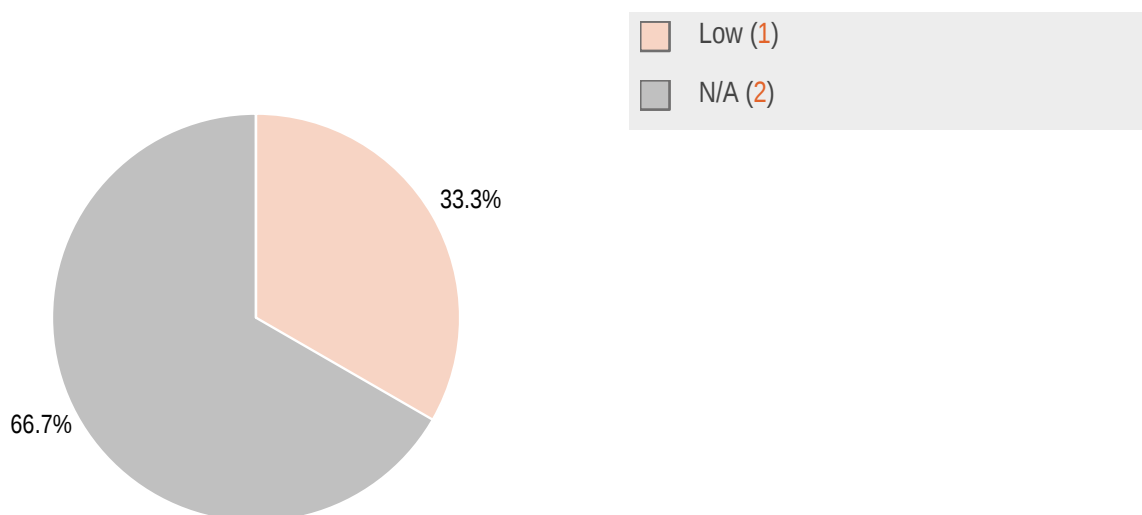
The first N/A finding [CLN-001](#) (page 12) concerns dependencies marked as unmaintained. This poses no immediate threat, but we reported it to place these dependencies as issues on the radar.

The second N/A finding in [CLN-004](#) (page 10) is a note that on Linux the `CAP_NET_ADMIN` capability is requested by Sniffnet but is not in fact required. `CAP_NET_ADMIN` grants substantial control over the system, and therefore *not* having this capability would be a significant win for the principle of least privilege.

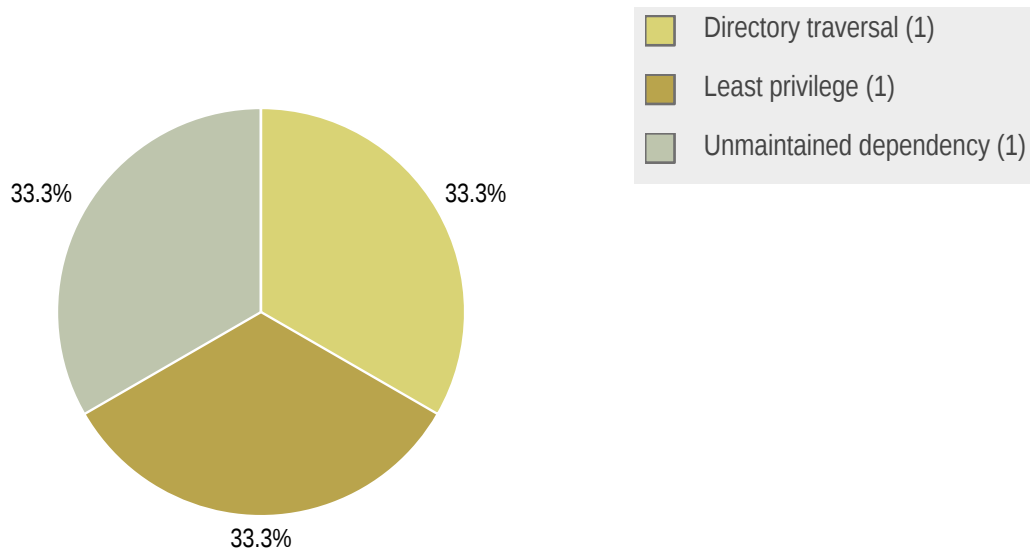
## 1.6 Summary of Findings

ID	Type	Description	Threat level
<a href="#">CLN-002</a>	Directory traversal	The file name field allows directory traversal outside the selected directory.	Low
<a href="#">CLN-004</a>	Least privilege	Sniffnet does not require the <code>CAP_NET_ADMIN</code> capability to function on Linux.	N/A
<a href="#">CLN-001</a>	Unmaintained dependency	The instant and paste crates are used in the build, but they are no longer maintained.	N/A

### 1.6.1 Findings by Threat Level



## 1.6.2 Findings by Type



## 1.7 Summary of Recommendations

ID	Type	Recommendation
CLN-002	Directory traversal	<ul style="list-style-type: none"><li>Filter file names so that they can't be used for directory traversal.</li></ul>
CLN-004	Least privilege	<ul style="list-style-type: none"><li>Amend the post install script and documentation to only set <code>CAP_NET_RAW</code> and not <code>CAP_NET_ADMIN</code> on the Sniffnet executable.</li></ul>
CLN-001	Unmaintained dependency	<ul style="list-style-type: none"><li>Replace the dependencies by updating or finding alternatives.</li></ul>

## 2 Methodology

### 2.1 Planning

Our general approach during code audits is as follows:

1. **Static Code Analysis**

We performed static analysis on the Rust code with `cargo geiger` and `cargo audit`. We performed dynamic testing by running existing unit tests with `miri`, an experimental Rust interpreter that can be used to detect various types of bugs (often in unsafe Rust).

2. **Code Analysis and Fuzzing**

Manual code review and static code analysis were accompanied by the development of fuzzing targets and developing entirely new targets. We used `cargo fuzz` and designed our targets to conform to the `libFuzzer` interface, which can be used with a variety of fuzzers including `AFL++`.

While reading the code we would occasionally spot interesting code paths and develop small stubs to exercise them. These stubs were then used with stepwise debuggers to understand how the code operates at runtime in greater depth.

### 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 Findings

We have identified the following issues:

### 3.1 CLN-002 — Directory traversal through file name field

**Vulnerability ID:** CLN-002

**Vulnerability type:** Directory traversal

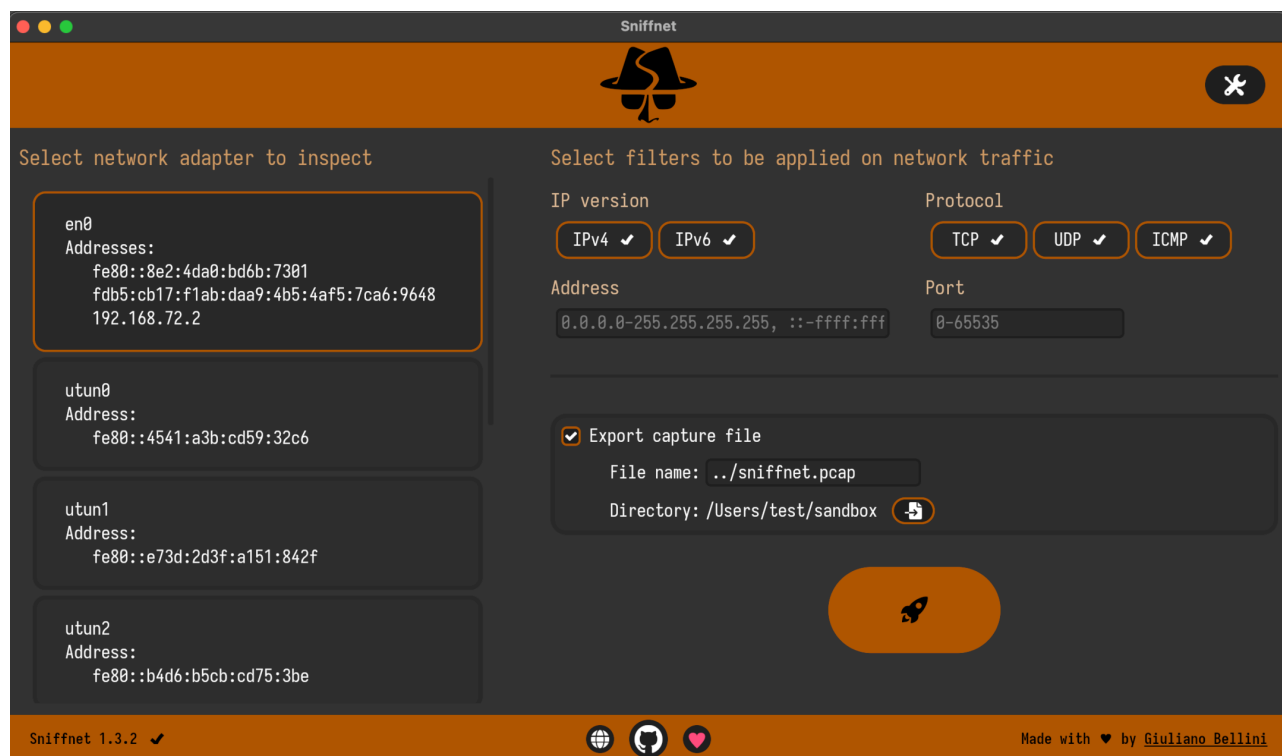
**Threat level:** Low

#### Description:

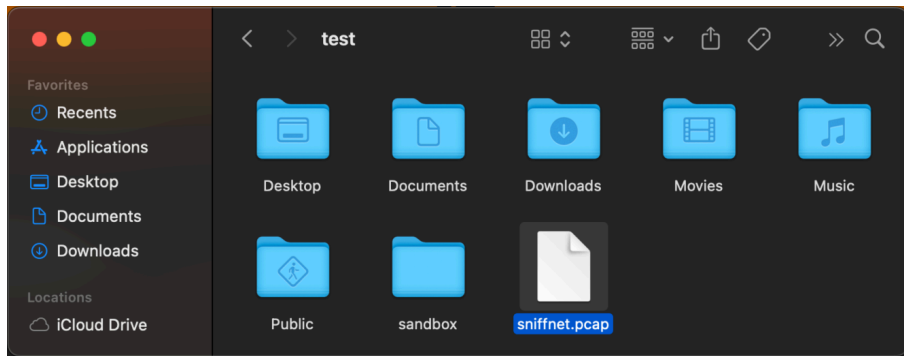
The file name field allows directory traversal outside the selected directory.

#### Technical description:

Sniffnet supports saving captured traffic to a PCAP file for further investigation. To provide this functionality a dialog is implemented to select the file path. The file name is a separate file path. It is expected that a file name will reside within the target file path, however the file name field permits directory traversal sequences.



The file name and the path are concatenated together to form the full path and no checks are performed on the canonicalized path. Consequently, the output can be directed to a file outside the target directory selected by the user.



### Impact:

It may be possible to trick a user into supplying a malicious file name for example via copy/pasting from a tutorial. In some cases Sniffnet will be running a privileged user giving it the ability to write to sensitive parts of the file system. If the user can also be convinced to download attack controlled content then the attacker would have partial control over the content of the file. A skilled attacker may be able to get the contents of the PCAP executed, chaining the directory traversal into remote code execution.

### Recommendation:

- Filter file names so that they can't be used for directory traversal.

## 3.2 CLN-004 — CAP\_NET\_ADMIN not required on Linux

**Vulnerability ID:** CLN-004

**Vulnerability type:** Least privilege

**Threat level:** N/A

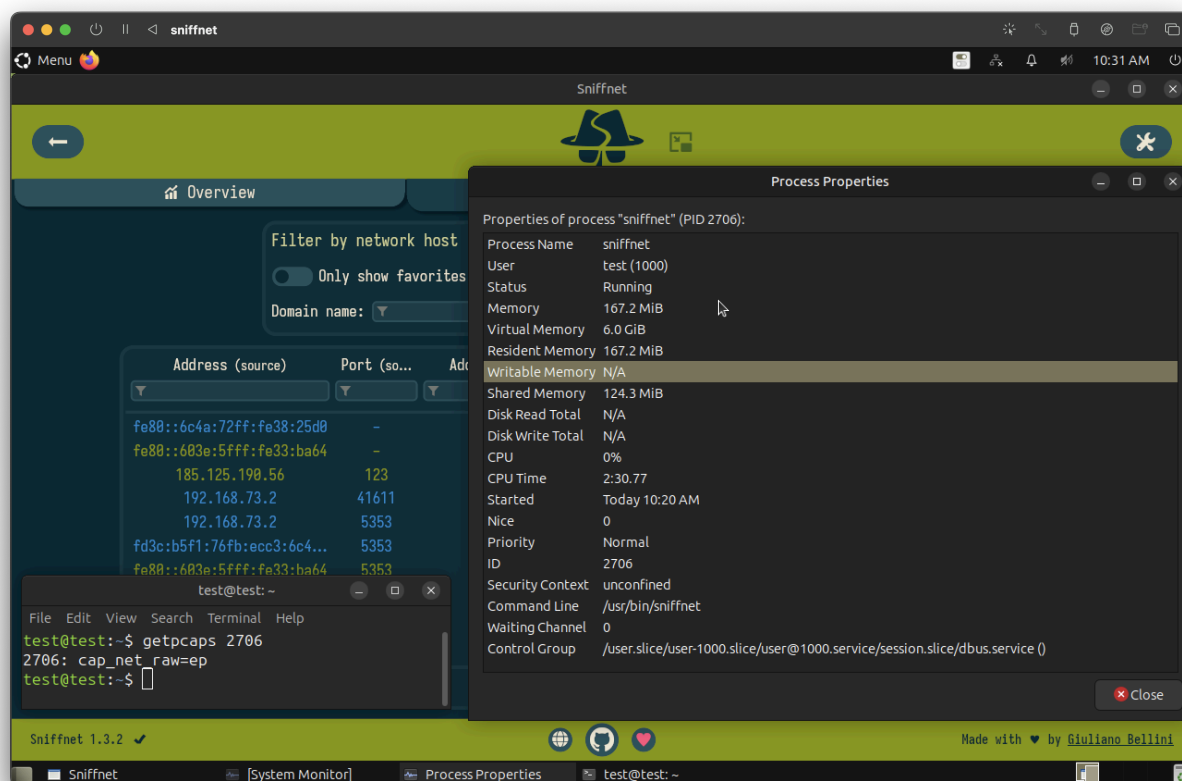
### Description:

Sniffnet does not require the `CAP_NET_ADMIN` capability to function on Linux.

## Technical description:

Sniffnet performs packet capture for the purpose of monitoring traffic to and from the host on which it is running. Sniffnet is therefore not interested in seeing traffic not destined for the host, hence it does not set the capture device to promiscuous mode, as other packet capture tools do. In this situation, the `CAP_NET_ADMIN` capability is not needed.

This screenshot demonstrates `CAP_NET_RAW` being sufficient to use Sniffnet.



## Impact:

An attacker able to take control of the Sniffnet process would be granted the `CAP_NET_ADMIN` capability. This is a powerful capability, allowing for example firewall and routing table changes. This additional privilege makes it a more interesting code injection target for local privilege escalation.

## Recommendation:

- Amend the post install script and documentation to only set `CAP_NET_RAW` and not `CAP_NET_ADMIN` on the Sniffnet executable.

## 3.3 CLN-001 — Unmaintained dependencies

**Vulnerability ID:** CLN-001

**Vulnerability type:** Unmaintained dependency

**Threat level:** N/A

## Description:

The instant and paste crates are used in the build, but they are no longer maintained.

## Technical description:

Cargo audit highlights two crates as being unmaintained. The scan was conducted on commit `8602f90a4114fa9c2c2a59e984157458729ce04a`. The output also details the path through which the dependencies are included:

```
Crate:      instant
Version:    0.1.13
Warning:    unmaintained
Title:      `instant` is unmaintained
Date:       2024-09-01
ID:         RUSTSEC-2024-0384
URL:        https://rustsec.org/advisories/RUSTSEC-2024-0384
Dependency tree:
instant 0.1.13
├── parking_lot_core 0.8.6
│   └── parking_lot 0.11.2
│       └── wasm-timer 0.2.5
│           └── iced_futures 0.13.2
│               ├── iced_winit 0.13.0
│               │   ├── iced 0.13.1
│               │   │   └── sniffnet 1.3.2
│               ├── iced_runtime 0.13.2
│               │   ├── iced_winit 0.13.0
│               │   └── iced_widget 0.13.4
│               │       ├── plotters-iced 0.11.0
│               │       │   └── sniffnet 1.3.2
│               │       └── iced 0.13.1
│               └── iced_graphics 0.13.0
│                   ├── plotters-iced 0.11.0
│                   └── iced_winit 0.13.0
```

```

└── iced_wgpu 0.13.5
    ├── iced_renderer 0.13.0
    │   ├── iced_widget 0.13.4
    │   └── iced 0.13.1
    ├── iced_tiny_skia 0.13.0
    │   ├── iced_renderer 0.13.0
    │   └── iced_renderer 0.13.0
    └── iced 0.13.1
└── parking_lot 0.11.2

Crate:      paste
Version:    1.0.15
Warning:    unmaintained
Title:      paste - no longer maintained
Date:       2024-10-07
ID:         RUSTSEC-2024-0436
URL:        https://rustsec.org/advisories/RUSTSEC-2024-0436
Dependency tree:
paste 1.0.15
└── metal 0.27.0
    ├── wgpu-hal 0.19.5
    │   ├── wgpu-core 0.19.4
    │   │   └── wgpu 0.19.4
    │   │       ├── iced_wgpu 0.13.5
    │   │       │   ├── iced_renderer 0.13.0
    │   │       │   │   ├── iced_widget 0.13.4
    │   │       │   │   │   ├── plotters-iced 0.11.0
    │   │       │   │   │   │   ├── sniffnet 1.3.2
    │   │       │   │   │   │   └── iced 0.13.1
    │   │       │   │   │   │       └── sniffnet 1.3.2
    │   │       │   │   └── iced 0.13.1
    │   │       │   └── iced_glyphon 0.6.0
    │   │       │       └── iced_wgpu 0.13.5
    │   │       └── wgpu 0.19.4
    └── wgpu 0.19.4

warning: 2 allowed warnings found

```

## Impact:

If bugs are found in these dependencies it is unlikely that they will be fixed and a new release published, potentially resulting in the unfixed bugs being exposed in Sniffnet.

## Recommendation:

In these case the unmaintained dependencies are included several sub-dependencies deep. The `instant` dependency can be removed by updating the direct dependency on `parking_lot` to `0.12.3` (latest at time of writing) and waiting for the next release of `iced` containing this change <https://github.com/iced-rs/iced/commit/6a584af1411b713fab52400712031bfa82bccd18>, which removed the unmaintained dependency `wasm-timer` which brings in the older `parking_lot` version and in turn includes `instant`.

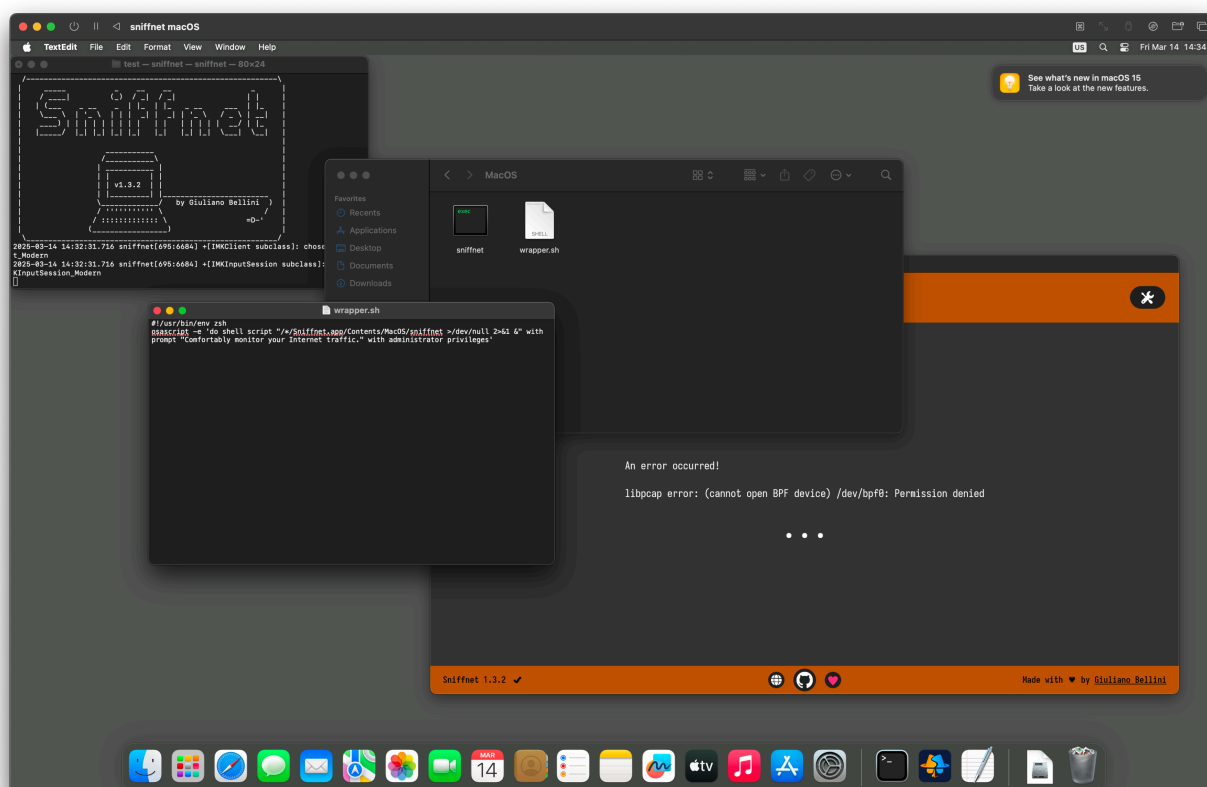
`paste` has only relatively recently been marked unmaintained by the author. A replacement for its functionality would need to be implemented in the `meta1` crate. It is likely that `wgpu` will opt to replace `meta1` with `objc2-meta1` in a future version. At this moment in time we suggest simply waiting for a new release of `wgpu` and the `iced` crates.

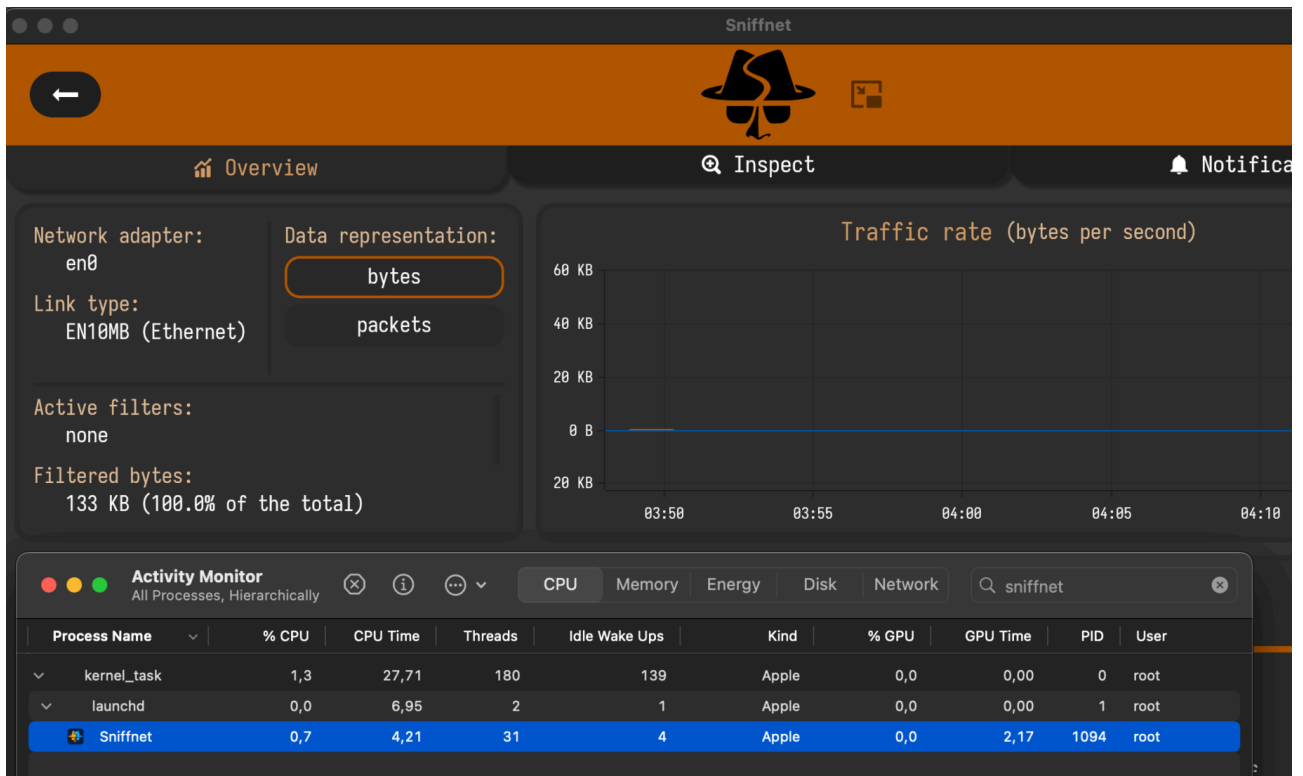
## 4 Non-Findings

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

### 4.1 NF-006 — MacOS privilege reduction

The macOS package of Sniffnet comes with an AppleScript wrapper that prompts the user to authorize running the application as an administrator to run as root. This means the entire Sniffnet app is running as root on macOS.





An alternative approach that is used by other packet capture software on macOS is to install a launchd script that chowns the BPF devices allowing for non-root group based access. The underlying libpcap library provides the bases for these launchd scripts, which are also used by Wireshark. More information can be found here: <https://github.com/the-tcpdump-group/libpcap/blob/master/doc/README.macros>.

By switching to chownBPF Sniffnet would no longer be run as a privileged root process. Users with the correct group membership would then also have access to the BPF devices without re-authentication. In general, the argument is that the reduction in privileged attack surface is more valuable for security than the authentication barrier for accessing BPF devices.

## 4.2 NF-005 — Popular Linux distros don't supply ambient ptrace

In CLN-004 (page 10) we note that Sniffnet could be an interesting code injection target due to its additional capabilities. It is worth noting that modern mainstream Linux distributions, such as our Ubuntu 24.04 test system, are configured to thwart trivial attempts to inject code into running processes.

For example we can't simply attach a debugger without having the ptrace capability. We could run Sniffnet as a child of the debugger however then it would not have had the `CAP_NET_ADMIN` capability we are interested in, as the debugger does not have this capability and without `CAP_SETFCAP/CAP_SETPCAP` the debugger can't give Sniffnet the capability. Also, if we controlled a process with `CAP_SETFCAP/CAP_SETPCAP` then there would be no need to target Sniffnet for local privilege escalation.

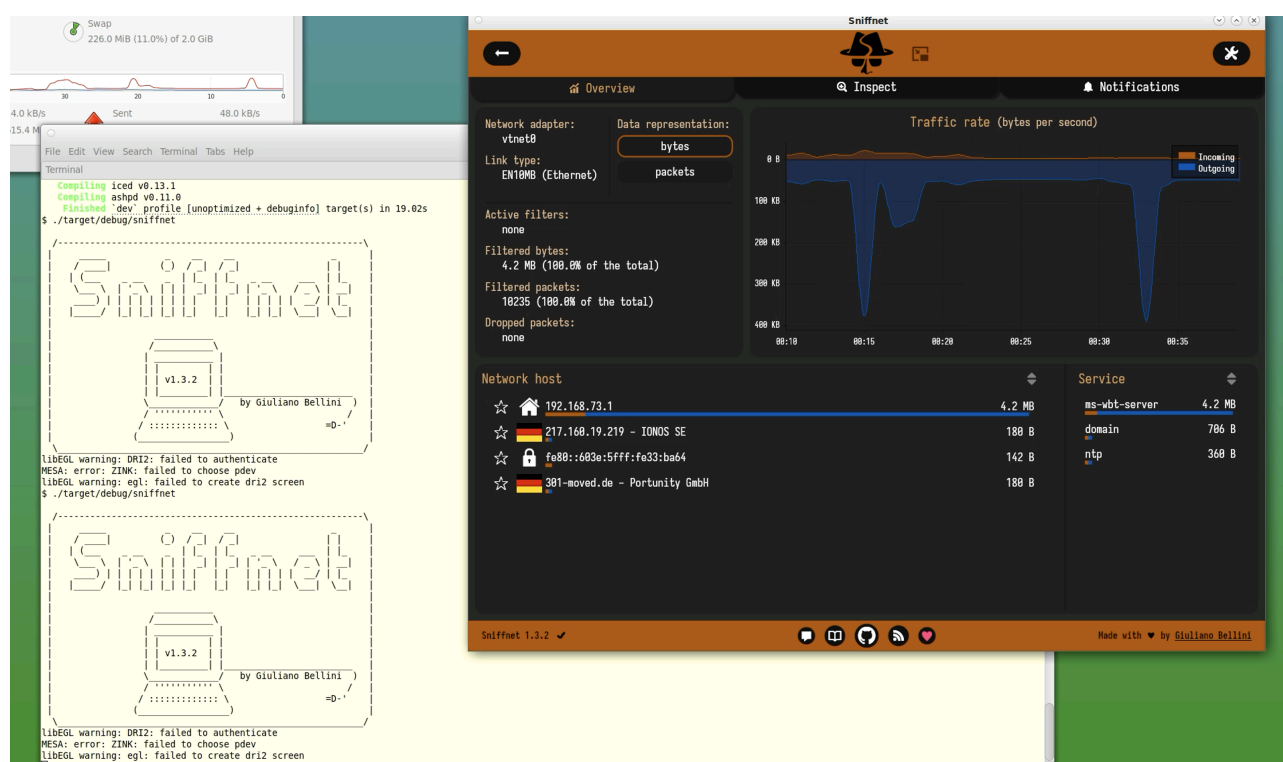


```
test@test:~$ lldb -p 2706
(lldb) process attach --pid 2706
error: attach failed: The current value of ptrace_scope is 1, which can cause ptrace to fail to attach to a running process. To fix this, run:
    sudo sysctl -w kernel.yama.ptrace_scope=0
For more information, see: https://www.kernel.org/doc/Documentation/security/Yama.txt.
(lldb)
```

### 4.3 NF-007 — FreeBSD privilege reduction

We tested on FreeBSD 14 against a Sniffnet build from `main` at `38dedec35af42de6c9d96550534cdfe1a11509a5`.

`libpcap` uses the same `/dev/bpf` interface as macOS to perform packet capture.



Running Sniffnet as root is required by default. For testing purposes the test user was part of the `wheel` group (i.e. an administrator) which owns the BPF devices. The quickest solution to avoid running as root was to allow members of the wheel group access to the BPF devices. It should be noted that this permissions change is ephemeral and needs to be applied on each boot.

```
# chmod g+rw /dev/bpf
```

Wireshark's FreeBSD package provides instructions for making the permissions changes apply automatically each boot, substituting the `wheel` group for the `network` group:

```
--
In order for wireshark be able to capture packets when used by unprivileged
user, /dev/bpf should be in network group and have read-write permissions.
For example:

# chgrp network /dev/bpf*
# chmod g+r /dev/bpf*
# chmod g+w /dev/bpf*

In order for this to persist across reboots, add the following to
/etc/devfs.conf:

own  bpf*  root:network
perm bpf*  0660
"
```

## 4.4 NF-008 — Update checker just checks

The update checker polls the GitHub release API via HTTPS to check if a new release of Sniffnet is available. The polling takes place in a secondary thread. When a newer version is detected, a mutex-protected boolean is set, so that on the next view draw, a button directing the user to get the latest version from the website is rendered in the application's footer. This is simple and does not significantly increase attack surface. No self-update functionality is provided it is up to the user to fetch, verify, and install the update themselves.

## 4.5 NF-011 — Minimal UI rendering attack surface

The only remotely controllable text rendered in the app is the PTR record or the IP address. This could be controlled by an attacker upstream on the network to return malicious content. However, Sniffnet uses iced rather than any web-technology-based UI framework so there is no opportunity for code injection. The only surface exposed is text rendering of a Rust String which by definition must be valid Unicode and is therefore unlikely to present any exploitable vulnerabilities.

## 5 Future Work

- **Drop capabilities on Linux in non pcap threads**

Currently, the entirety of Sniffnet runs with the capabilities `CAP_NET_RAW`, `CAP_NET_ADMIN` which are necessary in order to observe all traffic arriving at or departing from a system. These capabilities, in particular `CAP_NET_ADMIN`, permit a wide range of privileged actions. Only the thread setting up the packet capture requires these capabilities, which could therefore be dropped in the other threads, e.g. the updater and the GUI, reducing the impact of bugs in them.

- **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 a process that must be continuously evaluated and improved; this penetration test is just a single snapshot. Regular audits and ongoing improvements are essential in order to maintain control of your corporate information security.

## 6 Conclusion

We discovered 1 Low and 2 N/A-severity issues during this penetration test.

In this audit we looked at Sniffnet, a traffic analysis tool aimed at providing users with insight into how much data their system is exchanging and with whom. Sniffnet does not concern itself with the content of packets, but rather providing metadata such as ASN and geolocation for their source and destination.

There are only three findings in this report. One low severity directory traversal bug with an extremely low probability of exploitation [CLN-002](#) (page 9). There are two N/A issues, the first covering some unmaintained dependencies [CLN-001](#) (page 12), and the second an opportunity to further reduce privilege on Linux [CLN-004](#) (page 10). It is a good sign that this audit has not uncovered anything with significant impact.

Having a small attack surface is a nice characteristic of Sniffnet. It provides its functionality mostly via offline databases, only taking what it needs from the incoming traffic, and making reverse DNS lookups to provide hostnames for IPs. This information is then presented without any markup languages involved, cutting off the opportunity for any code injection.

Overall we are pleased with the security posture of Sniffnet and the engagement of the developer in this process. We are grateful to NLnet for supporting this work via the NGIO Commons fund and look forward to seeing its future development.

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

Morgan Hill ( <i>pentester</i> )	Morgan is a seasoned security consultant with a background in IoT and DevOps. He currently specialises in Rust and AVoIP.
Melanie Rieback ( <i>approver</i> )	Melanie Rieback is a former Asst. Prof. of Computer Science from the VU, who is also the co-founder/CEO of Radically Open Security.

Front page image by Slava (<https://secure.flickr.com/photos/slava/496607907/>), "Mango HaX0ring",  
Image styling by Patricia Piolon, <https://creativecommons.org/licenses/by-sa/2.0/legalcode>.