

CONFIDENTIAL

CYBERSECURITY AUDIT REPORT

Version 1.1

This document details the process and results of the security audit performed by CyStack on behalf of Demlabs from 18/09/2024 to 22/10/2024.

Prepared for

Demlabs

Prepared by

Vietnam CyStack Joint Stock Company

© 2024 CyStack. All rights reserved.

Portions of this document and the templates used in its production are the property of CyStack and cannot be copied (in full or in part) without CyStack's permission.

While precautions have been taken in the preparation of this document, CyStack the publisher, and the author(s) assume no responsibility for errors, omissions, or for damages resulting from the use of the information contained herein. Use of CyStack's services does not guarantee the security of a system, or that computer intrusions will not occur.

Contents

1 Executive Summary	4
1.1 Key Findings	4
1.2 Limitations	4
1.3 Assessment Components	5
2 Methodology	6
3 Dashboard	8
4 Recommendations	9
5 Code Review Details	10
5.1 Steps to Conduct	10
5.2 Manual Review Results	10
6 Vulnerability Details	15
7 Appendix	16
Appendix A - Vulnerability Severity Ratings	16
Appendix B - Vulnerability Categories	17
Appendix C - Security Assessment For Source Code Review	18

Confidentiality Statement

This document is the exclusive property of **Demlabs (Cellframe team)** and **CyStack Vietnam Joint Stock Company (CyStack)**. This document contains proprietary and confidential information. Duplication, redistribution, or use, in whole or in part, in any form, requires consent of both **Demlabs** and **CyStack**.

CyStack may share this document with auditors under non-disclosure agreements to demonstrate security audit requirement compliance.

Disclaimer

A security audit is considered a snapshot in time. The findings and recommendations reflect the information gathered during the assessment, not any changes or modifications made outside of that period.

Time-limited engagements do not allow for a full evaluation of all security controls. CyStack prioritized the assessment to identify the weakest security controls an attacker would exploit. CyStack recommends Cellframe conducting similar assessments on an annual basis by internal or third-party assessors to ensure the continued success of the controls.

Version History

Version	Date	Release notes
1.0	22/10/2024	Final audit report for Cellframe project exported for Demlabs. No critical vulnerabilities were found.
1.1	13/12/2024	Demlabs updated source code with patches for High and Medium issues found in static code analysis.

Contact Information

Company	Representative	Position	Email
Demlabs	Dmitry Gerasimov	General Director	naeper@demlabs.net
CyStack	Nguyen Ngoc Anh	Sales Manager	anhhtn@cystack.net

Auditors

Fullname	Role	Email address
Nguyen Huu Trung	Head of Security	trungnh@cystack.net
Nguyen Van Huy	Auditor	
Vu Trong Khoi	Auditor	
Phung Phuong Nam	Auditor	
Nguyen Ba Anh Tuan	Auditor	
Ha Minh Chau	Auditor	

Executive Summary

From 18/09/2024 to 22/10/2024, Cellframe engaged CyStack to evaluate the security posture of its infrastructure compared to current industry best practices. This security audit is chiefly based on Source Code Review methodology. Conducted security assessments in this audit project strictly follows *OWASP Code Review Guide* and customized test cases from CyStack.

CyStack's security assessment for Demlabs focused on evidence, which confirmed that Cellframe properly functions as a quantum-resistant blockchain platform, as well as security issues that are typical to dApps. The assessment emphasized remediation over analyzing exploitability, including issues reported by tools. This means that less time was spent determining how specific security flaws might be exploited and more time identifying as many possible security issues and associated remediation as time allowed. The audit results also included a cursory review of dependent libraries and recommendations for improving software assurance practices at Cellframe.

1.1 Key Findings

CyStack did not find any proof that indicates vulnerable usage and storage of users' wallet private keys in Cellframe nodes, nor any critical severity issues that would undermine the cryptography implementation and security of confidential transactions. A great number of coding issues were found after the source code static analysis. Most of these issues were related to Overflow vulnerabilities and Dereference of a Null pointer. Key findings from the automated static code analysis included:

- 5 Low issues related to Improper Null Termination;
- 48 Medium issues, most of which are Overflow vulnerabilities and Pointer Usage flaws;
- 1 High issue referred to the Potential Negative Number Used as Index.

However, after thorough manual secure code reviews from CyStack, no findings could be exploited practically performed or bypassed against the current safe checks.

1.2 Limitations

Due to the strict timeframe of the project, security vulnerabilities in their parties and open-source library dependencies might have not been discovered.

1.3 Assessment Components

Source Code Review

Source code contains the most detailed information about an application. Source code review allows security researchers to understand thoroughly how an application operates and performs. Researchers then can search for design flaws and security vulnerabilities in the application.

The safety and security assessment for application source code includes automated and manual tests. For automated tests, static code analysis tools are used to identify dead code, unsafe coding patterns and the usage of libraries or plugins with publicly known vulnerabilities. Automated tests also search for the existence of hard-coded sensitive information such as passwords, database connection strings, private keys for third-party services, etc.

Manual tests focus on analyzing the implementation of the application’s operational logic and functional components to detect critical vulnerabilities, which are possibly related to user input validation, unsafe database querying, unsafe file handling, etc. or business logic flows. People who perform manual tests are security researchers.

Scope

Initial target		
Assessment	Details	Type
Source Code Review	cellframe-node (all .h, .c, .cpp files in <i>sources</i> and <i>conftool</i>)	Source code
Source Code Review	dap-sdk (all .h, .c files, EXCEPT for those in <i>crypto</i> , <i>3rdparty</i> , <i>avrestream</i> and all subfolders named <i>test/tests</i>)	Source code
Re-test target		
Source Code Review	cellframe-node (all .h, .c, .cpp files in <i>sources</i> and <i>conftool</i>)	Source code
Source Code Review	dap-sdk (all .h, .c files, EXCEPT for those in <i>crypto</i> , <i>3rdparty</i> , <i>avrestream</i> and all subfolders named <i>test/tests</i>)	Source code

Scope Exclusions

Any other directories than those listed in the table Scope, for every repository.

Client Allowances

Cellframe did not provide any allowances to assist with the testing.

Methodology

CyStack performs a two-part for an application Source Code Review. The first part is an implementation review. During this part, CyStack focuses on validating specifications from the application documentation, adhering to its implementation. Also, issues related to cryptography and performance will be carefully looked for. The second part is a source code review for vulnerabilities using static and dynamic analysis and fuzz testing. According to the security issues, found from automated and manual tests, CyStack then reproduces concrete test cases for each to verify whether these issues are vulnerabilities and decide their severity levels. In the second part, security errors within the application implementation, for example, stack-based buffer overflows, data races, memory use-after-free issues, memory leaks, runtime error conditions, and business logic circumventions, will be researched. In addition, the review includes a cursory assessment of dependent third-party open source-code libraries used in the application. From the audit results, CyStack supports the developer team in understanding the root causes and provides the developers with solutions to prevent the repetition of similar bugs and vulnerabilities and other recommendations for improving software assurance practices. We aim to provide the most complete and timely support to the developer team to ensure that the application source code is always up to the maximum level of safety. The process for Review Source Code service from CyStack involves seven (7) main steps as follows:

Phase 1: Preparation

CyStack worked with Demlabs to clarify targets for the Source Code Review assessment, identify types of vulnerabilities, which are most important to them and understand the goal of this assessment. This collaborative process was used to:

- Gain an overview of the application
- Develop scope for the engagement
- Determine a sufficient testing window
- Determine the risk levels associated with each asset
- Gather shareable documentation covering the implementation of the application
- Identify the areas of scope that researchers should pay special attention to
- Identify what types of vulnerabilities that the customer is most interested in testing for

Phase 2: Discovery

CyStack performs a preliminary review of the source code and compares implementations in the source code with technical specifications or documentation provided by Demlabs to clarify the features, logic and operating procedures of the application, application type, language or framework. application deployment, application design and available security mechanisms, etc. CyStack will communicate directly with Demlabs throughout this period.

Phase 3: Automatic source code analysis

After the above two stages, CyStack conducts automatic analysis and scanning of the source code provided with available and self-developed tools. The results of automatic analysis of the source code show preliminary weaknesses as well as possible vulnerabilities in the source code. In addition, identifying application entry points, third-party libraries or plugins used in the application, or the existence of sensitive information (such as passwords, database connection strings, etc.) data, private keys for APIs or third-party services) are stored directly on

the system.

Phase 4: Threat modelling

Based on the gathered information, CyStack implements threat modelling to the application. Threat modelling includes the range of attack vectors, classification, and threat classifiers of identified threats, to provide a clear view of the level of risk by priority. With the threat model, researchers can prioritize testing and detailed evaluation of functions that are important or at high risk of being exploited.

Phase 5: Manual review and exploitation

Security researchers at CyStack directly evaluate the source code using both static and dynamic analysis methods. Static analysis means that researchers directly read and identify inappropriate pieces of code in the source code. At the same time, the researchers perform dynamic analysis, which means analyzing the processes performed during the application's operation, as well as finding ways to exploit the weaknesses previously found in the source code to conclude about the source code's security. Once the vulnerability is discovered, the researchers will build the exploit code and

save it as exploit proof to exchange and agree on a solution for the customer at a later stage. Customers will be notified of the discovered critical vulnerabilities in time for early patching.

Phase 6: Remediation proposal

Detected vulnerabilities are aggregated and notified to customers. During this phase, security researchers at CyStack will directly discuss with the application developers team to come up with a solution that best suits the application design and development infrastructure. application implementation, as well as customer needs, principles and standards. The solution can be temporary (mitigation) or definitive (remediation), depending on the specifics of the application design and application deployment system. Vulnerabilities will be prioritized by severity to ensure maximum application security in the product environment.

Phase 7: Reporting

After completing every security assessment for the application source code, CyStack will send a final report to the customer. The report includes an executive summary of audit results and detailed descriptions of found vulnerabilities.

Dashboard

Maintaining a healthy security posture requires constant review and refinement of existing security processes. Running a CyStack Security Audit allows Demlabs's internal security team to not only uncover specific vulnerabilities but gain a better understanding of the current security threat landscape.

Vulnerabilities by severities

No warnings from static analysis tools were exploitable, and no other vulnerabilities were found.

Vulnerabilities by assets

No warnings from static analysis tools were exploitable, and no other vulnerabilities were found.

Vulnerabilities by CWE

No warnings from static analysis tools were exploitable, and no other vulnerabilities were found.

Table of vulnerabilities

No warnings from static analysis tools were exploitable, and no other vulnerabilities were found.

Recommendations

Based on the results of this assessment, CyStack has the following high-level key recommendations:

Key recommendations	
Issues	<p>After the source code review for cellframe-node and dap-sdk, CyStack did not detect any severe security issues and design flaws. CyStack could conclude that Cellframe is a well-designed blockchain platform, equipped with post-quantum cryptography for encryption/decryption. Although many coding issues were found through static analysis, CyStack has manually reviewed and found no potential attack vectors to exploit these issues. No vulnerabilities were identified during the project.</p>
Recommendations	<ul style="list-style-type: none"> • Review the all findings from automated static code analysis. • Refactor code with safer functions/methods, if possible. • Thoroughly re-assess every input validation mechanisms before unsafe functions.
References	<ul style="list-style-type: none"> • https://googleprojectzero.github.io/0days-in-the-wild/rca.html • https://dl.acm.org/doi/10.5555/2530475 • https://medium.com/@AlexanderObregon/secure-coding-practices-in-c-12b756af90fe

Code Review Details

5.1 Steps to Conduct

CyStack analyzed the source code using a variety of static and dynamic analysis tools. Specifically, CyStack:

1. Statically analyzed the given source code with [Clang Static Analyzer](#) and [Snyk](#).
2. Manually reviewed the code with IDE and GDB debugger. Built test case for each function in Cellframe project. CyStack firstly focused on if any unsafe functions were used, then checked on the business logic of important mechanism such as stream encryption/decryption, data validation, etc.

5.2 Manual Review Results

Via Snyk, CyStack found a lot of potential coding issues in Cellframe, including using unsafe functions, overflows, null pointers, etc.

However, none of these issues were directly exploitable, or able to bypass against current input validation.


Demlabs later applied fixes on all findings from Snyk. The table below shows the state of remediation for issues found in cellframe-node and dap-sdk via Snyk:

#	Name	Affected Location	Severity	T1	T2
1	Potential Negative Number Used as Index	dap-sdk/net/client/dap_client_pv.c, line 881	HIGH	■	■
2	Integer Overflow	dap-sdk/core/src/dap_file_utils.c, line 665	MEDIUM	■	■
3	Integer Overflow	dap-sdk/global-db/dap_global_db_driver_pgsql.c, line 420	MEDIUM	■	■
4	Integer Overflow	dap-sdk/global-db/dap_global_db_driver_pgsql.c, line 505	MEDIUM	■	■
5	Integer Overflow	dap-sdk/global-db/dap_global_db_driver_sqlite.c, line 817	MEDIUM	■	■
6	Integer Overflow	dap-sdk/core/src/dap_file_utils.c, line 665	MEDIUM	■	■

7	Integer Overflow	dap-sdk/core/src/dap_file_utils.c, line 1141	MEDIUM	■	■
8	Integer Overflow	dap-sdk/core/src/dap_file_utils.c, line 1149	MEDIUM	■	■
9	Integer Overflow	dap-sdk/core/src/dap_file_utils.c, line 1176	MEDIUM	■	■
10	Integer Overflow	dap-sdk/core/src/dap_file_utils.c, line 1176	MEDIUM	■	■
11	Buffer Overflow	dap-sdk/core/src/dap_file_utils.c, line 613	MEDIUM	■	■
12	Buffer Overflow	sources/main_node_tool.c, line 482	MEDIUM	■	■
13	Buffer Overflow	sources/main_node_tool.c, line 482	MEDIUM	■	■
14	Buffer Overflow	sources/main_node_tool.c, line 574	MEDIUM	■	■
15	Buffer Overflow	sources/main_node_tool.c, line 605	MEDIUM	■	■
16	Buffer Overflow	sources/main_node_tool.c, line 623	MEDIUM	■	■
17	Buffer Overflow	dap-sdk/net/stream/stream/dap_stream.c, line 155	MEDIUM	■	■
18	Buffer Overflow	dap-sdk/net/client/dap_client_privt.c, line 411	MEDIUM	■	■
19	Buffer Overflow	dap-sdk/core/src/dap_file_utils.c, line 1141	MEDIUM	■	■
20	Buffer Overflow	dap-sdk/global-db/dap_global_db_driver_sqlite.c, line 545	MEDIUM	■	■
21	Buffer Overflow	dap-sdk/global-db/dap_global_db_driver_sqlite.c, line 557	MEDIUM	■	■
22	Buffer Overflow	dap-sdk/global-db/dap_global_db_driver_sqlite.c, line 569	MEDIUM	■	■
23	Buffer Overflow	dap-sdk/global-db/dap_global_db_driver_sqlite.c, line 644	MEDIUM	■	■

24	Buffer Overflow	sources/main_node_cli.c, line 89	MEDIUM	■	■
25	Path Traversal	sources/main_node_cli.c, line 90	MEDIUM	■	■
26	Path Traversal	dap-sdk/core/src/dap_file_utils.c, line 1141	MEDIUM	■	■
27	Path Traversal	sources/main_node_tool.c, line 574	MEDIUM	■	■
28	User Controlled Pointer	sources/main_node_tool.c, line 574	MEDIUM	■	■
29	Potential buffer overflow from usage of unsafe function	dap-sdk/net/server/enc_server/dap_enc_http.c, line 397	MEDIUM	■	■
30	Potential buffer overflow from usage of unsafe function	dap-sdk/net/stream/ch/dap_stream_ch_pkt.c, line 110	MEDIUM	■	■
31	Potential buffer overflow from usage of unsafe function	dap-sdk/net/stream/ch/dap_stream_ch_pkt.c, line 163	MEDIUM	■	■
32	Potential buffer overflow from usage of unsafe function	dap-sdk/net/stream/ch/dap_stream_ch_pkt.c, line 392	MEDIUM	■	■
33	Potential buffer overflow from usage of unsafe function	dap-sdk/net/server/http_server/dap_http_simple.c, line 530	MEDIUM	■	■
34	Potential buffer overflow from usage of unsafe function	dap-sdk/net/server/notify_server/src/dap_notify_srv.c, line 133	MEDIUM	■	■
35	Potential buffer overflow from usage of unsafe function	dap-sdk/net/server/notify_server/src/dap_notify_srv.c, line 179	MEDIUM	■	■
36	Potential buffer overflow from usage of unsafe function	dap-sdk/core/src/dap_file_utils.c, line 1370	MEDIUM	■	■
37	Potential buffer overflow from usage of unsafe function	dap-sdk/net/server/http_server/dap_http_simple.c, line 292	MEDIUM	■	■

38	Potential buffer overflow from usage of unsafe function	dap-sdk/core/src/unix/linux/dap_network_monitor.c, line 139	MEDIUM	■	■
39	Dereference of a NULL Pointer	dap-sdk/global-db/dap_global_db.c, line 1192	MEDIUM	■	■
40	Dereference of a NULL Pointer	dap-sdk/global-db/dap_global_db.c, line 1210	MEDIUM	■	■
41	Dereference of a NULL Pointer	dap-sdk/net/server/http_server/http_client/dap_http_user_agent.c, line 96	MEDIUM	■	■
42	Dereference of a NULL Pointer	dap-sdk/net/link_manager/dap_link_manager.c, line 575	MEDIUM	■	■
43	Dereference of a NULL Pointer	dap-sdk/net/link_manager/dap_link_manager.c, line 996	MEDIUM	■	■
44	Dereference of a NULL Pointer	dap-sdk/net/link_manager/dap_link_manager.c, line 1036	MEDIUM	■	■
45	Dereference of a NULL Pointer	dap-sdk/net/link_manager/dap_link_manager.c, line 1146	MEDIUM	■	■
46	Dereference of a NULL Pointer	dap-sdk/net/link_manager/dap_link_manager.c, line 1261	MEDIUM	■	■
47	Dereference of a NULL Pointer	dap-sdk/net/link_manager/dap_link_manager.c, line 1261	MEDIUM	■	■
48	Dereference of a NULL Pointer	dap-sdk/net/link_manager/dap_link_manager.c, line 1262	MEDIUM	■	■
49	Dereference of a NULL Pointer	dap-sdk/net/link_manager/dap_link_manager.c, line 1265	MEDIUM	■	■
50	Improper Null Termination	sources/main_node_tool.c, line 574	LOW	■	■
51	Improper Null Termination	dap-sdk/net/server/enc_server/dap_enc_http.c, line 218	LOW	■	■
52	Improper Null Termination	dap-sdk/net/server/json_rpc/rpc_core/src/dap_json_rpc_response_handler.c, line 70	LOW	■	■
53	Improper Null Termination	dap-sdk/net/client/dap_client_privt.c, line 1165	LOW	■	■

54	Improper Null Termination	sources/exh_win32.c, line 89	LOW		
----	---------------------------	------------------------------	-----	---	---

EXPLICATION

 Fixed  Open

Timeline for re-tests:

- T1: First test from 18/09/2024 to 22/10/2024
- T2: Second test from 27/11/2024 to 13/12/2024

Vulnerability Details

No warnings from static analysis tools were exploitable, and no other vulnerabilities were found.

Appendix

Appendix A - Vulnerability Severity Ratings

Severity	CVSS 3.0 score range	Definition
CRITICAL	9.0-10.0	Exploitation is straightforward and usually results in system-level compromise. It is advised to form a plan of action and patch immediately.
HIGH	7.0-8.9	Exploitation is more difficult but could cause elevated privileges and potentially a loss of data or downtime. It is advised to form a plan of action and patch as soon as possible.
MEDIUM	4.0-6.9	Vulnerabilities exist but are not exploitable or require extra steps such as social engineering. It is advised to form a plan of action and patch after high-priority issues have been resolved.
LOW	0.1-3.9	Vulnerabilities are non-exploitable but would reduce an organization's attack surface. It is advised to form a plan of action and patch during the next maintenance window.
INFO	N/A	No vulnerability exists. Additional information is provided regarding items noticed during testing, strong controls, and additional documentation.

Appendix B - Vulnerability Categories

CyStack uses CWE (Common Weakness Enumeration) for the vulnerability categorization. Common Weakness Enumeration (CWE) is a community-developed list of common software security weaknesses. It serves as a common language, a measuring stick for software security tools, and as a baseline for weakness identification, mitigation, and prevention efforts.

CWE categories used by CyStack are listed in the following table:

CWE ID	Name
CWE-16	Security Misconfiguration
CWE-77, CWE-259	Insecure OS Firmware
CWE-79	Cross-Site Scripting (XSS)
CWE-310	Broken Cryptography
CWE-311, CWE-319	Insecure Data Transport
CWE-352	Cross-Site Request Forgery (CSRF)
CWE-359	Privacy Concerns
CWE-400	Application Level Denial Of Service (DoS)
CWE-601	Unvalidated Redirects And Forwards
CWE-693	Lack Of Binary Hardening
CWE-723	Broken Access Control
CWE-729, CWE-922	Insecure Data Storage
CWE-919	Mobile Security Misconfiguration
CWE-929	Injection
CWE-930	Broken Authentication And Session Management
CWE-934	Sensitive Data Exposure
CWE-937	Using Components With Known Vulnerabilities

Appendix C - Security Assessment For Source Code Review

Test ID	Test name	Status
SCR_CONF	Configuration and Deploy Management Testing	Pass
SCR_CONF_1	Test Network Infrastructure Configuration	Pass
SCR_CONF_2	Test Application Platform Configuration	Pass
SCR_CONF_3	Test File Extensions Handling for Sensitive Information	N/A
SCR_CONF_4	Test HTTP Methods	Pass
SCR_CONF_5	Test HTTP Strict Transport Security	Pass
SCR_CONF_6	Test File Permission	Pass
SCR_IDNT	Identity Management Testing	Pass
SCR_IDNT_1	Test Role Definitions	Pass
SCR_IDNT_2	Test User Registration Process	N/A
SCR_IDNT_3	Test Account Provisioning Process	N/A
SCR_ATHN	Authentication Testing	Pass
SCR_ATHN_1	Testing for Credentials Transported over an Encrypted Channel	Pass
SCR_ATHN_2	Testing for Default Credentials	N/A
SCR_ATHN_3	Testing for Weak Lock Out Mechanism	N/A
SCR_ATHN_4	Testing for Bypassing Authentication Schema	Pass
SCR_ATHN_5	Testing for Weak Password Policy	N/A
SCR_ATHN_6	Testing for Weak Password Change or Reset Functionalities	N/A
SCR_ATHN_7	Testing for Weaker Authentication in Alternative Channel	Pass
SCR_ATHZ	Authorization Testing	Pass
SCR_ATHZ_1	Testing Directory Traversal File Include	Pass
SCR_ATHZ_2	Testing for Bypassing Authorization Schema	Pass
SCR_ATHZ_3	Testing for Privilege Escalation	Pass
SCR_SESS	Session Management Testing	Pass
SCR_SESS_1	Testing for Session Management Schema	Pass
SCR_SESS_2	Testing for Session Fixation	Pass

SCR_SESS_3	Testing for Exposed Session Variables	Pass
SCR_SESS_4	Testing for Logout Functionality	N/A
SCR_SESS_5	Testing Session Timeout	Pass
SCR_SESS_6	Testing for Session Puzzling	Pass
SCR_SESS_7	Testing for Session Hijacking	Pass
SCR_INPV	Input Validation Testing	Pass
SCR_INPV_1	Testing for HTTP Verb Tampering	Pass
SCR_INPV_2	Testing for HTTP Parameter pollution	Pass
SCR_INPV_3	Testing for SQL Injection	Pass
SCR_INPV_4	Testing for SSI Injection	Fail
SCR_INPV_5	Testing for Code Injection	Pass
SCR_INPV_6	Testing for Command Injection	Pass
SCR_INPV_7	Testing for Format String Injection	Pass
SCR_INPV_8	Testing for Incubated Vulnerabilities	Pass
SCR_INPV_9	Testing for HTTP Splitting Smuggling	Pass
SCR_INPV_10	Testing for HTTP Incoming Requests	Pass
SCR_INPV_11	Testing for Host Header Injection	Pass
SCR_ERRH	Error Handling	Pass
SCR_ERRH_1	Testing for Improper Error Handling	Pass
SCR_CRYP	Cryptography	Pass
SCR_CRYP_1	Testing for Weak Transport Layer Security	N/A
SCR_CRYP_2	Testing for Padding Oracle	Pass
SCR_CRYP_3	Testing for Sensitive Information Sent Via Unencrypted Channels	Pass
SCR_CRYP_4	Testing for Weak Encryption	Pass
SCR_BUSL	Business Logic Testing	Pass
SCR_BUSL_1	Test Business Logic Data Validation	Pass
SCR_BUSL_2	Test Ability to Forge Requests	Pass
SCR_BUSL_3	Test Integrity Checks	Pass

SCR_BUSL_4	Test for Process Timing	Pass
SCR_BUSL_5	Test Number of Times a Function Can be Used Limits	Pass
SCR_BUSL_6	Testing for the Circumvention of Work Flows	Pass
SCR_BUSL_7	Test Defenses Against Application Misuse	Pass
SCR_BUSL_8	Test Upload of Unexpected File Types	Pass
SCR_BUSL_9	Test Upload of Malicious Files	Pass
SCR_CLNT	Client-side Testing	Pass
SCR_CLNT_1	Testing for Client-side Resource Manipulation	Pass

LEGEND

Pass: Requirement is applicable to the given source code and implemented according to best practices.

Fail: Requirement is applicable to the given source code but not fulfilled.

N/A: Requirement is not applicable to the given source code.