RADICALLY OPEN SECURITY

Penetration Test Report

Liminix

V 1.0
Amsterdam, November 7th, 2024
Confidential

Document Properties

Client	Liminix
Title	Penetration Test Report
Target	Liminx
Version	1.0
Pentesters	Thomas Rinsma, Edoardo Geraci
Authors	Thomas Rinsma, Edoardo Geraci, Marcus Bointon
Reviewed by	Marcus Bointon
Approved by	Melanie Rieback

Version control

Version	Date	Author	Description
0.1	November 4th, 2024	Thomas Rinsma, Edoardo Geraci	Initial draft
0.2	November 7th, 2024	Marcus Bointon	Review
1.0	November 7th, 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

1.1 Introduction

Between October 7, 2024 and November 7, 2024, Radically Open Security B.V. carried out a penetration test for Liminix.

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

1.2 Scope of work

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

- Liminx

The scoped services are broken down as follows:

- Pentesting (incl. reporting): 40 hours
- **Total effort: 5 days**

1.3 Project objectives

ROS will perform a penetration test and audit of the Nix based firmware in order to assess the security of the Liminix router firmware for Liminix. To do so ROS will access the code of the Liminix application 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 October 7, 2024 and November 7, 2024.

1.5 Results In A Nutshell

During this crystal-box penetration test we found 1 Moderate and 1 Low-severity issues.

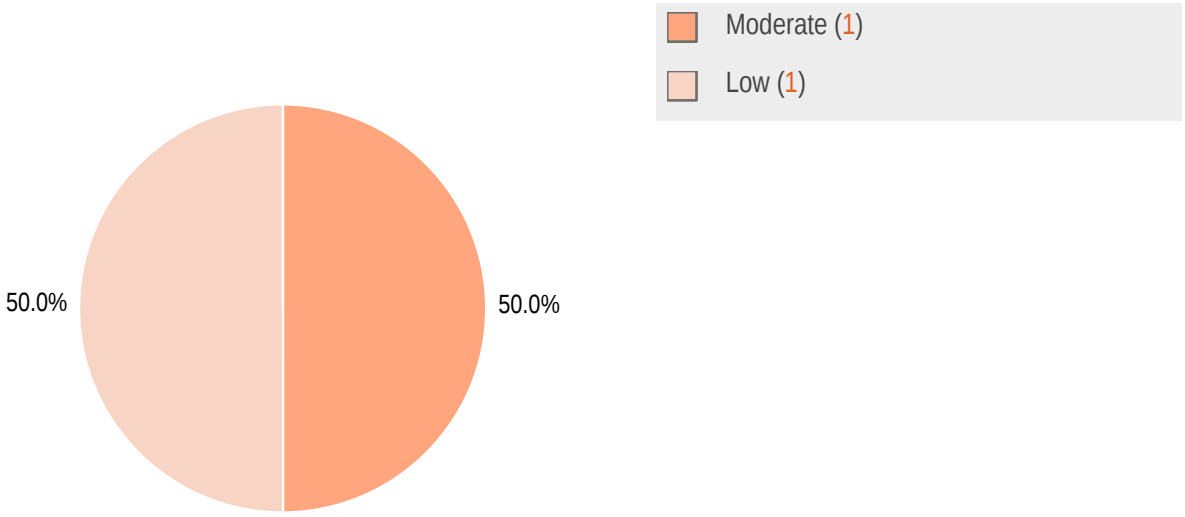
Both issues identified stem from improper handling of user input in path creation, resulting in file disclosure in the TFTP debug server LMN-001 (page 9) and giving a malicious user the ability to write files to a specified, controllable

directory via the provided secrets service LMN-002 (page 10), potentially breaking the availability, confidentiality and integrity of the application.

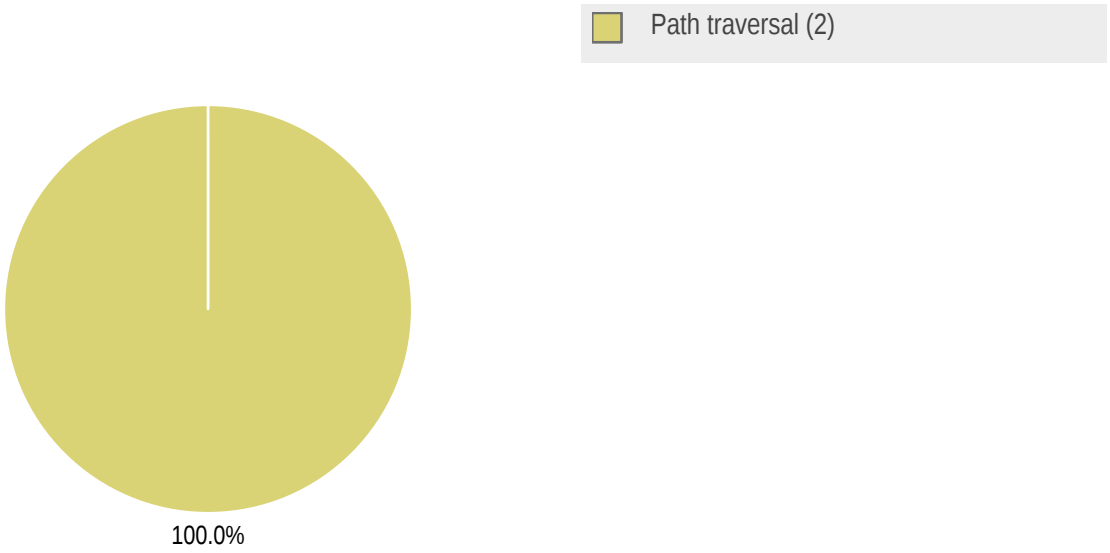
1.6 Summary of Findings

ID	Type	Description	Threat level
LMN-001	Path traversal	The TFTP server provided by the tufted package is vulnerable to path traversal, allowing a malicious user to access every file in the file system.	Moderate
LMN-002	Path traversal	The secrets service package is vulnerable to path traversal, allowing a malicious user with write privileges on the secrets JSON to write in every folder of the file system.	Low

1.6.1 Findings by Threat Level



1.6.2 Findings by Type



1.7 Summary of Recommendations

ID	Type	Recommendation
LMN-001	Path traversal	• Check that the requested file is located within the allowed folder and that no path traversal has occurred.
LMN-002	Path traversal	• Check that the requested file is located within the allowed folder, and that no path traversal has occurred.

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 (2021) 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 Findings

We have identified the following issues:

3.1 LMN-001 — Path traversal in TFTP server

Vulnerability ID: LMN-001

Vulnerability type: Path traversal

Threat level: Moderate

Description:

The TFTP server provided by the `tufted` package is vulnerable to path traversal, allowing a malicious user to access every file in the file system.

Technical description:

The `tufted` package relies on the `merge-pathname` function to generate the path for a requested file. This function uses the `..` operator in Fennel to concatenate the requested filename string and the `base-directory` string to create the file path that will be later used to access the retrieve the file that will be sent to the client.

```
(fn merge-pathname [directory filename]
  (if (directory:match "/$")
      (.. directory  filename) ;; --> string concatenation
      (.. directory "/" filename))) ;; --> string concatenation
```

A malicious client could connect to the TFTP server and request a file, providing a filename such as `../../../../etc/passwd`. Since no check is performed on the final path, a malicious user is able to gain read access to any file in the file system.

Impact:

A malicious user can access any file in the file system, since the TFTP server is running as a root process, breaking confidentiality.

Recommendation:

- Check that the requested file is located within the allowed folder and that no path traversal has occurred.

3.2 LMN-002 — Path traversal in secrets service

Vulnerability ID: LMN-002

Vulnerability type: Path traversal

Threat level: Low

Description:

The secrets service package is vulnerable to path traversal, allowing a malicious user with write privileges on the secrets JSON to write in every folder of the file system.

Technical description:

The service handling the ssh key update relies on the `write-changes` function to create the files containing the users public keys. The `write-changes` function uses the `..` fennel operator to concatenate strings and create a path with user provided input.

```
(fn write-changes [path old-tree new-tree]
  (when (not (table= old-tree new-tree))
    (io.stderr:write "new ssh keys\n")
    (each [username pubkeys (pairs new-tree)]
      (with-open [f (assert (io.open (.. path "/" username) :w))] ;; --> string concatenation
        ;;  the keys are "1" "2" "3" etc, so pairs not ipairs
        (each [_ k (pairs pubkeys)]
          (f:write k)
          (f:write "\n")))))
  (each [k v (pairs old-tree)]
    (when (not (. new-tree k))
      (os.remove (.. path "/" k))))
  new-tree)
```

This allows a malicious user to provide inputs that contain a path traversal payload to gain write access for folders he should not be allowed to write into. A simple example of how a malicious user can exploit such behavior is the following: Given that the malicious user has access to the content of the `secrets` JSON file, they could modify it:

```
"ssh": {
  "authorizedKeys": {
    "../../../../../../tmp/injected":["content that will be written in the file"],
  }
```

```
}
```

And this would create a file in the `/tmp/injected` folder.

Impact:

A malicious user with write privileges on the `secrets` JSON file, is able to write to every folder in the file system.

Recommendation:

- Check that the requested file is located within the allowed folder, and that no path traversal has occurred.

4 Non-Findings

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

4.1 NF-003 — Firewall rules

The Liminix application includes a list of default firewall rules in the `modules/firewall/default-rules.nix` file. Here is some advice on how to clarify these rules and what changes to make to avoid ambiguous behavior. While not vulnerabilities per se, these ambiguities could lead to security issues if misinterpreted by a user.

- The note about the `fib saddr . iif oif eq 0` rule should be improved by adding comments based on matching routing information and reverse path filtering;
- Add rate limiting to ICMP packets on both IPv4 and IPv6 from WAN side;
- Add a reference to the ICMPv6 RFC section explaining the dropping of certain types;
- The `input-ip4-wan` filter contains the `udp sport 53` rule which introduces an ambiguous behavior: accepting UDP packets with source port 53 (DNS) from WAN. Given that the purpose of this rule is to allow the router to receive DNS query replies, we recommend a different approach, such as using connection tracking to allow DHCP responses back to the Liminix instance without introducing raw port rules. Additionally, we suggest adding notes to the firewall rules regarding this matter;
- The duplicated rule `fc00::/7` should be removed for both `saddr` and `daddr`.

4.2 NF-004 — odhcp6c package usage and environment variable handling

While auditing the Liminix project, we encountered the `odhcp6c` package, which provides an embedded DHCPv6 client. This client is used in the `dhcp6c` module, where values from the environment are used to create OS commands that are later executed. Such values can be, to some extent, controlled by external users. An example of OS commands built this way can be found in `modules/dhcp6c/acquire-wan-address.fnl`:

```
(fn update-addresses [wan-device addresses new-addresses exec]
    (each [_ p (ipairs (deletions addresses new-addresses))]
        (exec
        (.. "ip address del " p.address "/" p.len " dev " wan-device))) ;; --> here
[...]
```

Based on the currently supported parameters, no vulnerabilities arise from this behavior. However, if the set of supported environment variables is extended, security issues could emerge. We recommend always avoiding string concatenation when creating OS commands.

4.3 NF-005 — Handling of user inputs in path creation

The main issues in the Liminix app arise from placing complete trust in user inputs used to create paths for reading and writing files in the filesystem. These issues stem from using Fennel's `..` operator, which concatenates strings but does not guarantee any properties of the created path. A vulnerable code example looks like this:

```
(.. base-path  user-provided-input) ;; --> string concatenation
```

This is vulnerable to path traversal attacks if a malicious user controls the final part of the generated string, which is often the case, and can lead to high-severity vulnerabilities. We recommend using a specialized library for path handling, enabling easier verification of paths generated from user inputs, and better enforcement of authorization rules regarding what and where a user can read or write.

5 Future Work

- **Standard dependencies audit**
  The Liminix project relies heavily on standard dependencies for networking services, such as DHCP, PPP, and others. While these are widely used libraries, it is crucial to ensure they are well-audited and up-to-date to prevent security issues. Although auditing these libraries is challenging and time-consuming, vulnerabilities can still exist, making this high-effort task a valuable safeguard.
- **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 Moderate and 1 Low-severity issues during this penetration test.

As expected, given the structure of the Liminix project and the extensive use of standard libraries, no high-severity issues were found. The application's structure is clear, and the primary source of security risks arises from the customization of the Liminix firmware by the end user, while the baseline service remains strong and secure. We recommend adding documentation with guidelines on securely extending this software. Additionally, we recommend a "trust-no-one" approach for all user-provided inputs to minimize security risks as much as possible.

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

Thomas Rinsma	Thomas Rinsma is a security analyst and hobby hacker. His specialty is in application-level software security, with a tendency for finding bugs in open-source dependencies resulting in various CVEs. Professionally he also has experience testing everything from hypervisors to smart meters, but anything with a security boundary to bypass interests him.
Edoardo Geraci	Edoardo Geraci is a cybersecurity enthusiast with focus on web exploitation and an active player of the CTF team fibonhack.
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.