secuvera
Cybersicherheit. Nachhaltig.

PENETRATION TEST REPORT

VERSION 1.0 | August 16th, 2023

Closing Report for Universitätsbibliothek Leipzig
Beethovenstr. 6
04107 Leipzig

secuvera GmbH
Siedlerstraße 22-24
71126 Gäufelden

Author(s)
Timo Schäpe

TABLE OF CONTENTS

1. SUMMARY

1.1. Source Code Analysis

In the period from July 7th to 18th, 2023, the source code of the VuFind® web application was analyzed on behalf of the Universitätsbibliothek Leipzig (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.

The source code was first checked using the automated static source code analysis scanning tool "Semgrep"[1], and the results were verified manually. Additionally, manual tests were performed to identify weaknesses in the source code. The VuFind® web application was also installed locally to understand, check, and verify findings in the source code. Also, two scripts were written to help verifying assumptions which were made based on findings in the source code.

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

During the source code analysis, four vulnerabilities could be identified.

It was discovered that the web application could be set up to run with the outdated PHP version 7.4.1. This version is no longer supported by the manufacturer ("end of life"). No more updates and, in particular, no more security patches are provided for this version. The security risk of software that is no longer supported must generally be considered critical, so this was assessed as a critical severity vulnerability.

Not hashing passwords of users by default leads to a high severity vulnerability. Although the web application supports hashing passwords, it is not enabled by default.

Another high severity vulnerability results from using an outdated version of the third-party library Laminas-diactoros in version 2.17. A vulnerability is known for this version. Attackers can carry out denial-of-service attacks.

The lack of protection against brute-force attacks in the login mask was rated as a medium severity vulnerability. An attacker is thus able to try out any number of username and password combinations.

In addition, the following three 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.

The web application is not using a strict comparison to check the md5 sum of a file during the upgrade. This could lead to type juggling.

A vulnerability without direct severity is caused by the fact that no requirements are specified by default for the login password. Users can use very simple passwords like "a" or "123", which can facilitate brute-force attacks on the login.

The possibility of prototype polluting function was detected. By adding or modifying attributes of an object prototype, it is possible to create attributes that exist on every object, or replace critical attributes with malicious ones.

[1] https://semgrep.dev/

Table 1: Statistical Summary of Security Levels of all Vulnerabilities

Number of identified vulnerabilities with Severity “Critical”	**1**
Number of identified vulnerabilities with Severity “High”	2
Number of identified vulnerabilities with Severity “Medium”	1
Number of identified vulnerabilities with Severity “Low”	0
Number of identified vulnerabilities with no Severity	3
Total number of identified vulnerabilities	7

Overall, only a critical severity level can be attested to the web application, as one critical vulnerability was found.

2. GENERAL INFORMATION

The test results are separated into different chapters. General remarks can be found in this chapter.

All results are valid for the described tests, software versions, and configuration only. New vulnerabilities or vulnerabilities introduced due to changes in the code 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) is being used to evaluate risks of vulnerabilities identified.[2] 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".

For each vulnerability rating, the according vector string[3] will also be provided. This will facilitate the further processing of the base score determined by the tester within the scope of risk evaluation and handling by the customer. The respectively determined base score represents the external tester's view of the test object. Using the CVSS environmental score, the values can be adapted by the customer to the respective application's context within the scope of the customer's own risk evaluation.

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

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

[3] https://www.first.org/cvss/specification-document#Vector-String

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:

- S_ 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 risk level evaluation. In this report, they are listed using the following nomenclature:

- H_ informational findings during the penetration test.

3. SOURCE CODE ANALYSIS

3.1. Testing Methodology and Goal

In the period from July 7th to 18th, 2023, the source code of the VuFind® web application was analyzed on behalf of the Universitätsbibliothek Leipzig 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 tool "Semgrep" and the results were verified manually. Furthermore, manual checks were performed to be able to identify weaknesses in the source code. The VuFind® web application was also installed locally to understand, check, and verify findings in the source code. Also, two scripts were written to help verify assumptions which were made based on findings in the source code.

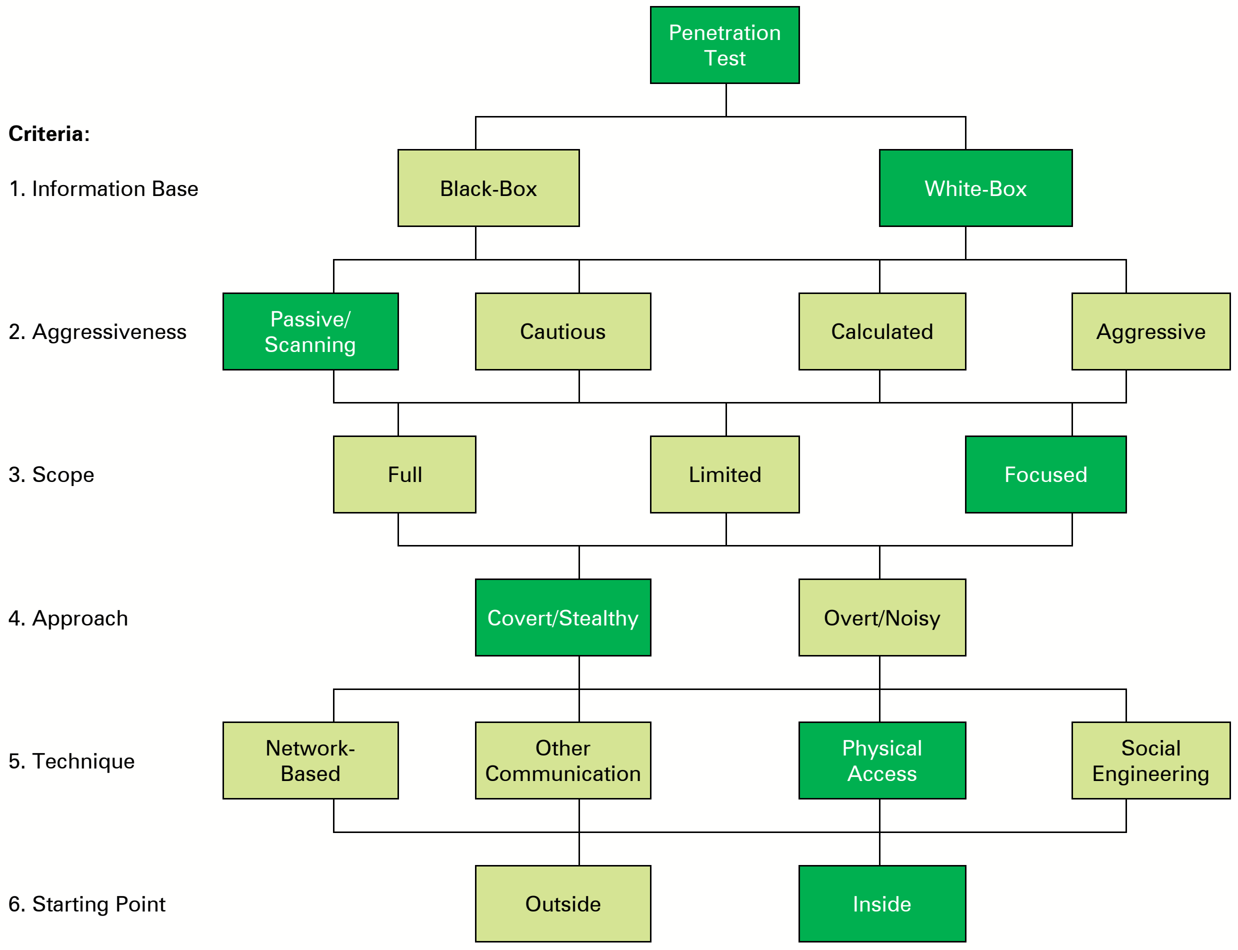
The following information was provided by the customer:

- VuFind 9.0.2;
- Git hash for this release: 203c4ad91ef0613d83dff1281cf4a7532d9c7bcd;
- directories that were not included:
  - data;
  - import;
  - packages;
  - solr;
  - tests.

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

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

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



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

3.2. Course of the Project and Results

All tests were performed as planned, following the methodology described in the previous chapter.

3.2.1. Vulnerability Listing

In order to increase readability and to avoid redundancies, every type of found 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 the 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 an SCA, 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.

S_01	**PHP 7.4.1 Has Reached End of Life**
Description	The minimal PHP version 7.4.1 is defined in the PHP package file *composer.json*. This is an outdated version of the PHP and has passed its official end-of-life (EOL) date and is accordingly no longer supported by the manufacturer. Also, various vulnerabilities are known to this version, and because it reached EOL, security updates are no longer released.
Effect	The outdated PHP version is vulnerable to various vulnerabilities that, for example, an attacker can exploit to cause a denial of service.
Additional info	The support period for PHP 7.4.1 ended 08/22/2022.
OWASP Top 10	2021 A06 – Vulnerable and Outdated Components
Remediation	Increase the PHP version to a current version that is still actively supplied with security updates. For the long-term elimination of the vulnerability, regular checks should be performed to determine whether the software in use is still supported. For this purpose, the announcements published by the respective manufacturer should be subscribed to.
References	https://www.php.net/supported-versions.php https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/
Risk Evaluation	Critical In the case of newly emerging vulnerabilities, there is usually no mapping by the manufacturer as to whether a vulnerability also exists in the software versions that are no longer supported. It is therefore assumed in the assessment that non-publicly known vulnerabilities exist and could be exploited that affect this version and for which no update exists. There is no assessment based on the CVSS score, but the severity is classified as critical based on the behavior described above, as the security risk for software that is no longer supported cannot be assessed.

S_02 Outdated Software Component in Use

Description	During the audit, an outdated version of the *laminas-diactoros* library was found. In the file *composer.json* the version of the library is defined as version 2.17, but there is a known vulnerability to this version.
Effect	*Laminas-diactoros* provides PSR HTTP Message implementations. In version 2.17, an attacker who creates HTTP requests or responses using *laminas-diactoros*, when providing a newline at the start or end of a header key or value, can cause an invalid message. This can lead to denial of service or application errors.
Additional info	Due to the limited amount of time during this audit it was not possible to check all third-party libraries in use. Choosing *laminas-diactoros* was just a random sample out of the list of dependencies.
OWASP Top 10	2021 A06 – Vulnerable and Outdated Components
Remediation	Increase the *laminas-diactoros* version to a current version with security updates. For the long-term elimination of the vulnerability, regular checks should be made to determine whether the software in use has security vulnerabilities and fixes. For this purpose, the announcements published by the respective manufacturer should be subscribed to.
References	https://getlaminas.org/security/advisory/LP-2023-01 https://nvd.nist.gov/vuln/detail/CVE-2023-29530 https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/
Severity Evaluation	High

Table 2: Severity Evaluation Vulnerability S_02

Severity Evaluation according to CVSS v3.1 (Base Score)		
Metric	**Rating**	**Explanatory Statement**
Attack Vector (AV)	Network	A vulnerability exploitable with network access means the vulnerable component is bound to the network stack, and the attacker's path is through the network layer.
Attack Complexity (AC)	Low	Specialized access conditions or extenuating circumstances do not exist. An attacker can expect repeatable success against the vulnerable component.
Privileges Required (PR)	None	The attacker is unauthorized prior to the attack and therefore does not require any access to settings or files to carry out an attack.
User Interaction (UI)	None	The vulnerable code can be exploited without interaction from any user.
Scope (S)	Unchanged	An exploited vulnerability can only affect resources managed by the same authority. In this case, the vulnerable component and the impacted component are the same.
Confidentiality Impact (C)	None	There is no loss of confidentiality within the impacted component.

Severity Evaluation according to CVSS v3.1 (Base Score)		
Metric	**Rating**	**Explanatory Statement**
Integrity Impact (I)	None	There is no loss of integrity within the impacted component.
Availability Impact (A)	High	There is total loss of availability, resulting in the attacker being able to fully deny access to resources in the impacted component.
Score	High	7,5 (CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H)

S_03 Passwords Are Not Hashed by Default

Description

The default configuration of VuFind® does not enforce hashing of passwords in the database. It requires a manual adjustment that passwords are hashed.

Effect

In general, plaintext passwords impose a risk for a user because of password reuse attacks. Additionally, it may facilitate privilege escalation attacks. Hashing passwords is considered as state of the art and a general requirement for storing passwords in databases. Because of this, it may also be a loss of reputation for the vendor of software in case of an incident.

Additional info

Although there is the possibility to enable password hashing in the file *config.ini* manually, it is considered insecure to not enable it by default.

Using the *Install/Home* page to finish a VuFind® installation/update will enable hashing and also encrypt all old plaintext passwords by pressing the button to fix the security issue. However, a quick research showed that 14 of 172 listed VuFind® instances[5] out there have not fixed the security issue yet (tested on 07/18/2023).

The vulnerability assessment with CVSS rating in the table below considers a generic scenario that an attacker gains access to the database. The attacker might be an administrator with read-only access (insider attack) or an attacker exploiting an SQL injection. These are just examples for demonstration.

OWASP Top 10

2021 A05 – Security Misconfiguration

Remediation

Enabling the password hashing by default will solve this vulnerability and is best practice. Password hashing should also be mandatory and the config flag for enabling/disabling the hashing should be removed.

References

https://owasp.org/www-community/vulnerabilities/Password_Plaintext_Storage

https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

Severity Evaluation High

[5] https://vufind.org/wiki/community:installations

Table 3: Severity Evaluation Vulnerability S_03

Severity Evaluation according to CVSS v3.1 (Base Score)		
Metric	**Rating**	**Explanatory Statement**
Attack Vector (AV)	Network	A vulnerability exploitable with network access means the vulnerable component is bound to the network stack, and the attacker's path is through the network layer.
Attack Complexity (AC)	Low	An attacker can simply use the found username/password combinations if they gain access to the stored plaintext passwords in the database through another vulnerability, e.g. SQL injection.
Privileges Required (PR)	None	The attacker is unauthorized prior to the attack and therefore does not require any access to settings or files to carry out an attack.
User Interaction (UI)	None	The vulnerable code can be exploited without interaction from any user.
Scope (S)	Unchanged	An exploited vulnerability can only affect resources managed by the same authority. In this case, the vulnerable component and the impacted component are the same.
Confidentiality Impact (C)	High	Access to restricted information is obtained, the disclosed information presents a direct, serious impact.
Integrity Impact (I)	None	There is no loss of integrity within the impacted component.
Availability Impact (A)	None	There is no impact to availability within the impacted compo-nent.
Score	High	7.5 (CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N)

S_04 Brute-Force Attack on Login Form Possible

Description	The web application does not provide protection against brute-force attacks when the database authentication is used. An attacker is thus able to try any number of username and password combinations.
Effect	An attacker would be able to guess the password to a known user name by trial and error and thus gain access to the user's account.
Additional info	To verify the assumption that no rate limiting for the login page is implemented, a python script was written and started to stress the local installed VuFind® instance. In combination with H_02 No Password Policy by Default, a brute-force attack is likely to be successful.
OWASP Top 10	2021 A07 – Identification and Authentication Failures
Remediation	Repeated invocation of the login function should be limited. For example, the use of so-called tar pits or CAPTCHAs would be possible. It would also be

	conceivable to temporarily block accounts after several failed attempts (e.g. ten attempts).
References	https://owasp.org/www-community/controls/Blocking_Brute_Force_Attacks https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html#login-throttling
Severity Evaluation	Medium

Table 4: Severity Evaluation Vulnerability S_04

Severity Evaluation according to CVSS v3.1 (Base Score)		
Metric	**Rating**	**Explanatory Statement**
Attack Vector (AV)	Network	A vulnerability exploitable with network access means the vulnerable component is bound to the network stack, and the attacker's path is through the network layer.
Attack Complexity (AC)	Low	Specialized access conditions or extenuating circumstances do not exist. An attacker can expect repeatable success against the vulnerable component.
Privileges Required (PR)	None	The attacker is unauthorized prior to the attack and therefore does not require any access to settings or files to carry out an attack.
User Interaction (UI)	None	The vulnerable code can be exploited without interaction from any user.
Scope (S)	Unchanged	An exploited vulnerability can only affect resources managed by the same authority. In this case, the vulnerable component and the impacted component are the same.
Confidentiality Impact (C)	Low	There is some loss of confidentiality. After a successful attack, the attacker knows a user's password and can view the user's data.
Integrity Impact (I)	None	There is no loss of integrity within the impacted component.
Availability Impact (A)	None	There is no impact to availability within the impacted component.
Score	Medium	5,3 (CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N)

3.2.2. Assignment of Vulnerabilities to the OWASP Top 10

In the following, the result of the source code analysis is assigned to the OWASP Top 10 categories presented in the description of the approach.

Table 5: Result Referencing Source Code Analysis on 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	Fail
2021 A06 – Vulnerable and Outdated Components	Fail
2021 A07 – Identification and Authentication Failures	Fail
2021 A08 – Software and Data Integrity Failures	Pass
2021 A09 – Security Logging and Monitoring Failures	Pass
2021 A10 – Server-Side Request Forgery	Pass

3.2.3. Informational Findings

During the audit, some problems were identified that cannot be clearly classified as vulnerabilities since they pose no direct risk. However, the general security level of the source code can be increased by addressing these findings.

H_01 Type Juggling Possible

Description	PHP is a loosely typed language, which means it tries to predict the programmer's intent and automatically converts variables to different types whenever it seems necessary. For example, a string containing only numbers can be treated as an integer or a float. However, this automatic conversion (or type juggling) can lead to unexpected results, especially when comparing variables using the '==' operator, which only checks for value equality (loose comparison), not type and value equality (strict comparison).
Effect	PHP type juggling vulnerabilities arise when loose comparison (== or !=) is employed instead of strict comparison (=== or !==) in an area where the attacker can control one of the variables being compared. This vulnerability can result in the application returning an unintended answer to the true or false statement, and may lead to unexpected behavior of the application.
Recommendation	Make sure comparisons involving md5 values are strict (use === not ==) to avoid type juggling issues.
References	https://www.php.net/manual/en/types.comparisons.php https://www.php.net/manual/en/language.types.type-juggling.php https://owasp.org/www-pdf-archive/PHPMagicTricks-TypeJuggling.pdf

H_02 No Password Policy by Default

Description	The web application does not implement any policy for passwords by default, so users can arbitrarily choose simple passwords.
Effect	It is much easier for an attacker to guess passwords of users if they could choose simple passwords. Should an attacker come into possession of valid credentials, they could retrieve and modify arbitrary files on behalf of the user.
OWASP Top 10	2021 A07 – Identification and Authentication Failures
Recommendation	A more suitable and restrictive password policy should be created and technically implemented. No “universal” policy can be recommended here, but a policy corresponding to the data handled and thus the necessary security level must be defined. Also, this policy should be enabled by default.
References	
	https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html#implement-proper-password-strength-controls

H_03 Prototype Pollution Possible

Description	By adding or modifying attributes of a JavaScript object prototype, it is possible to create attributes that exist on every object, or replace critical attributes with malicious ones. This can be problematic if the web application depends on existence or non-existence of certain attributes, or uses pre-defined attributes of object prototype (such as hasOwnProperty, toString or valueOf).
Effect	Although prototype pollution is often unexploitable as a standalone vulnerability, it lets an attacker control properties of objects that would otherwise be inaccessible. If the application subsequently handles an attacker-controlled property in an unsafe way, this can potentially be chained with other vulnerabilities.
Recommendation	Like many other security vulnerabilities, attackers exploit prototype pollution bugs through user input in web applications and sending their malicious code in text fields, headers, and files. One popular kind of defense is to create blocklists where developers remove risky fields from input strings. Checking the user input against those lists could be the first step to mitigate this issue.
	Other possible mitigations might be: freezing the object prototype, using an object without prototypes (via Object.create(null)), blocking modifications of attributes that resolve to object prototype, using Map instead of object.
References	https://portswigger.net/web-security/prototype-pollution
	https://cheatsheetseries.owasp.org/cheatsheets/Prototype_Pollution_Prevention_Cheat_Sheet.html

4. APPENDIX A: VERSIONS AND DIRECTORIES

Table 6: Version History

Version	Date	Author	Changes
1.0	08/16/2023	Ruben Konrad, secuvera	Final Report
0.9	07/26/2023	Ruben Konrad, secuvera	Report finalization, version to be harmonized with the customer
0.8	07/26/2023	Feuersänger, secuvera	Proofreading
0.7	07/24/2023	Ruben Konrad, secuvera	Approval of document amendments
0.6	07/21/2023	Timo Schäpe, secuvera	Worked in quality assurance review amendments
0.5	07/20/2023	Ruben Konrad, secuvera	Technical quality assurance review
0.4	07/20/2023	Timo Schäpe, secuvera	Finalized report
0.3	07/20/2023	Timo Schäpe, secuvera	Documented summary
0.2	07/19/2023	Timo Schäpe, secuvera	Documented general information source code analysis
0.1	07/19/2023	Timo Schäpe, secuvera	Document initialization

4.1. List of Figures

4.2. List of Tables

5. APPENDIX B: DESCRIPTION OF SECURITY LEVEL EVALUATION

5.1. Security Level Evaluation during Source Code Analysis

The security level of the source code is calculated according to the identified vulnerabilities and their severities.

Table 7: Security Level Evaluation for Source Code Analysis

Security Level	**Criteria for The Source Code Security Level**
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 considered when calculating the security level.