RADICALLY OPEN SECURITY

NInet Security Evaluation Report

Seedvault

V 1.0
Amsterdam, January 16th, 2025
Confidential

Document Properties

Client	Seedvault
Title	NLnet Security Evaluation Report
Targets	• Limited code review of Seedvault • Assist the Seedvault project in reproducing and fixing the discovered issues • Discuss potential improvements for documentation and testing
Version	1.0
Pentesters	Christian Reitter, Neha Chriss
Authors	Christian Reitter, Neha Chriss, Marcus Bointon
Reviewed by	Marcus Bointon
Approved by	Melanie Rieback

Version control

Version	Date	Author	Description
0.1	January 10th, 2025	Christian Reitter, Neha Chriss	Initial draft
0.2	January 15th, 2025	Marcus Bointon	Review
1.0	January 16th, 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

Table of Contents

1 Executive Summary

1.1 Introduction

Between December 9, 2024 and January 7, 2025, Radically Open Security B.V. carried out a code audit for Seedvault and NLnet Zero Entrust as part of the Seedvault Integrity project grant.

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

1.2 Scope of Work

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

- Limited code review of Seedvault
- Assist the Seedvault project in reproducing and fixing the discovered issues
- Discuss potential improvements for documentation and testing

The scoped services are broken down as follows:

- Review: 4 days
- Reporting: 1.5 days
- Communication, shared sessions and other tasks: 1.5 days
- **Total effort: 7 days**

1.3 Project Objectives

ROS will perform a code review of the Seedvault Android system application with Seedvault in order to assess its security. To do so ROS will perform a code review on the newest public code revision of the open source project and guide Seedvault in attempting to find vulnerabilities, exploiting and analyzing any such found to try and gain further understanding of the security impact. Where possible, ROS will assist with retesting of fixes and mitigations that become available during the engagement phase.

Reviewed Seedvault version: commit `d0de677ddc8283f948c0bc84e3ee736cb60e1111` on the `android15` branch, which was the newest at the start of the review. The analyzed code revision is between the `15-5.1` and `15-5.2` version tags.

Please note that our review focuses exclusively on security aspects, and does not cover other issues such as functional problems with the backup system or user experience concerns.

1.4 Previous Review

ROS previously performed a very brief quick security evaluation of Seedvault for NLnet NGI Zero PET as part of the Seedvault project grant in 2021 for a total effort of two person days.

1.5 Backup Threat Model

The Seedvault team publicly documented their threat model and related concerns for the application backup sub-system before the beginning of this review. This documentation also provides some context for expected limitations and known attack concepts.

1.6 Timeline

The security audit took place between December 9, 2024 and January 7, 2025.

1.7 Results In A Nutshell

During this crystal-box penetration test we found 2 Moderate and 1 Low-severity issues.

The findings relate to dependencies with known vulnerabilities SVT-015 (page 12), potential code injection risks SVT-018 (page 13), and potential supply chain attacks against the CI/CD system SVT-024 (page 15).

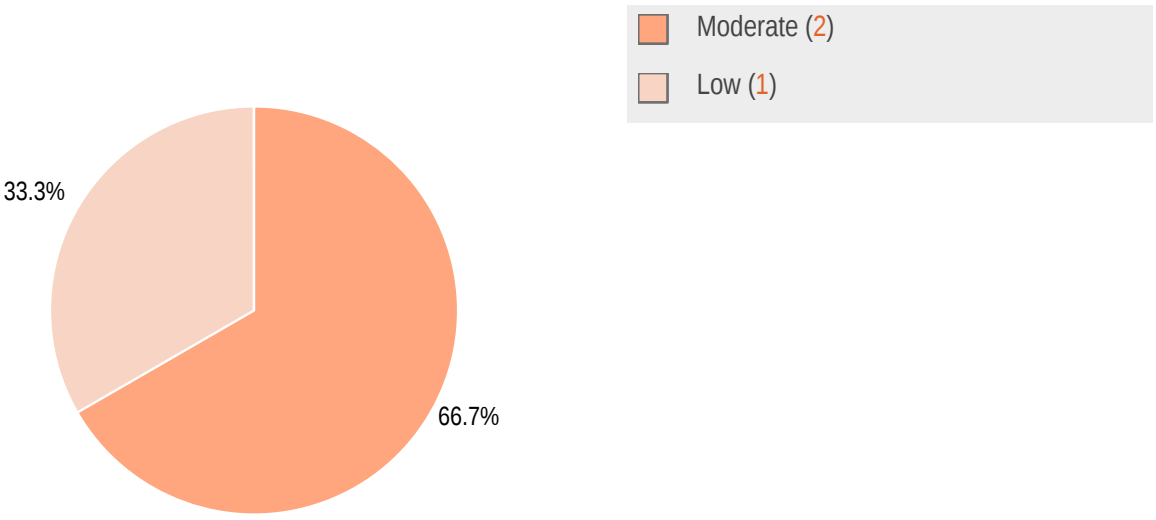
By exploiting these issues, an attacker might be able to cause a stack overflow in the application, inject code in scripts, or manipulate the CI/CD system. Notably, there are mitigating factors to most of these potential worst-case impacts, limiting their severity.

1.8 Summary of Findings

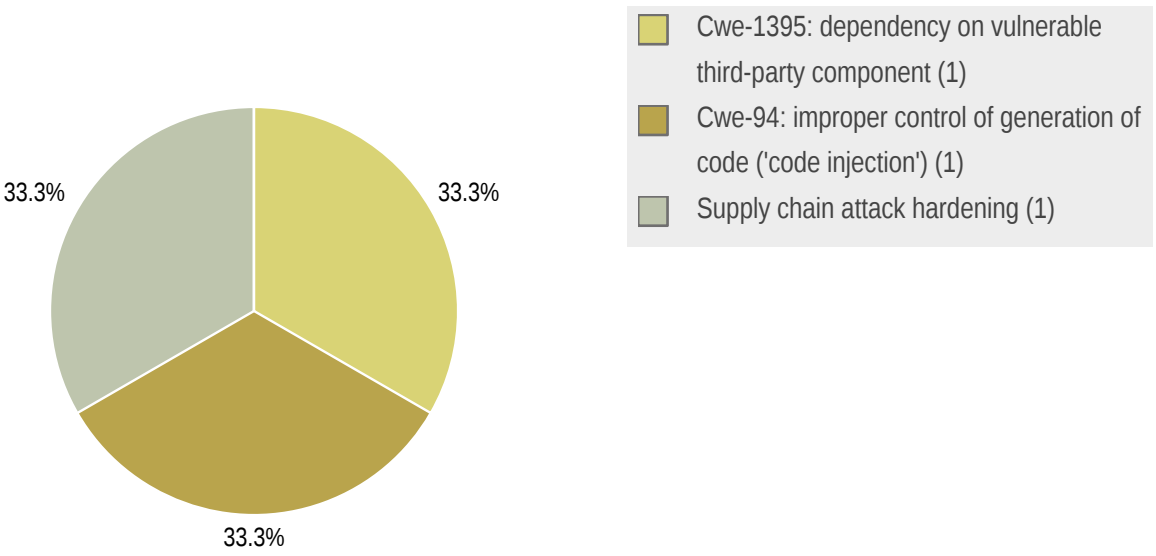
ID	Type	Description	Threat level
SVT-015	CWE-1395: Dependency on Vulnerable Third-Party Component	Seedvault depends on an older protobuf library version with known vulnerabilities.	Moderate
SVT-018	CWE-94: Improper Control of Generation of Code ('Code Injection')	Some bash scripts used for development and CI/CD use unquoted variable expansion, which could be a security risk under some conditions.	Moderate

SVT-024	Supply Chain Attack Hardening	Seedvault uses GitHub actions without strictly defined version dependencies.	Low

1.8.1 Findings by Threat Level



1.8.2 Findings by Type



1.9 Summary of Recommendations

ID	Type	Recommendation
SVT-015	CWE-1395: Dependency on Vulnerable Third-Party Component	• If possible, update to at least version `3.25.5` or a patched `4.x` version. • Further investigate the AOSP project's use of the outdated and known vulnerable version. • Investigate tools and mechanisms to detect similar known dependency vulnerabilities in the future.
SVT-018	CWE-94: Improper Control of Generation of Code ('Code Injection')	• Escape variable expressions to avoid the risky behavior. • Further investigate potential impact and relevance of the affected scripts. • Check existing shell scripts with common tools to spot similar issue patterns.
SVT-024	Supply Chain Attack Hardening	• Pin the versions of GitHub action dependencies and downloaded binaries. • We recognize that pinning leads to higher maintenance overhead, making this a project-specific trade-off.

2 Methodology

2.1 Planning

Our general approach during code audits is as follows:

During the code audit we verify if the proper security controls are present, work as intended and are implemented correctly. If vulnerabilities are found, we determine the threat level by assessing the likelihood of exploitation of this vulnerability and the impact on the Confidentiality, Integrity and Availability (CIA) of the system. We will describe how an attacker would exploit the vulnerability and suggest ways of fixing it.

This requires an extensive knowledge of the platform the application is running on, as well as the extensive knowledge of the language the application is written in and patterns that have been used. Therefore a code audit is done by specialists with a strong background in programming.

During this code audit, we take the following approach:

1. **Thorough comprehension of functionality**
   We try to get a thorough comprehension of how the application works and how it interacts with the user and other systems. Having detailed documentation at this stage is very helpful, as it aids the understanding of the application.
2. **Comprehensive code reading**
   Goals of the comprehensive code reading are:
   - to get an understanding of the whole code
   - identify adversary controlled inputs and trace their paths
   - identify issues
3. **Automated code scans**
   Using specialized tools, we scan the codebase for common issue patterns and then manually review the flagged code positions for potential negative security implications.

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.

- semgrep – https://semgrep.dev
- semgrep ruleset – https://github.com/semgrep/semgrep-rules
- semgrep ruleset – https://github.com/trailofbits/semgrep-rules
- TruffleHog – https://github.com/trufflesecurity/trufflehog
- Mobile Security Framework – https://github.com/MobSF/Mobile-Security-Framework-MobSF
- ShellCheck – https://github.com/koalaman/shellcheck

4 Findings

We have identified the following issues:

4.1 SVT-015 — Known vulnerability in protobuf dependency

Vulnerability ID: SVT-015

Vulnerability type: CWE-1395: Dependency on Vulnerable Third-Party Component

Threat level: Moderate

Description:

Seedvault depends on an older protobuf library version with known vulnerabilities.

Technical description:

In the newest `android15` branch, Seedvault has two direct dependencies on `protobuf 3.21.12`:

- The `libs/protobuf-kotlin-lite-3.21.12.jar` archive, link
- The `gradle/libs.versions.toml` dependency entry `protobuf = { strictly = "3.21.12" }`, link

According to the Maven repository, this version is from circa Dec 15, 2022. While it is the newest version in the `3.21.x` branch, there are newer releases in the `3.x` series. There is also a new major `4.x` version available.

Known vulnerabilities according to the Maven repository:

CVE-2024-7254, GHSA-735f-pc8j-v9w8:

> Any project that parses untrusted Protocol Buffers data containing an arbitrary number of nested groups / series of SGROUP tags can corrupted by exceeding the stack limit i.e. StackOverflow. [..]

CVE-2023-2976

> Use of Java's default temporary directory for file creation in `FileBackedOutputStream` in Google Guava versions 1.0 to 31.1 on Unix systems and Android Ice Cream Sandwich allows other users and apps on the machine with access to the default Java temporary directory to be able to access the files created by the class. [..]

CVE-2020-8908

> A temp directory creation vulnerability exists in all versions of Guava, allowing an attacker with access to the machine to potentially access data in a temporary directory created by the Guava API com.google.common.io.Files.createTempDir(). [..]

It is unclear if these are practically reachable within normal Seedvault usage, and what the exact security impacts are. Based on our limited analysis, the Seedvault application is unlikely to be directly affected by the temporary directory creation issues due to its use of safer alternatives. We're not clear whether the attack conditions for `CVE-2024-7254` are satisfied for Seedvault and scored the overall issue severity accordingly. According to the developers, Seedvault only processes externally provided protobuf data from encrypted and authenticated backups. Successful attacks would therefore require knowledge of the encryption key to prepare specially crafted backups.

Additionally, the Seedvault developers made us aware that the official Google AOSP Android project uses the vulnerable version `3.21.12` in its protobuf dependency definition for Android 15.0.0 r1. Newer versions like protobuf for Android 15.0.0 r8 still depend on it. As far as we can see, this limits the security patch options of the Seedvault developers through upgrades of the protobuf dependency at the moment.

Impact:

- The exact impact is unclear.
- Based on public vulnerability information, the impact could be a denial of service when handling malformed protobuf messages or use of insufficiently protected temporary directories and files.

Recommendation:

- If possible, update to at least version `3.25.5` or a patched `4.x` version.
- Further investigate the AOSP project's use of the outdated and known vulnerable version.
- Investigate tools and mechanisms to detect similar known dependency vulnerabilities in the future.

4.2 SVT-018 — Insecure variable expansion in bash scripts

Vulnerability ID: SVT-018

Vulnerability type: CWE-94: Improper Control of Generation of Code ('Code Injection')

Threat level: Moderate

Description:

Some bash scripts used for development and CI/CD use unquoted variable expansion, which could be a security risk under some conditions.

Technical description:

Several shell scripts in the Seedvault repository express bash variables in an unsafe way when constructing and running commands. Instead of writing `$variable`, the safer pattern `"$variable"` should be used to prevent problematic behavior, for example if the variable content contains spaces.

Shellcheck warning (excerpt):

```
In .github/scripts/build_aosp.sh line 93:
lunch $DEVICE-$RELEASE-$TARGET
      ^-----^ SC2086 (info): Double quote to prevent globbing and word splitting.
              ^------^ SC2086 (info): Double quote to prevent globbing and word splitting.
                       ^-----^ SC2086 (info): Double quote to prevent globbing and word splitting.

Did you mean:
lunch "$DEVICE"-"$RELEASE"-"$TARGET"
```

```
In app/development/scripts/start_emulator.sh line 22:
nohup $ANDROID_HOME/emulator/emulator -avd "$EMULATOR_NAME" -gpu swiftshader_indirect [...]
      ^-----------^ SC2086 (info): Double quote to prevent globbing and word splitting.

Did you mean:
nohup "$ANDROID_HOME"/emulator/emulator -avd "$EMULATOR_NAME" -gpu swiftshader_indirect -writable-
system >/dev/null 2>&1 &
```

Semgrep rule (excerpt):

```
.github/scripts/build_aosp.sh
     # bash.lang.correctness.unquoted-variable-expansion-in-command

          Variable expansions must be double-quoted so as to prevent being split into multiple
 pieces according to whitespace or whichever separator is specified by the IFS variable. If you
 really wish to split the variable's contents, you may use a variable that starts with an underscore
 e.g. $_X instead of $X, and semgrep will ignore it. If what you need is an array, consider using a
 proper bash array.



           93# lunch $DEVICE-$RELEASE-$TARGET
```

The practical security risks of this pattern depend on an attacker's abilities to inject bad content into the variables in question, for example via control of environment variables. However, in case of accidental use of problematic variable content by the developers, the current construction can also lead to bugs.

Due to time limitations, we were not able to investigate this issue in depth. The chosen severity rating reflects a mix of relevant factors of potential risk but low likelihood of practical attacks.

According to the Seedvault developers, `.github/scripts/build_aosp.sh` is currently unused.

Impact:

- In a worst-case scenario, a less-privileged user can inject code into a script which is executed in a higher-privileged environment.
- According to the Seedvault developers, at least one affected CI/CD script is currently unused.

Recommendation:

- Escape variable expressions to avoid the risky behavior.
- Further investigate potential impact and relevance of the affected scripts.
- Check existing shell scripts with common tools to spot similar issue patterns.

4.3 SVT-024 — GitHub action templates do not pin exact versions

Vulnerability ID: SVT-024

Vulnerability type: Supply Chain Attack Hardening

Threat level: Low

Description:

Seedvault uses GitHub actions without strictly defined version dependencies.

Technical description:

The `semgrep` analysis tool warns of the use of GitHub actions without exact version pinning:

```
.github/workflows/build.yml
    ## yaml.github-actions.security.third-party-action-not-pinned-to-commit-sha

          An action sourced from a third-party repository on GitHub is not pinned to a full length
 commit SHA. Pinning an action to a full length commit SHA is currently the only way to use an
 action as an immutable release. Pinning to a particular SHA helps mitigate the risk of a bad actor
 adding a backdoor to the action's repository, as they would need to generate a SHA-1 collision for
 a valid Git object payload.

           38# uses: gradle/actions/wrapper-validation@v3
            ##----------------------------------------
           57# uses: mikepenz/action-junit-report@v4
```

```
   .github/workflows/test.yml
   ## yaml.github-actions.security.third-party-action-not-pinned-to-commit-sha

         An action sourced from a third-party repository on GitHub is not pinned to a full length
commit SHA. Pinning an action to a full length commit SHA is currently the only way to use an
action as an immutable release. Pinning to a particular SHA helps mitigate the risk of a bad actor
adding a backdoor to the action's repository, as they would need to generate a SHA-1 collision for
a valid Git object payload.

          40# uses: Wandalen/wretry.action@v1.3.0
```

Pinning exact versions can also help to guarantee higher reproducibility of builds.

Additionally, consider strong pinning of any software downloaded within GitHub CI tasks, such as `https://storage.googleapis.com/git-repo-downloads/repo` used by `build_aosp.sh`.

At the moment, the Seedvault project does not use the GitHub actions to build official software releases or sign them cryptographically, so we think the potential worst-case impact through a supply chain attack of the CI code is limited to malicious influences of the specific CI run.

Impact:

- By not pinning the GitHub actions to a specific software commit, a supply chain attack through a GitHub action has a higher chance of succeeding.
- We see the overall worst-case impact on the project as limited.

Recommendation:

- Pin the versions of GitHub action dependencies and downloaded binaries.
- We recognize that pinning leads to higher maintenance overhead, making this a project-specific trade-off.

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-001 — Investigate used BIP39 libraries

Seedvault uses the BIP39 standard to encode a complex secret key into a human-readable and standardized representation for easier backup and recovery handling. Weaknesses in the cryptographic code which generates the BIP39 secret would severely undermine the security of the encrypted backups and recovery procedures.

We briefly investigated the kotlin-bip39 in version `v1.0.8` used by Seedvault. We are not aware of any publicly known issues in this software, making this a non-finding.

5.2 NF-008 — Add a SECURITY.md with public security information

We recommend adding a `SECURITY.md` file providing information for potential security vulnerability submitters on where and how they can report a vulnerability.

Additionally, it could be helpful to outline the Seedvault project's current maintenance goals for its versions and branches. According to discussion with the developers on 2024-12-09, once Seedvault development switches to a new Android version target and branch (e.g. `android14` -> `android15`), the previously used branch(es) effectively become unmaintained. The current project documentation doesn't make this clear, so integrating projects and end users may have an incorrect understanding of the project maintenance level when using older versions. This has relevance for security if security backports are missing for older branches.

The Seedvault developers fixed this during the review via https://github.com/seedvault-app/seedvault/pull/810.

5.3 NF-011 — Investigate HTTP/TLS handling and WebDav implementation

Due to speed and reliability problems with the standard Android APIs for talking to remote WebDav servers (according to the developers), Seedvault talks to WebDav servers (like Nextcloud) natively for file storage.

We analyzed the security configuration of the `dav4jvm` and `okhttpclient` libraries, which Seedvault uses for WebDav and TLS connections. We specifically evaluated the service's susceptibility to downgrade attacks or insecure transport of backup data.

WebDav library details:

- `dav4jvm`
- seems to be used in a git commit version after the last official stable release `2.2.1` (Jan 2023)

- version description fits to the https://github.com/bitfireAT/dav4jvm/commit/c1bc143 git commit hash

When reviewing Seedvault's `OkHttpClient.Builder()` configuration, we noted it includes the `okhttpclient` lib's default `MODERN_TLS` configuration.

```
private val okHttpClient = OkHttpClient.Builder()
        .followRedirects(false)
        .authenticator(authHandler)
        .addNetworkInterceptor(authHandler)
        .connectTimeout(30, TimeUnit.SECONDS)
        .writeTimeout(60, TimeUnit.SECONDS)
        .readTimeout(240, TimeUnit.SECONDS)
        .pingInterval(45, TimeUnit.SECONDS)
        .connectionSpecs(listOf(ConnectionSpec.MODERN_TLS))
        .retryOnConnectionFailure(true)
        .build()
```

The `MODERN_TLS` spec include cipher suites that do not offer ephemeral key exchange, are based on TLS stream, block or null cipher types. While a small portion of these cipher suites are considered weak, the more secure alternative, `RESTRICTED_TLS`, would limit compatibility with a wide range of Nextcloud endpoints.

For the purposes of this review, we believe the current configuration offers the minimum viable secure configuration for Seedvault users.

5.4 NF-014 — Outdated DEP-5 copyright file

The Seedvault project has a machine-readable copyright file which follows the Debian copyright format, also known as `DEP-5`.

During review, we noticed that many entries are outdated for the `android15` branch and appear to have information belonging to previous Android releases.

Example:

```
Files: libs/dav4jvm/okhttp-4.11.0.jar
       libs/dav4jvm/okio-jvm-3.7.0.jar
Copyright: 2024 Square, Inc.
License: Apache-2.0
```

Seedvault has moved the library to `core/libs/dav4jvm/okhttp-4.12.0.jar` and updated the library version.

We recommend updating the copyright information or considering deprecating the support of this format.

5.5 NF-016 — Check git repository for accidentally uploaded credentials

No secrets were uncovered after scanning with the Trufflehog tool.

Trufflehog output:

```
$ trufflehog git https://github.com/seedvault-app/seedvault --results=verified,unknown


2024-12-18T22:36:39+09:30       info-0  trufflehog      running source  {"source_manager_worker_id":
 "pZ7Z7", "with_units": true}
2024-12-18T22:36:39+09:30       info-0  trufflehog      scanning repo   {"source_manager_worker_id":
 "pZ7Z7", "unit_kind": "dir", "unit": "/var/folders/2y/y2m9hchj3dj9f556glmvm0wh0000gn/T/
trufflehog-25646-3062318450", "repo": "https://github.com/seedvault-app/seedvault"}
2024-12-18T22:36:51+09:30       info-0  trufflehog      finished scanning
 {"chunks": 38291, "bytes": 84527197, "verified_secrets": 0, "unverified_secrets": 0,
 "scan_duration": "28.857634917s", "trufflehog_version": "3.88.0", "verification_caching":
 {"Hits":2,"Misses":1,"HitsWasted":0,"AttemptsSaved":2,"VerificationTimeSpentMS":789}}
```

5.6 NF-020 — Investigated potentially risky use of temporary files

The `MobileSecurityFramework` analysis tool warns of a potentially dangerous use of temporary files:

```
warning
App creates temp file. Sensitive information should never be written into a temp file.
CWE: CWE-276: Incorrect Default Permissions
OWASP Top 10: M2: Insecure Data Storage
OWASP MASVS: MSTG-STORAGE-2     com/stevesoltys/seedvault/restore/install/ApkRestore.java
```

The relevant function is `cacheApk()`. Note that there are multiple different implementations of the `createTempFile()` function in different frameworks. The code uses java.io.File.

For reference, here is the similarly-named Kotlin file mechanism which is deprecated:

> Deprecated
>
> Avoid creating temporary files in the default temp location with this function due to too wide permissions on the newly created file. Use kotlin.io.path.createTempFile instead or resort to java.io.File.createTempFile.

The general concern is that, for example through lax read or write permissions of the newly created file, a malicious local application with sufficient access to the temporary directory could obtain information or impact the integrity of the file.

The Seedvault developers noted that in their use of the function, the directory of the temporary files is an application-specific cache directory which is isolated from other applications by default, as per the Android documentation.

We therefore see this as a non-finding.

5.7 NF-021 — Investigated potentially risky SQL queries

The Seedvault code contains direct SQL calls, which we investigated for potential SQL injection.

We think they're not vulnerable since both SQL statements are hard-coded and only use constants, instead of attacker-controlled content, which avoids any opportunity for malicious SQL injections.

```
app/src/main/java/com/stevesoltys/seedvault/transport/backup/KVDbManager.kt
  109,12:         db.execSQL(SQL_CREATE_ENTRIES)
  118,46:     override fun vacuum() = writableDatabase.execSQL("VACUUM")
```

We recommend adding some code documentation to outline the current usage and highlight potential risks of code refactoring.

5.8 NF-022 — Logic bug in CI build script build_aosp.sh

The `shellcheck` analysis tool warns of a problem in the build_aosp.sh build script:

```
In .github/scripts/build_aosp.sh line 32:
  while [[ $attempt < $max_attempts ]]
                    ^-- SC2071 (error): < is for string comparisons. Use -lt instead.
```

Due to the literal comparison, this function will behave incorrectly for some variable assignments. For example, it will perform fewer than expected attempts for `$max_attempts=10`.

Additionally, the script appears to download the same `repo` binary twice, via different methods.

While this may cause unintended behavior for some variable configurations, we do not see it as a security vulnerability. Additionally, the `build_aosp.sh` script is currently unused, according to the developers.

5.9 NF-023 — Investigated security of CI/CD credentials

Seedvault uses the GitHub Actions Continuous Integration (CI) system to run special actions and tests on git commits to the repository.

The CI tasks use a secret to log into an external server which could be revealed through malicious project commits.

The build job passes login credentials of a Nextcloud account as environment variables into the virtual machine which builds the new software version. If attackers can run malicious code in this environment while the secrets are present, they can leak and use the long-lived login credentials.

The purpose of the Nextcloud usage is to perform end-to-end tests of the WebDav network storage functionality with emulated Seedvault debug builds.

Although the build job task runs on trusted commits as well as untrusted pull requests (see here), we confirmed experimentally that GitHub's default protections prevent attacks through malicious pull requests. CI actions triggered by pull requests do not get access to repository secrets, and instead only run with empty string placeholders, making the attack ineffective.

To perform a successful attack, an attacker would therefore have to trick the project developers into accepting malicious code of some sort into an official git branch of the project, either through malicious code changes or a compromised code dependency.

The Seedvault developers outlined that the Nextcloud account in question is only used for this particular purpose and has limited access, which we see as a mitigating factor to limit the impact of any successful attacks. Note that the Nextcloud instance itself was out-of-scope for this security review and not analyzed.

Overall, given that some form of external network storage access is necessary to test the relevant software functionality, we see the current usage as a reasonable trade-off, but still wanted to highlight the associated risk and efforts performed to evaluate it.

The following proof-of-concept attack modifies the `gradlew` shell script to leak a secret. When called by the `build` target with secrets, it will print a sensitive environment variable to the log output, circumventing a basic GitHub leak protection.

```
diff --git a/gradlew b/gradlew
index f5feea6d..da05a0b1 100755
--- a/gradlew
+++ b/gradlew
@@ -64,6 +64,30 @@
 #
 ##############################################################################

+echo "Proof-of-Concept secret extraction"
+poc="${NEXTCLOUD_PASS}"
+# naive way to print the secret to the log output
+# the GitHub CI system will detect and censor this secret
+echo "$poc"
+poc_len="${#poc}"
+# print the length of the secret, for debugging
+echo "length: $poc_len"
+# print the secret one character at the time, separated by spaces
+# this will show up in the logs since it does not get censored
+# use external command instead of bash string handling to be compatible
+# with shells like sh
+i=0
+while [ "$i" -lt $poc_len ]; do
+  echo -n "${poc}" | cut -c$(( i + 1)),$(( i + 1)) -z
+  echo -n " "
+  i=$(( i + 1 ))
+done
+
+echo ""
+echo "Early exit."
+# intentionally abort the build with errors
+exit 1;
+
```

5.10 NF-026 — Investigate the use of AES-GCM-HKDF in Seedvault

After review, Seedvault appears to use secure defaults for encryption. AES-GCM-HKDF stream encryption is configured with the following settings:

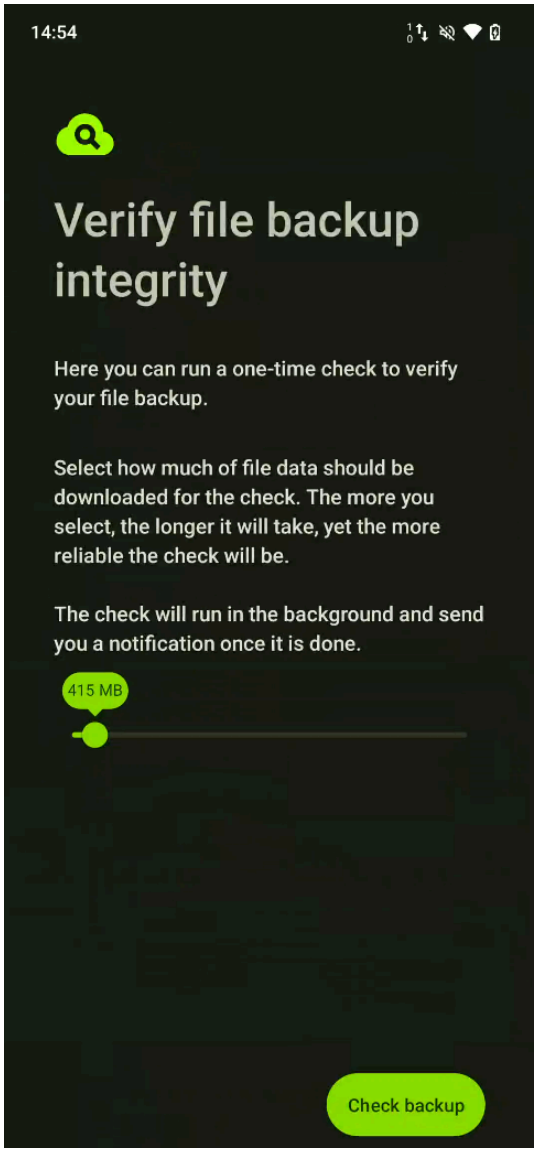
- AES-GCM with 256-bit key size
- 1MB segment size for streaming encryption
- HMAC-SHA256 for key derivation
- Uses Google's Tink library's `AesGcmHkdfStreaming` implementation

HKDF is using secure defaults as well:

- Uses HMAC-SHA256 as the hash function
- Supports context-specific info parameter for key derivation
- Implements the HKDF-Expand function as specified in RFC 5869

5.11 NF-028 — Discussed new partial file backup check feature

During the review period, the Seedvault developers added new functionality to allow end users to trigger "Verify file backup integrity" operations in the application. Since the code was introduced after the revision targeted by our review, we did not analyze the code changes, but discussed the high-level properties of the new functionality with the developers.



This verification mechanism is mainly designed to check the presence and integrity of file-related backups on an external network storage. Notably, end users can optionally specify that only a randomly sampled portion of the backup should be downloaded to the smartphone and checked, which implements a trade-off between full backup checks and no backup checks.

We regard this as useful functionality to help end users to check whether their backup is still viable, and that it has not been deleted through some accidental action (e.g., unreliable storage server) or malicious attack (e.g., ransomware overwriting or deleting the backup files). The partial backup check mechanism is clearly limited in terms of guarantees about the whole backup, but we see it as an interesting middle ground which may make this functionality more accessible and usable to end users with slow network connections, strongly limited data plans on metered networks, or other performance concerns such as limited battery power.

As far as we're aware, the backup verification currently has to be triggered manually. We recommend investigating future options for an optional mode to regularly schedule these checks, for example once a month, once the presence of an unmetered network connection has been detected.

5.12 NF-029 — Software dependencies with available updates

During our review, we found the use of outdated software components, with updates that have been available for more than 6 months, but have not yet been applied:

- `kotlin-logging-jvm-6.0.3.jar`, newer `6.0.x` and `7.x` versions available, see MVN repository

This list is not exhaustive. We recommend evaluating and switching to one of the newer versions.

Additionally, we spotted a dependency that is possibly unmaintained since it has not seen a software update in over 1.5 years:

- `logback android`, MVN repository, GitHub repository

We recommend investigating whether this dependency still satisfies the Seedvault project's requirements, for example its impact on security maintenance.

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 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 information security.

7 Conclusion

We discovered 2 Moderate and 1 Low-severity issues during this penetration test.

Overall, our impression of the Seedvault project security is very positive. Within the limited time available for this NLnet security evaluation project, we did not find any significant or easily exploitable flaws in the Seedvault application. While we did not have the resources to perform in-depth cryptographic reviews of the different components, the basic primitives and configurations appear to be reasonable, and we did not spot any substantial problems of the Android application configuration.

In terms of application security, the most relevant finding is the use of a known-vulnerable version of the protobuf library, which may introduce several different security problems via three known vulnerabilities. Due to complex circumstances and mitigating factors, there is some uncertainty in regard to its practical impact. We suspect this could only be used by a privileged attacker to perform denial-of-service attacks, as a worst case scenario. Additionally, it appears that the official Android AOSP system also uses the affected library version, which likely contributed to the introduction of this issue.

The remaining findings address security details of the Seedvault CI/CD system on GitHub, and development scripts. Although the CI/CD system does not perform any highly sensitive tasks such as signing release binaries, we still think it could be hardened against supply chain attacks by pinning the exact versions of GitHub action templates, and applying stronger checks to downloaded binaries. Multiple shell scripts for CI/CD and development used an insecure variable expansion expression, which under some conditions may lead to unexpected behavior like code injection. We see the associated risks as limited, but recommend applying stricter coding standards, for example via static code analysis.

Overall, we have the impression that the Seedvault developers take care to avoid common vulnerability patterns, focus on secure default configurations, and regularly invest time in keeping their dependencies up-to-date, which are good to see from a review perspective.

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.

We would like to thank the Seedvault developers Torsten Grote and Chirayu Desai for their assistance and expert feedback during this review.

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 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

Christian Reitter *(pentester)*	Christian is an IT Security Consultant with experience in the area of software security and security relevant embedded devices. After his M.Sc. in Computer Science, he has worked as a developer and freelance security consultant with a specialization in fuzz testing. Notable published research includes several major vulnerabilities in popular cryptocurrency software and hardware wallets. This includes remote recovery of wallet keys with weak entropy, remote code execution on hardware wallets, remote extraction of secret keys from hardware wallets and circumvention of 2FA protection. He also discovered multiple memory issues in well-known smartcard driver stacks.
Neha Chriss *(pentester)*	Neha is a principal application security engineer and a classic builder/breaker, with over two decades in infrastructure and product security. Her recent work focuses on testing payments and banking application stacks, though her interests lay in various systems and protocols, from SmartTV embedded operating systems and DRM to the GSN (Global Seismographic Network), and national energy management systems.
Melanie Rieback *(approver)*	Melanie Rieback is a former Asst. Prof. of Computer Science from the VU, who is also the co-founder/CEO of Radically Open Security.