RADICALLY OPEN SECURITY

Penetration Test Report

Rosenpass

V 1.0
Amsterdam, January 4th, 2024
Public

Document Properties

Client	Rosenpass
Title	Penetration Test Report
Target	Rosenpass
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	December 22nd, 2023	Morgan Hill	Initial draft
0.2	December 29th, 2023	Marcus Bointon	Review
1.0	January 4th, 2024	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

Table of Contents

1	**Executive Summary**	**4**
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
2	**Methodology**	**7**
2.1	Planning	7
2.2	Risk Classification	7
3	**Reconnaissance and Fingerprinting**	**9**
4	**Findings**	**10**
4.1	CLN-006 — Denial of service with one byte (RUSTSEC-2023-0077)	10
4.2	CLN-009 — GitLab CI config may temporally expose SSH private key to other users on the host	11
4.3	CLN-010 — GitHub release workflow uses unmaintained action	12
4.4	CLN-011 — GitHub QC workflow uses unmaintained actions from actions-rs	13
4.5	CLN-012 — `RespHello` `Biscuit` and `Auth` fields swapped in implementation versus protocol paper	14
5	**Non-Findings**	**16**
5.1	NF-007 — Logs when key is stale	16
5.2	NF-008 — No command injection possible in RP script	16
6	**Future Work**	**17**
7	**Conclusion**	**18**
Appendix 1	**Testing team**	**19**

1 Executive Summary

1.1 Introduction

Between November 2, 2023 and December 22, 2023, Radically Open Security B.V. carried out a penetration test for Rosenpass.

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:

- Rosenpass

1.3 Project objectives

ROS will perform a penetration test of Rosenpass with Rosenpass developers in order to assess the security of the Rosenpass protocol and implementation. To do so ROS will access the Rosenpass source code on GitHub and guide Rosenpass developers 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 November 2, 2023 and December 22, 2023.

1.5 Results In A Nutshell

During this crystal-box penetration test we found 1 Moderate, 3 Low and 1 N/A-severity issues.

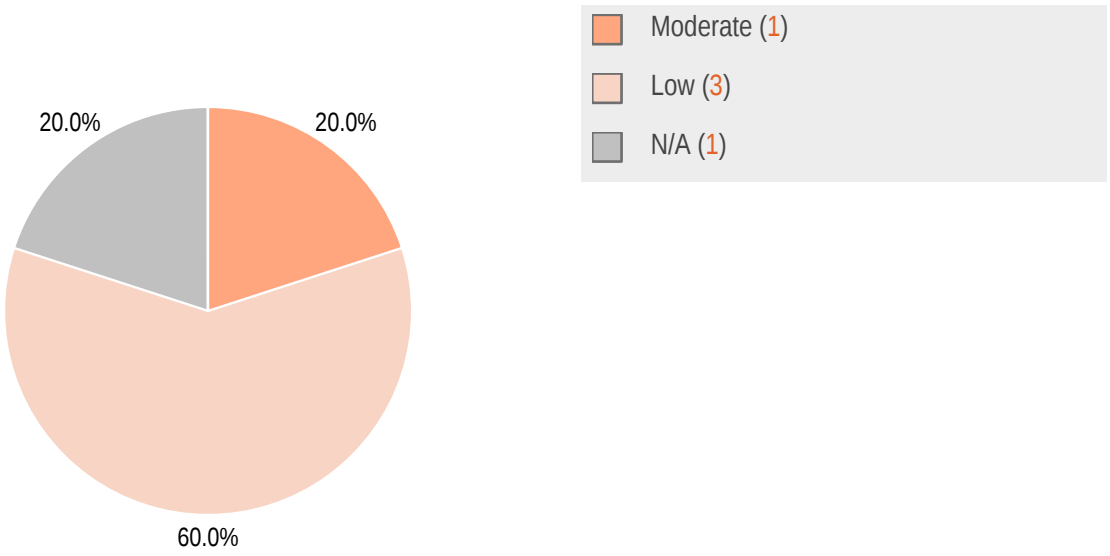
This audit revealed a remotely exploitable DoS condition in the message parsing code CLN-006 (page 10). This issue was fixed in release `0.2.1`. Users running `0.2.0` or earlier will experience a panic if an attacker sends a maliciously crafted UDP datagram to the Rosenpass daemon, resulting in the process exiting.

The other findings were areas of improvement for CI/CD hardening in CLN-009 (page 11), use of unmaintained GitHub actions in CLN-010 (page 12) and CLN-011 (page 13), and a discrepancy in field ordering between the paper and the implementation CLN-012 (page 14).

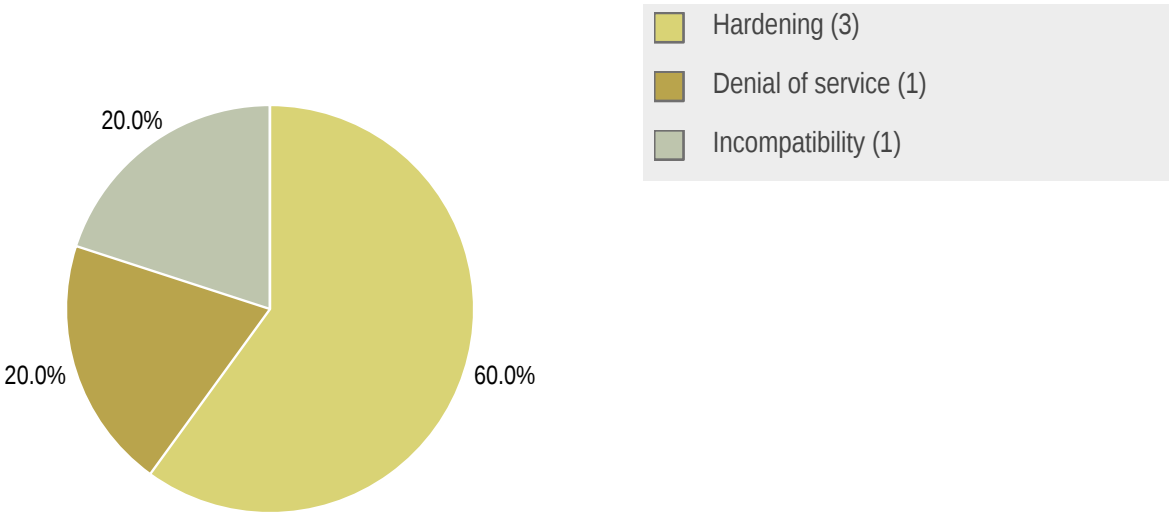
1.6 Summary of Findings

ID	Type	Description	Threat level
CLN-006	Denial of service	The message handler does not validate the length of messages resulting in panics from accessing out of range indices.	Moderate
CLN-009	Hardening	The file mode is set after the file is written, leaving a short period when the files may have less-strict permissions.	Low
CLN-010	Hardening	The softprops/action-gh-release action used in the release workflow is unmaintained.	Low
CLN-011	Hardening	The actions-rs/audit-check GitHub action used in QC workflow is unmaintained.	Low
CLN-012	Incompatibility	The paper suggests the order of the fields is Biscuit then Auth but the implementation does the reverse.	N/A

1.6.1 Findings by Threat Level



1.6.2 Findings by Type



1.7 Summary of Recommendations

ID	Type	Recommendation
CLN-006	Denial of service	• Check message length against what is expected before processing it any further.
CLN-009	Hardening	• Run `umask 077` before writing the SSH private key to the filesystem
CLN-010	Hardening	• Find a maintained alternative. • Regularly check that the actions used are maintained and up to date. • For high assurance, fork and review all changes to actions used.
CLN-011	Hardening	• Find a maintained alternative. • Regularly check that the actions used are maintained and up to date. • For high assurance, fork and review all changes to actions used.
CLN-012	Incompatibility	• Update the documentation in the paper to match the implementations.

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 – http://nmap.org

4 Findings

We have identified the following issues:

4.1 CLN-006 — Denial of service with one byte (RUSTSEC-2023-0077)

Vulnerability ID: CLN-006	**Status:** Resolved
Vulnerability type: Denial of service	
Threat level: Moderate	

Description:

The message handler does not validate the length of messages resulting in panics from accessing out of range indices.

Technical description:

The `rosenpass::protocol::CryptoServer::handle_msg` takes an arbitrary string of bytes and attempts to decode a Rosenpass protocol message from it. It first uses the first byte to determine what type of message it is as described; this outer protocol layer is referred to as the envelope. Once the message type is established, an attempt is made to decode the message in the envelope's payload. The length of the payload is not checked against the expected length of the message type which results in a panic when the code attempts to access out-of-range memory.

In the Rosenpass daemon, `handle_msg` receives raw data straight from a UDP socket, making this behaviour remotely exploitable.

Sending a payload to the daemon over the network:

```
[morgan@t480 rosenpass]$ nc -u ::1 10001 < fuzz/artifacts/handle_msg/crash-031d97a21295bfba24a7fd69d690815
beec90030
```

The daemon crashing:

```
[morgan@t480 rosenpass]$ cargo run exchange-config config-test.toml
   Compiling rosenpass v0.2.0 (/home/morgan/work/ROS/rosenpass)
    Finished dev [unoptimized + debuginfo] target(s) in 2.37s
     Running `target/debug/rosenpass exchange-config config-test.toml`
thread 'main' panicked at src/msgs.rs:228:1:
range end index 144 out of range for slice of length 4
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
```

An example payload:

```
00000000: 84                                       .
```

Impact:

An adversary can send small UDP datagrams to a target or targets resulting in Rosenpass crashing. Rosenpass crashing means that key exchange won't be taking place so the PSK may become stale, or never complete successfully at all. By default, user data would continue to be passed by the Wireguard interface. In non-finding NF-007 (page 16) we confirmed that a key becoming stale is logged, but it is however up to the user to monitor and decide what to do if there is no PSK or the PSK is stale.

Recommendation:

- Check message length against what is expected before processing it any further.

4.2 CLN-009 — GitLab CI config may temporally expose SSH private key to other users on the host

Vulnerability ID: CLN-009

Vulnerability type: Hardening

Threat level: Low

Description:

The file mode is set after the file is written, leaving a short period when the files may have less-strict permissions.

Technical description:

GitLab CI doesn't set the file-creation mask mode before writing the SSH key to the file system from an environment variable.

```
before_script:
    - mkdir ~/.ssh/
    - echo "$SSH_KNOWN_HOSTS" > ~/.ssh/known_hosts
    - echo "$REPO_SSH_KEY" > ~/.ssh/id_ed25519
    - chmod 600 --recursive ~/.ssh/
    - git config --global user.email "ci@gitlab.com"
    - git config --global user.name "CI"
```

https://github.com/rosenpass/rosenpass/blob/main/.gitlab-ci.yml

Impact:

The SSH key is potentially exposed to other users on the file system for a short time. An attacker with persistence as another on the CI runner could monitor for and take advantage of this momentary exposure to steal the private key.

Recommendation:

- Run `umask 077` before writing the SSH private key to the filesystem

4.3 CLN-010 — GitHub release workflow uses unmaintained action

Vulnerability ID: CLN-010

Vulnerability type: Hardening

Threat level: Low

Description:

The `softprops/action-gh-release` action used in the release workflow is unmaintained.

Technical description:

Rosenpass uses GitHub actions to automate common operations on the project, such as creating a release. GitHub actions can include shared third-party automations for common tasks. In this case, Rosenpass has a release workflow that uses a 3rd-party action to create GitHub releases, but this action does not appear to be actively maintained.

The action: https://github.com/softprops/action-gh-release.

At the time of reporting there has been no release of this action for a year, and no commits to the repository since March 2023. There are also 26 open PRs, some of which have comments enquiring about whether the project is alive, suggesting the project is no longer maintained.

Impact:

Actions have a privileged position in project CI/CD pipelines. As such, they could leak credentials such as keys and tokens, or even inject vulnerabilities into the produced artifacts. In the case of Rosenpass, the project does not yet

provide binary release artifacts for releases. It also appears that this action is unmaintained rather than malicious so the main concern is vulnerabilities being found in the action and not being addressed.

Recommendation:

- Find a maintained alternative.
- Regularly check that the actions used are maintained and up to date.
- For high assurance, fork and review all changes to actions used.

4.4 CLN-011 — GitHub QC workflow uses unmaintained actions from actions-rs

Vulnerability ID: CLN-011

Vulnerability type: Hardening

Threat level: Low

Description:

The `actions-rs/audit-check` GitHub action used in QC workflow is unmaintained.

Technical description:

Similar to CLN-010 (page 12), the QC workflow uses an unmaintained action.

Action: https://github.com/actions-rs/audit-check.

All actions-rs actions have been archived and were not maintained for a period before that.

Impact:

Vulnerabilities found in the unmaintained action or it's dependencies will not be fixed. This could compromise the integrity of the build system. See CLN-010 (page 12).

Recommendation:

- Find a maintained alternative.
- Regularly check that the actions used are maintained and up to date.

- For high assurance, fork and review all changes to actions used.

4.5 CLN-012 — `RespHello` `Biscuit` and `Auth` fields swapped in implementation versus protocol paper

Vulnerability ID: CLN-012

Vulnerability type: Incompatibility

Threat level: N/A

Description:

The paper suggests the order of the fields is `Biscuit` then `Auth` but the implementation does the reverse.

Technical description:

Rosenpass messages in the Rust implementation are serialized and de-serialized using lenses which allow zero copy operations by interacting with slices of the underlying buffer array. For convenience, these lenses are implemented using a declarative macro that allows accessing message fields via their names. This macro relies on the order of the fields in its arguments to determine their offsets within the buffer.

In the case of the `RespHello` message, the order of fields is different to how the message is documented in the paper. The implementation has `Auth` before `Biscuit` whereas the paper has `Biscuit` before `Auth`. This would create incompatibility with other implementations that follow the paper.

Here is a test demonstrating the expected order versus what is implemented:

```
mod test_resp_hello {
    use crate::msgs::{RespHelloExt};

    #[test]
    fn doc_msg_order() {
        let mut a = [0u8;1096];
        a[4+4+768+188] = 1; // sidr + sidi + ecti + scti + biscuit + auth
        assert_eq!(a.resp_hello().unwrap().biscuit()[0], 1u8);
    }

    #[test]
    fn impl_msg_order() {
        let mut a = [0u8;1096];
        a[4+4+768+188+16] = 1; // sidr + sidi + ecti + scti + auth + biscuit
        assert_eq!(a.resp_hello().unwrap().biscuit()[0], 1u8);
    }
}
```

The Go implementation constructs this message in the same way as the Rust implementation: https://github.com/cunicu/go-rosenpass/blob/main/messages.go#L175, suggesting this misordering relative to the docs is a de-facto standard.

Impact:

As all current implementations use the same ordering there is little practical impact. The MAC of the message would have to be valid in order for the fields to be processed.

Recommendation:

- Update the documentation in the paper to match the implementations.

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-007 — Logs when key is stale

It is important to ensure there is some record that a key is stale if a peer goes down, is taken down, or the Rosenpass communication is disrupted. This is important as the user should be made aware that their tunnel no longer has the same assurances, and they can either fix the issue, continue using it with accepted risk, or shut it down. A log message is sufficient for this purpose.

We confirmed that a message is logged when a stale key is encountered:

```
output-key peer iExl8/R3RrfDqQHfTih6B0mceHoQmieYSs7YJihXs6I= key-file "peer-a-rp-out-key" exchanged
output-key peer iExl8/R3RrfDqQHfTih6B0mceHoQmieYSs7YJihXs6I= key-file "peer-a-rp-out-key" exchanged
output-key peer iExl8/R3RrfDqQHfTih6B0mceHoQmieYSs7YJihXs6I= key-file "peer-a-rp-out-key" stale
```

5.2 NF-008 — No command injection possible in RP script

All the input is escaped correctly using `printf '%q' $var`, so it was not possible to escape the escaping and inject commands.

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 1 Moderate, 3 Low and 1 N/A-severity issues during this penetration test.

Rosenpass is a relatively young project aiming to provide a production-ready Post-Quantum Wireguard VPN solution. An unbiased audit was seen as an appropriate step to assess its security as adoption increases and the impact becomes higher.

This engagement was principally carried out by 1 pentester, with support from others at ROS, in collaboration with the Rosenpass developers.

The bulk of the engagement was conducting a line-by-line code audit generating several leads which were then subjected to further interrogation. Owing to the generally high code quality, and the safety characteristics of the Rust programming language, most of these leads could be disregarded.

The moderate severity issue is a DoS condition (RUSTSEC-2023-0077) found by fuzzing the message parsing code. The bug itself is concealed in a macro that implements zero-copy field accessors based on byte offsets inside the message buffer, which is missing length checks. In Rust the attempted buffer overread results in a panic; in other languages this could have been a more severe issue. This message parsing code is hooked directly to a UDP socket, meaning the DoS condition could be triggered remotely. The issue was identified and fixed promptly by the developers, and the fix was then tested successfully by the pentester. The fuzz target used to discover this issue was contributed to Rosenpass along with other fuzz targets created as part of the engagement.

There was also an element of discussion between the developers and ROS around threat modeling and the protocol itself.

The overall impression from this engagement is of a responsive and engaged project utilizing modern tools to achieve their aims. Our recommendation is to take a look at hardening CI/CD starting with the 3 low severity issues in this report, and continue on the current track with the thoughtful consideration of improvements to the protocol and implementation.

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	Morgan is a seasoned security consultant with a strong background in DevOps and IoT. He played a pivotal role in designing and implementing significant portions of Holoplot's professional audio products, which are prominently used at the MSG Sphere in Las Vegas. His expertise in media security was showcased when he presented at the MCH2022 event. Morgan's exceptional performance in the field is further demonstrated by his position on the SANS Advisory Board, where he achieved a high score and emerged victorious in the CTF in SEC-488. Aside from his IT accomplishments, Morgan has also made substantial contributions to the rail sector. He successfully orchestrated the delivery of new signaling schemes and station remodeling, granting him a unique perspective on Operational Technology. Currently, he is committed to utilizing his innovative mindset and skill set to elevate the security landscape.
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.