Penetration Test Report

Marcus Rohrmoser Mobile Software

V 1.0
Amsterdam, November 26th, 2024
Public

Document Properties

Client	Marcus Rohrmoser Mobile Software
Title	Penetration Test Report
Targets	Seppo Social Web Server Instance Seppo Web Application Communication between client-server
Version	1.0
Pentesters	Nicolas Jeannerod, Sabrina Deibe
Authors	Sabrina Deibe, Nicolas Jeannerod, Marcus Bointon
Reviewed by	Marcus Bointon
Approved by	Melanie Rieback

Version control

Version	Date	Author	Description
0.1	September 4th, 2024	Sabrina Deibe, Nicolas Jeannerod	Initial draft
0.2	October 30th, 2024	Marcus Bointon	Review
1.0	November 26th, 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 August 19, 2024 and August 30, 2024, Radically Open Security B.V. carried out a penetration test for Marcus Rohrmoser Mobile Software

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):

- Seppo Social Web Server Instance
- Seppo Web Application
- Communication between client-server

The scoped services are broken down as follows:

- Pentest of Seppo Application (incl. reporting): 10 days
- PM/Review: 1 days
- **Total effort: 11 days**

1.3 Project objectives

The objective is to perform a penetration test of the Seppo social application to assess its security. To do so, ROS will perform a security assessment using the STRIDE threat model to identify potential threats within the application's design and employ the OWASP Top Ten methodology for the web application penetration test. Any vulnerabilities discovered will be exploited to evaluate the potential for further access and privilege escalation, providing Marcus Rohrmoser Mobile Software with a comprehensive understanding of security risks and recommendations for remediation.

1.4 Timeline

The security audit took place between August 19, 2024 and August 30, 2024.

1.5 Results In A Nutshell

During this crystal-box penetration test we found 3 Moderate and 7 Low-severity issues.

We did not find any severe issues in the Seppo project, though there are some minor ones.

As is often the case, there are issues surrounding login mechanisms, such as improper session handling providing an opportunity to create unlimited session tokens CLN-011 (page 14), an incomplete logout mechanism CLN-009 (page 13), and a lack of anti-automation on the login form CLN-007 (page 17).

Incomplete input validation can lead to application crashes CLN-010 (page 19). Some error messages contain links to documentation that does not exist CLN-002 (page 15). The application fails to declare some required dependencies, making it more difficult to deploy CLN-005 (page 16), and also fails to declare sufficiently strict version constraints, making reproducible builds harder. Incomplete exception handling can lead to information disclosure through verbose error messages CLN-013 (page 19).

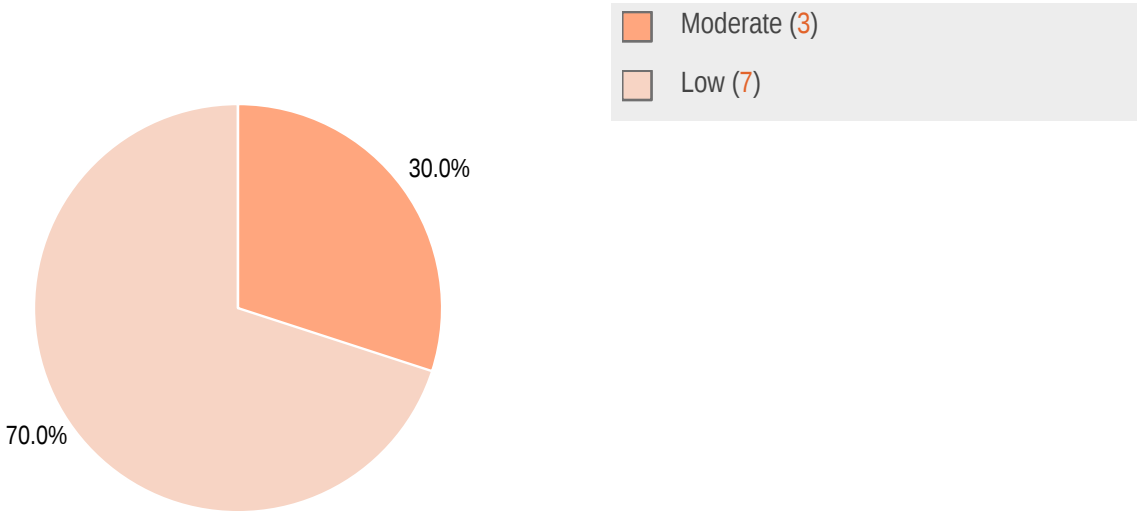
Some file output operations are subject to concurrency problems, possibly resulting in file corruption CLN-003 (page 12), and race conditions in handling PID file locks can result in corrupt, missing, or incorrect file contents CLN-014 (page 20).

1.6 Summary of Findings

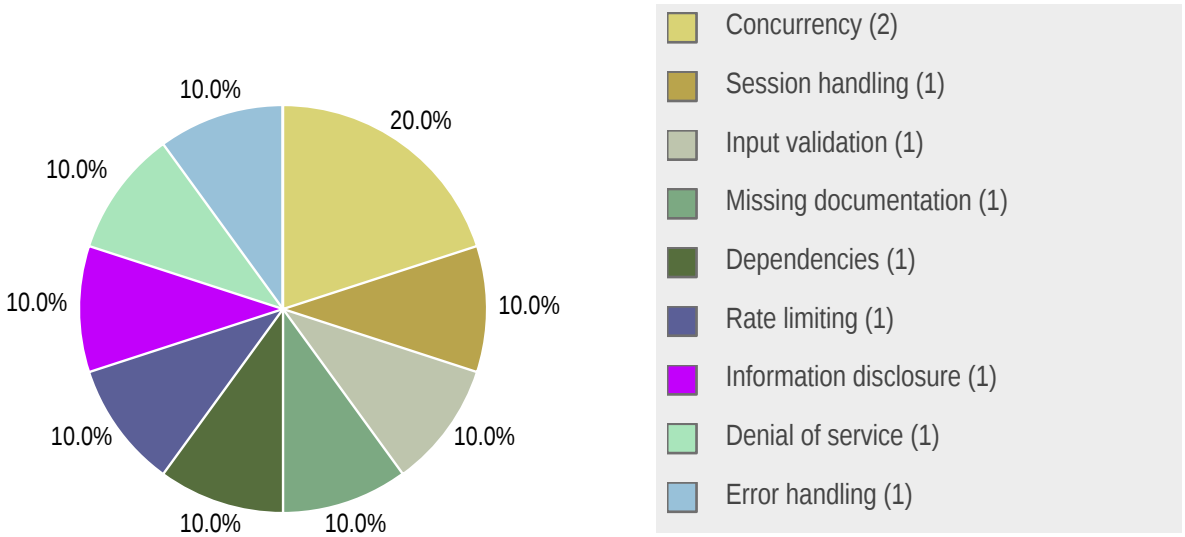
ID	Type	Description	Threat level
CLN-003	Concurrency	The function File.out_channel' can append to an existing file, potentially leading to corrupted files.	Moderate
CLN-009	Session handling	Session cookies storing the user session are replayable and are not invalidated on logout.	Moderate
CLN-011	Input validation	An attacker may spoof a user session by providing unexpected input that makes the server generate unlimited valid session tokens.	Moderate
CLN-002	Missing documentation	Some errors contain error codes and documentation links, but some of these links are broken.	Low
CLN-005	Dependencies	The .opam file is missing two dependencies. crunch and tyre, without which the build cannot succeed. In general, the dependencies are specified without their versions, which might hinder future reproducibility.	Low
CLN-007	Rate limiting	There is no account lockout enforced on the application allowing password guessing attacks. Usernames are public.	Low
CLN-008	Information Disclosure	Internal error messages including detailed code errors and/or stack traces are displayed to the end user.	Low
CLN-010	Denial of Service	Certain injections or unexpected input causes the server to crash and freeze, without allowing any kind of interaction.	Low

CLN-013	Error handling	In various places, rejected Lwt promises can slip through a catch-all try ... with, potentially reaching the top-level.	Low
CLN-014	Concurrency	The PID file lock mechanism seems to work, but the content of the lock file is not guaranteed to be the PID of the process.	Low

1.6.1 Findings by Threat Level



1.6.2 Findings by Type



1.7 Summary of Recommendations

ID	Type	Recommendation
CLN-003	Concurrency	• Check that the file to write to does not exist, or truncate the file when opening, or generate temporary file names randomly to avoid clashes.
CLN-009	Session handling	• Expire the session server-side, not just on the client. • Invalidate session cookies upon change of password, logout, or when the browser is closed. • Implement stringent controls over cookie sessions, using short expiry times.
CLN-011	Input validation	• Invalidate server-side session state correctly. • Validate session cookies injected on the cookie header, and expected behaviour server side. • Implement idle and absolute session timeouts. • Limit login retry attempts to mitigate brute-forcing.
CLN-002	Missing documentation	• Write the missing documentation for the errors. • Ensure survivability of this documentation by, for example, archiving the website and pointing users to the archive in error messages, and/or by moving the error descriptions into Seppo's repository.
CLN-005	Dependencies	• Add the missing dependencies to the `.opam` file. • Establish a policy for the definition of dependencies in a reproducible way, for instance by exporting a working opam switch or by relying on reproducibility-oriented tools. • Test periodically, ideally in an automated way, that one can rebuild Seppo from scratch.
CLN-007	Rate limiting	• Lock accounts after 3 to 5 unsuccessful login attempts.

		• Only unlock accounts after a predetermined period of time, via a self-service unlock mechanism, secret questions, or intervention by an administrator. • Use a CAPTCHA to prevent automated login attempts.
CLN-008	Information Disclosure	• Display only generic error messages to end users. • Log detailed error messages internally. • Establish a policy for handling errors, and ensure that there is a global error handler to catch any errors not specifically intercepted.
CLN-010	Denial of Service	• Implement robust input validation (including size limits) and escaping of special characters in all user inputs.
CLN-013	Error handling	• Rework catch-all places to ensure that either they do not process an `Lwt` promise, or that the rejected `Lwt` promises are processed correctly.
CLN-014	Concurrency	• Change the code to acquire the lock before any modification is made to the file.

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 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-003 — File.out_channel' can append to an existing file

Vulnerability ID: CLN-003

Vulnerability type: Concurrency

Threat level: Moderate

Description:

The function `File.out_channel'` can append to an existing file, potentially leading to corrupted files.

Technical description:

The function `File.out_channel'` can append to an existing file, because it opens the file in question in append mode. In particular, this can make it non-idempotent in situations where the execution is interrupted. For instance:

- Call `File.out_channel' f "file"` a first time. This will create a temporary file `file~` and fill it with whatever `f` does.
- Interrupt the execution before renaming `file~` to `file`.
- `Call File.out_channel' f "file"` another time. This will append to the temporary file `file~`, then rename it to `file`.

Impact:

The function `File.out_channel'` might not be idempotent, breaking the mental model that the developers have of it. In practice, it could lead to files being corrupted by containing the content of two calls to `File.out_channel'` instead of just one.

Recommendation:

Prevent this situation from happening within `File.out_channel'` by checking that the file to write to does not exist, or truncating the file when opening, or by generating temporary file names randomly to avoid clashes.

4.2 CLN-009 — Insecure session handling

Vulnerability ID: CLN-009

Vulnerability type: Session handling

Threat level: Moderate

Description:

Session cookies storing the user session are replayable and are not invalidated on logout.

Technical description:

Session cookies can be maliciously or fraudulently repeated or delayed by any attacker who might intercept the data, have access to session that contains the cookie, or any other mechanism that allows getting hold of the cookie.

After the user logs out, the session cookie is deleted from the client, however, the session remains active and valid on the server side.

On the upside, the `HttpOnly` flag is set correctly on the session cookie, so scripts cannot access its value.

Impact:

Session cookies remain usable even after the user logs out of the application. The cookie is only released from the browser but remains active on the server, which allows for session replay and impersonation. Someone can access / seppo.cgi/passwd and change the credentials of the user that is associated with the session cookie. If an attacker obtains a copy of the session cookie, they will be able to continue to use it after the user thinks they have logged out.

Recommendation:

- Expire the session server-side, not just on the client.
- Invalidate session cookies upon change of password, logout, or when the browser is closed.
- Implement stringent controls over cookie sessions, using short expiry times.

4.3 CLN-011 — Improper input validation results in multiple session tokens

Vulnerability ID: CLN-011

Vulnerability type: Input validation

Threat level: Moderate

Description:

An attacker may spoof a user session by providing unexpected input that makes the server generate unlimited valid session tokens.

Technical description:

On the endpoint `/seppo.cgi/session`, a user is able to force the server to provide multiple session cookies with indefinite validity.

When a user injects different payloads in place of the session cookie, the server creates new session cookies that belong to the same user and are all valid simultaneously. Subsequent calls to `/seppo.cgi/timeline/p/seppo.cgi/seppo.cgi/seppo.cgi` are generated, causing the server crash after some seconds with an error message: `ERR_TOO_MANY_REDIRECTS. Try clearing your cookies`.

Impact:

An adversary that has access to the session cookies is able to impersonate the user by submitting the active session cookie. It is also able to generate new session cookies equally valid.

Recommendation:

- Invalidate server-side session state correctly.
- Validate session cookies injected on the cookie header, and expected behaviour server side.

General session recommendations:

- Implement idle and absolute session timeouts: Terminate sessions that have been inactive for a while (session timeout) and an absolute session timeout requiring reauthentication after a fixed time period (absolute timeout), regardless of activity.

- Limit login retry attempts to mitigate brute-forcing.

4.4 CLN-002 — Missing error documentation

Vulnerability ID: CLN-002

Vulnerability type: Missing documentation

Threat level: Low

Description:

Some errors contain error codes and documentation links, but some of these links are broken.

Technical description:

Some errors have no documentation, e.g. error code `E1005` points to `https://seppo.social/E1005` for explanations, which, at the time of writing, does not exist.

Impact:

This might prevent other users from choosing Seppo, or from contributing. In the far future, the fact that error descriptions are not in the Seppo repository but on another website may lead to these descriptions not being available any more.

Recommendation:

- Write the missing documentation for the errors.
- Ensure survivability of this documentation by, for example, archiving the website and pointing users to the archive in error messages, and/or by moving the error descriptions into Seppo's repository.

4.5 CLN-005 — Missing dependencies

Vulnerability ID: CLN-005

Vulnerability type: Dependencies

Threat level: Low

Description:

The `.opam` file is missing two dependencies. `crunch` and `tyre`, without which the build cannot succeed. In general, the dependencies are specified without their versions, which might hinder future reproducibility.

Technical description:

Seppo specifies dependencies via a `seppo.opam` file in its repository. However, this file fails to mention at least two dependencies, the OCaml libraries `crunch` and `tyre`, that are necessary to build it properly. Additionally, only the name of the dependencies are specified, without version bounds or exact version constraints.

Impact:

Both the missing dependencies and the lack of version bounds hinders reproducibility. It makes it harder for an external person or for our future selves to take on Seppo, build it, and contribute to it. The lack of exact versions makes it near impossible to rebuild the exact same executable.

Recommendation:

- Add the missing dependencies to the `.opam` file.
- Establish a policy for the definition of dependencies in a reproducible way, for instance by exporting a working opam switch or by relying on reproducibility-oriented tools.
- Test periodically, ideally in an automated way, that one can rebuild Seppo from scratch.

4.6 CLN-007 — No lock-out mechanism

Vulnerability ID: CLN-007

Vulnerability type: Rate limiting

Threat level: Low

Description:

There is no account lockout enforced on the application allowing password guessing attacks. Usernames are public.

Technical description:

Seppo lacks account lockout mechanisms that are useful in mitigating brute-force attacks. An attacker might implement brute force attacks or guessing attacks towards different accounts of users without any limitation or lockout mechanism. Usernames are public, making it easier for an attacker to target valid account names.

Impact:

The result of a successful attack is dangerous as the attacker will have full access to the user account, along with all the functionality and services they have access to.

Recommendation:

- Lock accounts after 3 to 5 unsuccessful login attempts.
- Only unlock accounts after a predetermined period of time, via a self-service unlock mechanism, secret questions, or intervention by an administrator.
- Use a CAPTCHA to prevent automated login attempts.

4.7 CLN-008 — Verbose error messages are displayed

Vulnerability ID: CLN-008

Vulnerability type: Information Disclosure

Threat level: Low

Description:

Internal error messages including detailed code errors and/or stack traces are displayed to the end user.

Technical description:

Web applications frequently generate error conditions during normal operation, such as null pointers, network timeouts, general failures and other common conditions that can lead to different types of errors. These must be handled according to a well-thought-out scheme that will provide a meaningful error message to the user, without revealing any internal or sensitive information about the application.
We found the following detailed error messages in the application:

Source URL: `test0.seppo.social/seppo.cgi/passwd`

```
Status: 500 Internal server Error see https://Seppo.Social/E1005 File "bin/cgi.ml", line 42,
 characters 2-8: Assertion Failed
```

Source URL: `test0.seppo.social/seppo.cgi/activitypub/actor.xml`

```
Status: 500 Internal Server Error see https://Seppo.Social/E1005 File "lib/ap.ml", line 1170,
 characters 4-10: Assertion failed
```

These errors were generated upon release of passwd file and retrieving resources, among other actions. These errors reveal internal information and should be treated to provide generic error messages to the end user.

Source URL: `/seppo.cgi/login`

```
Status: 500 Internal Server Error
see https://Seppo.Social/E1005
End_of_file*
```

This last error happens on sending a POST request without a form. It comes from the `form` function in `bin/cgi.ml` calling `Html.Form.of_channel`, itself using `input_line` from OCaml's standard library. `input_line` raises an `End_of_file` exception, caught only by the catchall handler in `bin/seppo_bin.ml`.

Impact:

While not a direct threat, the leaked internal information can help to target other attacks.

Recommendation:

- Display only generic error messages to end users.
- Log detailed error messages internally.

- Establish a policy for handling errors, and ensure that there is a global error handler to catch any errors not specifically intercepted.

4.8 CLN-010 — Incomplete input validation leads to crash

Vulnerability ID: CLN-010

Vulnerability type: Denial of Service

Threat level: Low

Description:

Certain injections or unexpected input causes the server to crash and freeze, without allowing any kind of interaction.

Technical description:

Injections cause a degradation of service leading to application freeze. Specifically, this occurs on the login page, on the user and password field when the following input has been inserted: `</script><script>alert(1)</script>`

Impact:

An attacker can send malformed requests, so that Seppo Web Server crashes, halts, or runs slowly; in all cases impacting availability.

Recommendation:

- Implement robust input validation (including size limits) and escaping of special characters in all user inputs.

4.9 CLN-013 — Improper handling of rejected Lwt promises

Vulnerability ID: CLN-013

Vulnerability type: Error handling

Threat level: Low

Description:

In various places, rejected `Lwt` promises can slip through a catch-all `try ... with`, potentially reaching the top-level.

Technical description:

Several places in Seppo feature a catch-all using the `try ... with` OCaml construct. See for instance L189-L194 of `lib/main.ml` as of `e37cc766737e6784672f08b7cb72a610c967dc3b`:

```
let%lwt r = try
   dispatch_job ~base ~pk j p
 with exn ->
   let e = exn |> Printexc.to_string in
   Logr.warn (fun m -> m "%s.%s Uncaught Exception job:%s %s" "Main.Queue" "process_new_and_due" j
 e);
   Error e |> Lwt.return
```

However, in some places, the value evaluated is an `Lwt` promise. Rejected `Lwt` promises will not be caught by such a construct, and will reach the top-level `Lwt_main.run` call, only then being transformed into exceptions.

Impact:

Exceptions might be propagated into unexpected places, potentially leading to unexpected control flow, and potentially leaking error messages as in CLN-008 (page 17). In the example above, if an exception occurs in `dispatch_job` for any reason, the cleanup phase of `Main.Queue.loop` does not happen and the job stays in `run` forever, not being retried, and not being cleaned up.

Recommendation:

Rework catch-all places to ensure that either they do not process an `Lwt` promise, or that the rejected `Lwt` promises are processed correctly, for instance by adding a `try%lwt ... with` construct or an explicit `Lwt.catch`. Note that it is not sufficient to ensure that `Lwt.fail` is never called, as regular exceptions can be turned into rejected `Lwt` promises by quite a few `Lwt` control flow handles.

4.10 CLN-014 — Improper handling of the PID in the lock file

Vulnerability ID: CLN-014

Vulnerability type: Concurrency

Threat level: Low

Description:

The PID file lock mechanism seems to work, but the content of the lock file is not guaranteed to be the PID of the process.

Technical description:

Consider this code in `Queue.loop` in `lib/main.ml` (see https://codeberg.org/seppo/seppo/src/commit/3b0578ff5f441a020603e8d74031f1a3cfaea0a6/lib/main.ml#L233-L238):

```
let fd = Unix.openfile lock [O_CLOEXEC; O_CREAT; O_TRUNC; O_WRONLY; O_SYNC] 0o644 in
let oc = fd |> Unix.out_channel_of_descr in
Printf.fprintf oc "%i" (Unix.getpid ());
flush oc;
Unix.lockf fd F_TLOCK 0;
```

The lock does not prevent another process from writing to the file, only from acquiring a write lock on the file (this is a cooperative mechanism). This means that another process will be able to open the file, truncate it, write its PID and flush, and only then raise an exception because it will not be able to acquire a lock.

Even if the lock prevented other processes from writing, the code contains a race where another process could have the time to write its PID entirely or partially to the file before the first process locks it.

Technically, there is also a race going on between processes where a process could acquire a file descriptor on the lock file before the other deletes it, resulting in a later exception when trying to write to the file descriptor. However, it is probably not a problem that one of the processes crashes when there is more than one.

Impact:

The lock mechanism actually works, however, the content of the lock file could be pretty much anything, from the right PID, the PID of another process, several PIDs mangled together, etc. If other pieces of software rely on this PID for other things, they could go wrong.

Recommendation:

Change the code to acquire the lock before any modification is made to the file. For instance:

```
let fd = Unix.openfile lock [O_CLOEXEC; O_CREAT; O_WRONLY; O_SYNC] 0o644 in
Unix.lockf fd F_TLOCK 0;
Unix.ftruncate fd 0;
let oc = Unix.out_channel_of_descr fd in
Printf.fprintf oc "%i" (Unix.getpid ());
flush oc
```

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-004 — Little documentation

The Seppo source code features very little documentation, rendering inspection and contribution more difficult. Some modules link to RFCs, but some of those RFC links are dead. This makes inspection and contributions more difficult, because it forces users to read all the code to understand what is happening. Additionally, because the functions have no specification, it is impossible to say whether peculiar behaviours are bugs or features.

Add at least:

- An overview document somewhere explaining what each directory in the repository is for, e.g. the difference between `bin` and `chkr`, `lib` and `as2_vocab`, etc.
- In each file/module and each submodule, a comment (can be two lines) explaining the purpose of this module and the kind of functions it contains.
- Ideally, a comment (and maybe a type) for some generic helpers, for instance in the Make module.
- Ideally, a comment on each function describing what is expected of it.

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 3 Moderate and 7 Low-severity issues during this penetration test.

We found several vulnerabilities in the Seppo web application, including issues with error handling, the potential for application crashes or stoppages, session cookie replay attacks, improper logout functionality, and a weak lockout mechanism. Fortunately none of these have high severity, but they weaken the app's security posture nonetheless.

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

Nicolas Jeannerod	Nicolas Jeannerod is a software consultant, specializing in functional programming languages such as OCaml, Haskell, and Nix/NixOS. Since joining Tweag in 2021, he has worked on various projects, from building dynamic analysis tools for smart contracts to enhancing blockchain consensus algorithms for Cardano. Currently, he is involved in security audits and leading the development of NixOS modules for the Fediversity project. Nicolas holds a PhD in Computer Science from Université de Paris, focusing on the verification of shell scripts for file hierarchy transformations.
Sabrina Deibe	Sabrina Deibe is an Information Systems Engineer with a Master's in Cryptography, Security, and Privacy. She has extensive experience in security and risk assessments, encryption, and key management using both open-source and proprietary tools. A dedicated educator, she teaches Network Security at the University of Buenos Aires. Her expertise includes network traffic analysis, intrusion detection/prevention systems, SIEM, SOAR, and programmable networks in cloud environments. Sabrina holds several certifications, including CCNP, CRISC, Google Cloud Certified Professional Architect, and AWS Certified Solutions Architect.
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.