secuvera
Cybersicherheit. Nachhaltig.

PENETRATION TEST REPORT

VERSION 1.7 – Redacted Report | 10th September, 2026

Closing Report for heylogin GmbH
Sophienstr. 40
38118 Braunschweig

secuvera GmbH
Siedlerstraße 22-24
71126 Gäufelden

Authors
Timo Schäpe, Dmytro Volosnik

TABLE OF CONTENTS

1. SUMMARY

1.1. Static Source Code Analysis

From 10th to 19th September, 2025 the source code of the software heylogin was analyzed on behalf of the heylogin GmbH (referred to as "customer" in the following) in a security audit in the form of a static source code analysis. The goal of the audit was to identify vulnerabilities in the source code.

Overall, a medium level can be attested to the source code, as one or more vulnerabilities with medium severity werde identified.

Table 1: Statistical Summary of Identified Vulnerabilities and Informational Findings during Static Source Code Analysis

Total Number of		Number of Vulnerabilities (Severity)				Overall Security Level
Vulner-abilities	Informational Findings	Critical	High	Medium	Low	
3	2	0	0	3	0	Medium

During the penetration test, three vulnerabilities could be identified. These are listed below in descending order according to the severity of the vulnerability. The resulting security level of the web application is explained in Appendix B (see section 6.).

Medium **C_01 Hardcoded Credentials (Remediated)**

Hard-coded secrets were found. When assessing the severity (5.1), it was taken into account that only a limited group of people have access to the source code. However, secrets should never be included in the source code. Instead, sensitive application data can be made available in environment variables.

Medium **C_02 CORS Misconfiguration (Remediated)**

This medium-severity vulnerability results from a CORS misconfiguration. This potentially allows external domains to query the application's content or, if necessary, make changes to it.

Medium **C_03 Weak iOS Keychain Accessibility Configuration (Remediated)**

The application configures iOS Keychain storage with a weak accessibility level (`kSecAttrAccessibleAfterFirstUnlockThisDeviceOnly`) that allows sensitive authentication data to remain accessible in device memory after the initial unlock. This configuration enables attackers with physical device access or on jailbroken devices to extract sensitive using readily available security testing tools.

In addition, two informational findings were identified. No direct risks ensue from these findings, which is why they are not considered to be vulnerabilities. However, the general security level of the source code can be increased by addressing these findings. Information of these identified informational finding are described in chapter 3.2.3.

A detailed description of the used testing methodology can be found in section 3.1.

1.2. Retest 4th, September, 2026

From 3rd to 4th September, 2026, a penetration retest was conducted. This follow-up test relates to version 1.0 of the results report, dated 17th November, 2025. All vulnerabilities identified during the original test were retested.

Overall, a very high level can be attested to the source code, as no remaining vulnerabilities were identified.

Table 2: Statistical Summary of Identified Vulnerabilities and Informational Findings during Retest

	Total Number of		Number of Vulnerabilities (Severity)				Overall Security Level
	Vulner-abilities	Informa-tional Findings	Critical	High	Medium	Low	
Penetration Test 17th Nov, 2025	3	2	0	0	3	0	Medium
Retest 4th Sep, 2026	0	2	0	0	0	0	Very high

Table 3: Comparison of Vulnerabilities to Previous Test

Penetration Test 17th November, 2025		Retest 4th September, 2026	
Vulnerability	**Severity**	**Vulnerability**	**Status**
C_01 Hardcoded Credentials	Medium	C_01 Hardcoded Credentials	Remediated
C_02 CORS Misconfiguration	Medium	C_02 CORS Misconfiguration	Remediated
C_03 Weak iOS Keychain Accessibility Configuration	Medium	C_03 Weak iOS Keychain Accessibility Configuration	Remediated

A detailed description of the testing methodology used can be found in section 3.1.2.

2. GENERAL INFORMATION

This chapter provides general information on the structure of the report. The results of the retest are presented in a separate chapter.

All results are valid for the described tests, software versions, and configuration only. New vulnerabilities or vulnerabilities introduced due to changes in systems or applications cannot be identified in advance. Therefore conclusions to future robustness cannot be derived from the results presented in this document. In case of major changes, retesting is recommended.

Tests were conducted with the effort defined by the project scope. This approach assures the best test coverage possible. Naturally, with this testing methodology and the limited time, complete test coverage is impossible.

2.1. Common Vulnerability Scoring System

The Common Vulnerability Scoring System (CVSS) v4.0 is being used to evaluate severities of vulnerabilities identified.[1] CVSS is the leading industry standard for risk evaluation of vulnerabilities and was developed by the Forum of Incident Response and Security Teams (FIRST).

In the field of IT security, the "defense in depth" approach has become the standard. In all layers of IT, all commercially viable security measures should therefore be taken. This leads to sustainable resilience against attacks. For example, we are advising to use the recommendations for cryptographic techniques of the German BSI and show differences to these recommendations, even though no exploitable vulnerabilities might result from those differences. The CVSS-Score of such findings is normally "none".

The base score indicates the severity of this vulnerability and relates to the test cases performed and the behavior of the test objects. Consideration of the full context within the customer environment is only possible to a limited extent in the context of penetration tests. The rating therefore does not represent a final risk assessment in accordance with current standards (e.g. ISO 27005). The basic score is calculated by the tester for all independently identified vulnerabilities.

Vulnerabilities often also result from the use of (outdated) software with publicly known vulnerabilities. Usually it is only known that the version is vulnerable, but very often no details about the exact nature of the vulnerability, its exploitability and its exact impact are publicly known.

In these cases, the auditor cannot carry out a manual assessment, as the information situation does not allow this and it would not be productive in these cases to carry out the necessary vulnerability research as part of the customer project. Instead, we reference information from trustworthy vulnerability databases or directly from the manufacturer of the affected software.

Please note that when referencing in this way, it may be necessary to use ratings that were calculated with an older CVSS version. Especially in the case of long-known vulnerabilities or particularly old versions, the CVSS ratings in the vulnerability disclosures of the manufacturers or the vulnerability databases are often not updated to the latest CVSS version.

For risk assessment and handling by the customer, the corresponding vector string is specified for each vulnerability assessment. The customer can use the CVSS Environmental Metrics, for example, to adapt to the respective application context.

[1] https://www.first.org/cvss/

2.2. Presentation

The goal of the security audit as well as the steps taken to reach it are documented in chronological order in separate chapters. This ensures the traceability of the drawn conclusions with regards to the results.

All reports generated by tools used during the tests were verified and evaluated manually and attached to this document. Findings listed in these but not within this document are either false positives or have no relevance.

Further information on the identified vulnerabilities can be found in the tool-generated reports. To offer a convenient overview of the vulnerabilities, no detailed information is listed in this document.

Each vulnerability and informational finding is initially described separately along with its possible impact. Subsequently, an individual severity evaluation is given, as well as a recommended course of action in order to remediate the vulnerability or informational finding. The listings also contains results from manually conducted tests.

In order to easily reference individual vulnerabilities and informational findings, each of the findings is given a unique identifier and labelled with a continuous index. The nomenclature is described in the following:

- C_ vulnerabilities found during source code analysis.

Informational findings are not classified as vulnerabilities, because they do not pose a direct risk or may only be a deviation from established security standards or best practices. The CVSS score of informational findings is usually “none”, and they have no influence in a security level evaluation. In this report, they are listed using the following nomenclature:

- H_ informational findings.

Since informational findings do not pose a direct risk, they are not taken into account when determining safety levels.

3. STATIC SOURCE CODE ANALYSIS

3.1. Testing Methodology and Goal

3.1.1. Static Source Code Analysis

From 10th to 19th, September, 2025 the source code of the Software heylogin was analyzed on behalf of the customer in a security audit in the form of a static source code analysis. The goal of the audit was to identify vulnerabilities in the source code.

The source code was first checked using the automated static source code analysis scanning tools "Coverity"[2] and "Semgrep Code"[3], and the results were verified manually. Additionally, manual tests were performed to identify weaknesses in the source code.

The test was performed as a white-box test, meaning that the testers had access to the source code.

The source code was made available via the Git repository https://url.to.git.repo/heylogin.git in the branch main. The source code was downloaded on 10th September 2025 (last commit hash: 4d04d24343038084219b543dddb567614a3c60e8).

The repository is a monorepo containing all of the customer's applications. Of these applications, the following were in the focus of the review:

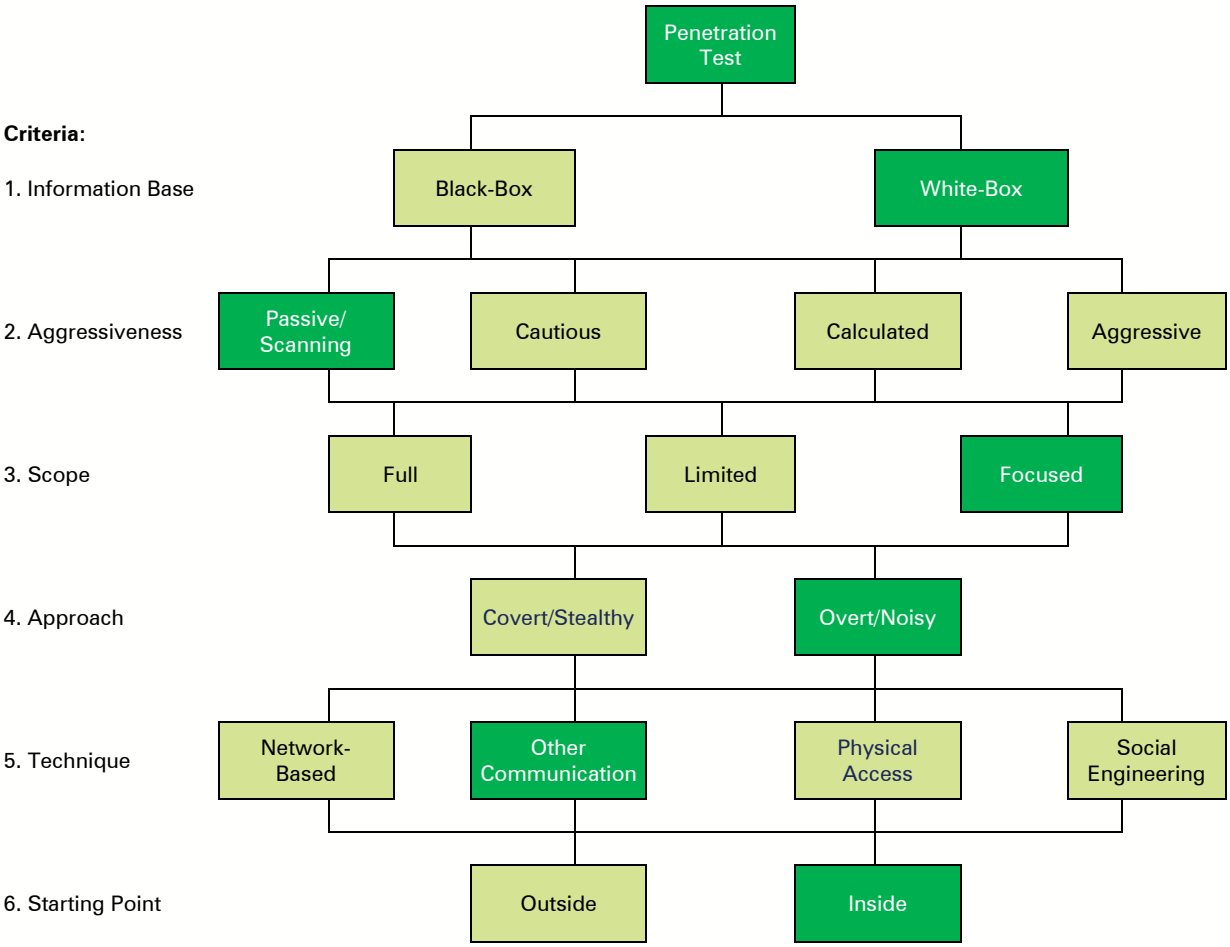
- Priority 1: component-01
- Priority 1: component-02
- Priority 1: component-03
- Priority 2: component-04
- Priority 2: component-05
- Priority 2: component-06
- Priority 2: component-07
- Priority 3: component-08
- Priority 3: component-09
- Priority 3: component-10

The tests were carried out according to the penetration testing model of the German Federal Office for Information Security (BSI).

[2] https://www.synopsys.com/software-integrity/static-analysis-tools-sast/coverity.html

[3] https://semgrep.dev/products/semgrep-code/

Figure 1: Methodology according to the BSI Study "A Penetration Testing Model[4]"



3.1.2. Retest 4th September, 2026

From 3rd to 4th September, 2026, the source code of the Software heylogin was analysed on behalf of the customer as part of a security audit conducted in the form of a static source code analysis. The goal of the audit was to identify vulnerabilities in the source code.

The vulnerabilities identified during the previous audit were reviewed manually. For each vulnerability, the affected code locations were examined in the current version of the source code to verify whether the customer had remediated the reported vulnerabilities.

Informational findings from the previous audit were not reassessed as part of the retest.

The test was performed as a white-box test, meaning that the testers had access to the source code.

The source code was made available via the customer's Nextcloud instance and was downloaded on 26th August, 2026 (heylogin-80c17219.tar.gz).

The repository is a monorepo containing all of the customer's applications. For the retest, the review was limited to the source code locations affected by the vulnerabilities identified during the previous audit. Other parts of the repository were not systematically reassessed.

---

4 https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/Penetrationstest/penetrationstest.html

The tests were carried out according to the penetration testing model of the German Federal Office for Information Security (BSI), see Figure 1.

3.2. Course of the Project and Results

All tests were performed as planned, following the methodology described in the previous chapter. The tests were carried out using a risk-orientated, time-based approach. For this reason, especially findings that were assessed by Coverity and Semgrep Code as having a medium or high impact were analyzed in greater depth.

A vulnerability related to insecure communication was found. However, after consultation with the client, it was removed, as the client explained that encrypted communication is ensured through other means.

The vulnerability "Missing TLS Certificate Pinning Implementation" was downgraded to an informational finding. The OWASP Cheat Sheet is advising against the implementation of certificate pinning: "There is almost no situation where you should consider pinning. The risk of outages almost always outweighs any security risks given advances in security."[5]

The vulnerability "HTTP Response Injection through Unvalidated User Input" was removed from the report. After consultation with the client, it is an implementation that complies with Microsoft's specification and accordingly cannot be implemented differently.

3.2.1. Vulnerability Listing

In order to increase readability and to avoid redundancies, every type of identified vulnerability is described in this chapter.

Note on vulnerability assessment: In order to be able to evaluate the vulnerabilities that were identified during the source code analysis in a uniform manner, they are evaluated using the CVSS metric. However, the metric is not fully mappable to vulnerabilities identified during source code analysis. For example, in some cases it is not possible to make an accurate statement about the values for "Attack Complexity" or the "Privileges Required", because in SAST, the focus is on weaknesses in the development and not on the exploitation of the resulting vulnerability. In these cases, a "worst case" scenario is therefore assumed, and the vulnerability is assessed accordingly.

C_01	**Hardcoded Credentials (Remediated)**
Retest Status	The hardcoded credentials identified during the initial assessment were no longer present in the reviewed source code.
Description	During the test, hardcoded credentials were found in the source code.
Impact	Hardcoded credentials may compromise system security in a way that cannot be easily remedied. Not only does hardcoding a credential allow all of the project's developers to view the password, it also makes fixing the problem extremely difficult. Once the code is in production, the password cannot be changed without patching the software.
Additional info	Project members have access to the source code and therefore access to the login information. This also applies to previous project members who had access to the code. A leak of the code or incorrect authorizations for a repository could also make the information publicly accessible.

[5] https://cheatsheetseries.owasp.org/cheatsheets/Pinning_Cheat_Sheet.html#when-do-you-perform-pinning

Example The following hardcoded credentials were found:

Table 4: Hardcoded Credentials

File	Line of Code	Type of Secret
path/to/vars/gitlab_ci_variables.yml	310	API Token
path/to/.env.backend	5, 6, 18, 21, 23, 24, 25	API Token
path/to/fcm-credentials.json	5	Private Key
path/to/group_vars/class_sendy.yml	3	Password
path/to/files/config.json	7, 14	API Key
path/to/loki.json	4	Password

OWASP Top 10 2021 A05 – Security Misconfiguration

Recommendation Credentials should never be hardcoded into the source. Instead try to transfer sensitive data into environment variables.

References https://cheatsheetseries.owasp.org/cheatsheets/Secrets_Management_Cheat_Sheet.html

Severity Medium

Table 5: Severity Evaluation Vulnerability C_01

Severity evaluation according to CVSS v4.0 (Base Score, CVSS-B)		
Metric	**Rating**	**Explanatory Statement**
Attack Vector (AV)	Network	The vulnerability can be exploited via a network.
Attack Complexity (AC)	Low	The attacker does not need to take any quantifiable measures to actively circumvent or overcome existing security-enhancing precautions in order to exploit the vulnerability.
Attack Require-ments (AT)	None	No special requirements for the configuration of the application need to be met in order to exploit the vulnerability.

Severity evaluation according to CVSS v4.0 (Base Score, CVSS-B)		
Metric	**Rating**	**Explanatory Statement**
Privileges Required (PR)	High	Before exploiting the vulnerability, an attacker does require access to the source code.
User Interaction (UI)	None	The vulnerability can be exploited independently of other users.
Vulnerable System Impact Metrics		
Confidentiality Impact (VC)	Low	By exploiting the vulnerability, an attacker can potentially view sensitive data, but cannot influence the scope of the data.
Integrity Impact (VI)	Low	By exploiting the vulnerability, an attacker can potentially alter sensitive data, but cannot influence the scope of the data.
Availability Impact (VA)	None	Exploiting this vulnerability has no impact on the availability of the application.
Subsequent System Impact Metrics		
Confidentiality Impact (SC)	Low	By exploiting the vulnerability, an attacker can view sensitive data in subsequent systems, but cannot influence the scope of the data.
Integrity Impact (SI)	Low	By exploiting the vulnerability, an attacker can alter sensitive data from subsequent systems, but cannot influence the scope of the data.
Availability Impact (SA)	None	Exploiting this vulnerability has no impact on the availability of subsequent systems.
CVSS-B	5,1 CVSS:4.0/AV:N/AC:L/AT:N/PR:H/UI:N/VC:L/VI:L/VA:N/SC:L/SI:L/SA:N	

C_02 CORS Misconfiguration (Remediated)

Retest Status: The affected code locations were reviewed manually in the current version of the source code. The permissive CORS configuration was no longer present.

Description: The application uses an overly permissive postMessage() configuration that allows messages to be sent to any origin using a wildcard ('*') target origin.

This configuration bypasses the browser's Same-Origin Policy protection mechanism, which normally serves as a security boundary for cross-origin communication.

Impact

This configuration significantly increases the risk for cross-origin message interception and information disclosure attacks. An attacker could create a malicious website or inject malicious iframes that listen for postMessage events while an authenticated user visits the compromised page. Since all origins are permitted as message recipients, any website can intercept the transmitted messages containing potentially sensitive client parameters, extension state information, or other application data. Attackers may therefore be able to perform extension fingerprinting, gather sensitive application state, and potentially manipulate cross-origin communication flows if the receiving end processes untrusted messages.

Example

A CORS misconfiguration was detected in the listed files:

Table 6: CORS-Misconfiguration

Source Code directory	**Path and filename**	**Line**
directory_name	path/to/DebugPane.tsx	740, 743, 1304, 1308
	path/to/SettingsGlobalSearchContainer.tsx	30
	path/to/App.tsx	317, 334
	path/to/writeToClipboard.tsx	12

Figure 2: CORS-Misconfiguration in path/to/App.tsx

```
298   function useExtensionSessionSync(
299     clientCoreParameters: ClientCoreParameters | undefined,
300     setClientCoreParameters: (newCCP: ClientCoreParameters) => void,
301   ) {
302     const previousClientCoreParametersRef = useRef<ClientCoreParameters>();
303
304     // Notify extension when CCP change
305     useEffect(() => {
306       if (
307         !clientCoreParameters ||
308         deepEqual(previousClientCoreParametersRef.current, clientCoreParameters)
309       ) {
310         return;
311       }
312       previousClientCoreParametersRef.current = clientCoreParameters;
313       const changeMessage: ClientCoreParametersChangeMessage = {
314         type: 'clientCoreParametersChange',
315       };
316       // try to update extension (see clientCoreParameterStorage.ts in ext)
317       window.postMessage(changeMessage, '*');
318     }, \[clientCoreParameters]);
```

OWASP Top 10	2021 A05 – Security Misconfiguration
Recommendation	Only domains that actually require access should be configured in the CORS settings. Wildcards should not be used.
References	https://portswigger.net/web-security/cors https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Headers_Cheat_Sheet.html#access-control-allow-origin
Severity	Medium

Table 7: Severity Evaluation Vulnerability C_02

Severity evaluation according to CVSS v4.0 (Base Score, CVSS-B)		
Metric	**Rating**	**Explanatory Statement**
Attack Vector (AV)	Network	The vulnerability can be exploited via a network.
Attack Complexity (AC)	Low	The attacker does not need to take any quantifiable measures to actively circumvent or overcome existing security-enhancing precautions in order to exploit the vulnerability.
Attack Require-ments (AT)	None	No special requirements for the configuration of the application need to be met in order to exploit the vulnerability.

Severity evaluation according to CVSS v4.0 (Base Score, CVSS-B)		
Metric	**Rating**	**Explanatory Statement**
Privileges Required (PR)	None	Before exploiting the vulnerability, an attacker does not require authentication on the application.
User Interaction (UI)	None	The vulnerability can be exploited independently of other users.
Vulnerable System Impact Metrics		
Confidentiality Impact (VC)	Low	By exploiting the vulnerability, an attacker can potentially view sensitive data, but cannot influence the scope of the data.
Integrity Impact (VI)	Low	By exploiting the vulnerability, an attacker can potentially alter sensitive data, but cannot influence the scope of the data.
Availability Impact (VA)	None	Exploiting this vulnerability has no potential impact on the availability of the application.
Subsequent System Impact Metrics		
Confidentiality Impact (SC)	None	Exploitation of the vulnerability has no effect on subsequent systems.
Integrity Impact (SI)		
Availability Impact (SA)		
CVSS-B	6,9 CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:L/VI:L/VA:N/SC:N/SI:N/SA:N	

C_03 Weak iOS Keychain Accessibility Configuration (Remediated)

Retest Status: The weak iOS Keychain accessibility configurations identified during the initial assessment were no longer present in the reviewed source code.

Description: During the security assessment, the iOS application was found to implement a weak keychain accessibility configuration. The application uses `kSecAttrAccessibleAfterFirstUnlockThisDeviceOnly` for storing

sensitive login vault secrets, which keeps the encrypted data accessible in device memory from the first unlock until the next device restart. This configuration creates a significant security vulnerability where stored secrets remain accessible even when the device is subsequently locked.

Impact — This weak configuration allows attackers to extract sensitive authentication data through multiple attack vectors. On jailbroken devices, attackers can use readily available tools such as Objection, Frida, or custom keychain dump utilities to access all stored secrets. Physical device access combined with device compromise can lead to exposure of user authentication credentials, stored passwords, session tokens, and other sensitive vault data. The vulnerability effectively bypasses the intended security controls of both the application and iOS keychain protection mechanisms.

Example — The following locations in the source code implement a weak keychain accessibility configuration:

Table 8: Weak iOS Keychain Accessibility Configuration

Path and Filename	Line
path/to/SafariWebExtensionHandler.swift	76
path/to/NotificationService.swift	77
path/to/StorageAssembly.swift	132
path/to/SecretStorageBuilder.swift	25

OWASP Top 10 — 2024 M9 – Insecure Data Storage

Recommendation — The storage of passwords in the Keychain with the configuration `kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly` or `kSecAccessControlBiometryCurrentSet` is recommended. This way, the Keychain can only be accessed when a passcode is set and the device is unlocked. Additionally, it is recommended to encrypt the values as an extra layer of protection. This makes it impossible for an attacker to access the Keychain entries via a jailbreak or through a data backup.

References — https://owasp.org/www-project-mobile-top-10/2023-risks/m9-insecure-data-storage.html

https://mas.owasp.org/MASTG/tests/android/MASVS-STORAGE/MASTG-TEST-0001/

https://mas.owasp.org/MASTG/iOS/0x06d-Testing-Data-Storage/#the-keychain

https://developer.apple.com/documentation/security/keychain_services/keychain_items/item_attribute_keys_and_values#1679100

https://www.securing.pl/en/stealing-your-apps-keychain-entries-from-locked-iphone/

Severity — Medium

Table 9: Severity Evaluation Vulnerability C_03

Severity evaluation according to CVSS v4.0 (Base Score, CVSS-B)		
Metric	**Rating**	**Explanatory Statement**
Attack Vector (AV)	Physical	To exploit the vulnerability, an attacker requires physical access to the iOS device in order to read out the Keychain.
Attack Complexity (AC)	High	To exploit the vulnerability, an attacker must first gain possession of the device and jailbreak it in order to be able to read out the iOS Keychain data.
Attack Require-ments (AT)	None	No special requirements for the configuration of the application need to be met in order to exploit the vulnerability.
Privileges Required (PR)	None	Before exploiting the vulnerability, an attacker does not require authentication on the application.
User Interaction (UI)	Passive	To exploit the vulnerability, a user must have unlocked their device.
Vulnerable System Impact Metrics		
Confidentiality Impact (VC)	High	By exploiting the vulnerability, an attacker can read out particularly sensitive data.
Integrity Impact (VI)	High	By exploiting the vulnerability, an attacker can modify particularly sensitive data.
Availability Impact (VA)	None	Exploiting this vulnerability has no impact on the availability of the application.
Subsequent System Impact Metrics		
Confidentiality Impact (SC)	None	Exploitation of the vulnerability has no effect on subsequent systems.
Integrity Impact (SI)		
Availability Impact (SA)		

Severity evaluation according to CVSS v4.0 (Base Score, CVSS-B)		
Metric	**Rating**	**Explanatory Statement**
CVSS-B	5,3 CVSS:4.0/AV:P/AC:H/AT:N/PR:N/UI:P/VC:H/VI:H/VA:N/SC:N/SI:N/SA:N	

3.2.2. Mapping Vulnerabilities to OWASP Top 10 Risks

The results of the web application penetration test are mapped to the OWASP Top 10 categories in the following.

Table 10: Mapping Vulnerabilities to OWASP Top 10 Risks

Risk	Fail/Pass
2021 A01 – Broken Access Control	Pass
2021 A02 – Cryptographic Failures	Pass
2021 A03 – Injection	Pass
2021 A04 – Insecure Design	Pass
2021 A05 – Security Misconfiguration	Pass*
2021 A06 – Vulnerable and Outdated Components	Pass
2021 A07 – Identification and Authentication Failures	Pass
2021 A08 – Software and Data Integrity Failures	Pass
2021 A09 – Security Logging and Monitoring Failures	Pass
2021 A10 – Server-Side Request Forgery	Pass

Legend: *There are informational findings for this category; see chapter below.

3.2.3. Informational Findings

During the original audit from November 2025, some problems were identified that could not be clearly classified as vulnerabilities since they posed no direct risk. However, the general security level of the source code can be increased by addressing these findings. These were not retested during the course of the retest from September 2026.

H_01 Improvement Notes on the Docker Configurations

Description: Docker files are used, which leave room for improvement in their configuration. The problems identified are summarized and assigned in the table in the Example section.

Impact: The Docker configurations described do not lead directly to a vulnerability. Therefore, they do not pose an immediate threat. However, fixing the problems can help to improve the security of the Docker containers created and is therefore recommended.

Example: The following configuration problems were detected:

Missing use of a dedicated user

Applications inside a Docker container are running as root by default. It is recommended to follow the security best-practice “principle of least privilege”, configuring the process to use a non-privileged user, to prevent privilege escalation attacks.

Table 11: List of Files with Missing Use of a Dedicated User

Path and File
path/to/first/Dockerfile
path/to/second/Dockerfile
path/to/third/Dockerfile
path/to/fourth/Dockerfile

Lack of restriction of capabilities

A missing restriction of the Linux capabilities assigned to the container was detected. Different capabilities might open up different attack vectors and expand the attack surface.

Table 12: List of Files with Lack of Restriction of Capabilities

File
path/to/first/docker-compose.yml
path/to/second/docker-compose.yml
path/to/third/docker-compose.yml
path/to/fourth/docker-compose.yml

OWASP Top 10: 2021 A05 – Security Misconfiguration

Remediation: The Docker configuration should follow current best practices. In addition, a process should be established that Docker files are regularly checked for correct configuration. The following remediations can be used to address the identified problems.

Table 13: Remediation of the Configuration Problems

Problem	Remediation
Missing use of a dedicated user	Create a user with non-root privileges inside of the Docker container and start the application in the context of this user.
Lack of restriction of capabilities	The best practices recommended to remove all capabilities (--cap-drop=all) and explicitly add those that are needed (--cap-add)

References https://cheatsheetseries.owasp.org/cheatsheets/Docker_Security_Cheat_Sheet.html

H_02 Missing TLS Certificate Pinning Implementation

Description The code does not verify the authenticity of the TLS certificate of the backend systems itself. Methods for ensuring authenticity, such as certificate pinning, are not used.

Impact A possible attack scenario could arise if an attacker with direct access to an unlocked or unsecured device installs a new system certificate controlled by the attacker and sets up a proxy. Alternatively, the attacker could trick a user into installing such a certificate themselves. This would allow the attacker to take up a man-in-the-middle (MitM) position between the app and the server and decrypt, read, and modify communications.

Example Missing TLS Certificate Pinning was found in the following files:

Table 14: Missing Certificate Pinning

Source Code Directory	File
first_directory	path/to/Info.plist
second_directory	path/to/Info.plist
third_directory	path/to/Info.plist
fourth_directory	path/to/Info.plist

OWASP Top 10 2016 M3 – Insecure Communication

Remediation The app should use certificate pinning to check whether the correct certificate is being used to establish the connection. The app should not leave this check to the operating system or a browser, as it is not possible to control the user settings here.

Before integrating certificate pinning, it should be noted that this can lead to restrictions for users. If they are in a network environment that checks network

traffic by breaking the TLS connection, the app would prevent communication. The security gain should nevertheless be taken into account due to the context of the data being handled.

4. RESET CHANGED CONFIGURATIONS

This chapter is intended as a reminder to reverse configurations that were specifically adapted for the pentest. The following section highlights some important points that should be part of the cleanup process. However, this checklist is not intended to be exhaustive.

General information:

- ☐ Any user with access to the repository and source code should be deleted.

It is recommended (for a possible follow-up penetration test) to document all configuration changes directly during the adaptation process, both before and during a penetration test. This makes it easier to keep track of the changes and restore the original configuration.

5. APPENDIX A: VERSIONS AND DIRECTORIES

5.1. Version History

Table 15: Version History

Version	Date	Author	Changes
1.7	17.09.2026	Schäpe, secuvera	Components and paths have been redacted
1.6	10.09.2026	Schäpe, secuvera	Report finalisation
1.5	10.09.2026	Feuersänger, secuvera	Proofreading
1.4	09.09.2026	Metzger, secuvera	Approval of document amendments
1.3	09.09.2026	Schäpe, secuvera	Worked in quality assurance review amendments
1.2	09.09.2026	Metzger, secuvera	Technical quality assurance review of retest
1.1	09.09.2026	Schäpe, secuvera	Retest documented
1.0	17.11.2025	Schäpe, secuvera	Report finalisation
0.9a	16.10.2025	Schäpe, secuvera	Report revised after consultation with the client
0.9	25.09.2025	Schäpe, secuvera	Report finalisation
0.8	24.09.2025	Nicklaß, secuvera	Proofreading
0.6	22.09.2025	Schäpe, secuvera	Worked in quality assurance review amendments
0.5	22.09.2025	Schäfer, secuvera	Technical quality assurance review
0.4	19.09.2025	Schäpe, secuvera	Finalised report
0.3	18.09.2025	Schäpe, secuvera	Wrote summary
0.2	11.09.2025	Schäpe, Volosnik, secuvera	Documented Static Source Code Analysis

Version	Date	Author	Changes
0.1	11.09.2025	Schäpe, Volosnik, secuvera	Document initialisation

5.2. List of Figures

5.3. List of Tables

6. APPENDIX B: DESCRIPTION OF SECURITY LEVEL EVALUATION

6.1. Security Level Evaluation during Static Source Code Analysis

The security level of a tested API is evaluated according to the following table.

Table 16: Security Level Evaluation for Static Source Code Analysis

Security Level	Criteria
Very High	Is chosen if no vulnerability was identified.
High	Is chosen if only vulnerabilities with low severity were identified.
Medium	Is chosen if only vulnerabilities with low and medium severity were identified.
Low	Is chosen if any vulnerability with high severity was identified.
Critical	Is chosen if any vulnerabilities with critical severity was identified.

Please note: Since informational findings do not pose a direct risk, they are not taken into account when calculating the security level.