CyStack

PUBLIC ACCESS

CYBERSECURITY AUDIT REPORT

Version 1.2

This document details the security re-evaluation peformed by CyStack on behalf of Busy Technology s.r.o. from 12/08/2022 to 16/08/2022.

Prepared for

Busy Technology s.r.o.

Prepared by

Vietnam CyStack Joint Stock Company

Contents

Confidentiality Statement

This document is the exclusive property of **Busy Technology s.r.o. (Busy Technology 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 **Busy Technology s.r.o.** 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 and 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 Busy Technology 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	04/06/2022	CyStack sent a report with 2 found issues and other recommendations to improve the general security posture of Busy Technology s.r.o.'s products.
1.1	13/08/2022	CyStack retested and confirmed that all found issues were resolved and all recommendations were applied by Busy Technology s.r.o..
1.2	23/08/2022	Busy Technology s.r.o. confirmed to publish the audit report publicly.

Contact Information

Company	Representative	Position	Email
Busy Technology	Robert Michálek	CTO	michalek@busy.technology
Busy Technology	Vladimír Lieger	CEO	lieger@busy.technology
CyStack	Vo Huyen Nhi	Sales Manager	nhivh@cystack.net

Auditors

Fullname	Role	Email address
Nguyen Huu Trung	Head of Security	trungnh@cystack.net
Nguyen Trung Huy Son	Auditor	
Vu Hai Dang	Auditor	
Ha Minh Chau	Auditor	
Nguyen Van Huy	Auditor	
Nguyen Ba Anh Tuan	Auditor	

Executive Summary

From 04/05/2022 to 04/06/2022, Busy Technology 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 Busy Technology s.r.o. focused on evidence, which confirmed that Busy Technology properly functions as a decentralized distributed solution leveraging blockchain technology. 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 Busy Technology.

1.1 Key Findings

CyStack did not find any proves that indicates vulnerable usage and storage of users' wallet private keys in Busy Technology platform, nor any critical severity issues that would undermine the security of confidential transactions. CyStack identified only few of missing input validations. Key findings from the engagement included:

- Missing input validation for the variable *token* in the function *busyVoting:CreatePool*
- Missing input validation for the variable *token* in the function *busyVoting:CreateVote*

1.2 Limitations

Because of the quantity of static and dynamic analysis diagnostics, some findings were not fully analyzed during the assessment and some security vulnerabilities in third-party open source library dependencies might have not been discovered. Some effort was redirected to propose detailed remediation to the development team to ensure that the repairs would be made before the initial release of the product.

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 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, in order 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

Assessment	Details	Type
Initial targets		
Source Code Review	BusyAPI	Source code
Source Code Review	BusyChaincode	Source code
Source Code Review	BusyEvents	Source code
Source Code Review	BusyNetwork	Source code
Re-evaluated targets		
Source Code Review	BusyAPI	Source code
Source Code Review	BusyChaincode	Source code
Source Code Review	BusyEvents	Source code
Source Code Review	BusyNetwork	Source code

During the audit project, there were two minor changes in repository BusyAPI and BusyChaincode, respectively:

- Fixing get Transactions and Removing UpdateTransferFees (1);
- Removing Update Transfer Fee function (2).

These modifications were accepted by CyStack and were checked in the audit project.

Scope Exclusions

Any other repositories that not listed in the table Scope.

Client Allowances

Busy Technology provides guidance on BusyNetwork installation and public BusyAPI documentation.

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 adhered 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 to understand the root causes, as well as provides the developers 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 Busy Technology s.r.o. 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 application
- Identify the areas of scopes 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 Busy Technology s.r.o. 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 Busy Technology s.r.o. 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

On the basis of 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 come to a conclusion about the security of the source code. 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. The discovered critical vulnerabilities will be notified to customers 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.

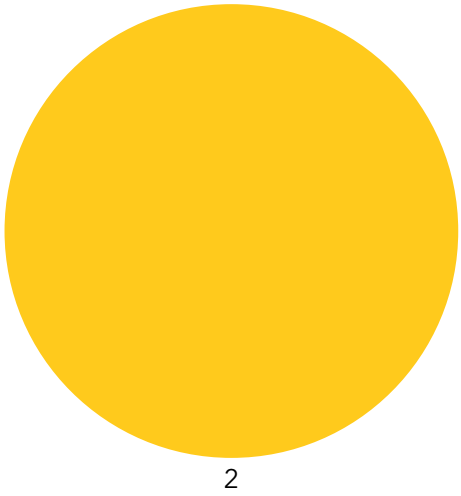
Phase 7: Reporting

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.

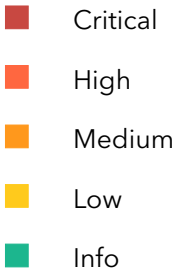
Dashboard

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

Vulnerabilities by severities



Legend



Vulnerabilities by assets

BusyChaincode	2

Vulnerabilities by CWE

Injection (CWE-929)	2

Table of vulnerabilities

ID	Status	Vulnerability	Severity
#busytech-003	Fixed	Missing input validation for the variable *token* in the function *busyVoting:CreatePool*	LOW
#busytech-004	Fixed	Missing input validation for the variable *token* in the function *busyVoting:CreateVote*	LOW

Recommendations

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

Key recommendations	
Issues	After the source code review for the four repositories BusyAPI, BusyChaincode, BusyEvents and BusyNetwork of the Busy Technology project, CyStack confirmed that Busy Technology is a well designed blockchain platform. No issues higher than Low have been found. Only two issues found related to missing input validation, but their impacts are all Low. These issues were all resolved by Busy Technology.
Recommendations	• Identify similar patterns of unsafe code according to the reported issues. • Evaluate the audit results with several different security audit third-parties for the most accurate conclusion.
References	• https://snyk.io/test/npm/fabric-client/1.4.20 • https://brightsec.com/blog/api-security/ • https://www.hyperledger.org/blog/2021/11/18/hyperledger-fabric-security-threats-what-to-look-for • https://arxiv.org/pdf/2109.03574.pdf

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 repository BusyAPI, BusyChaincode and BusyEvents with codeql and snyk. The results is shown in the following section Results.
2. Deployed BusyNetwork following the installation guidance in README.md.
3. Dynamically analyzed BusyAPI and BusyChaincode when running BusyNetwork with Burp Suite.
4. Manually reviewed the code of BusyAPI, BusyChaincode and BusyEvents with IDEs and debuggers. Built test case for each function in Busy Technology project. CyStack firstly focused on if any unsafe functions were used, then checked on the business logic of important mechanism such as signature verification, authorization, data validation, etc. The results is shown in the following section and found vulnerabilities will be reported in Vulnerability Details.

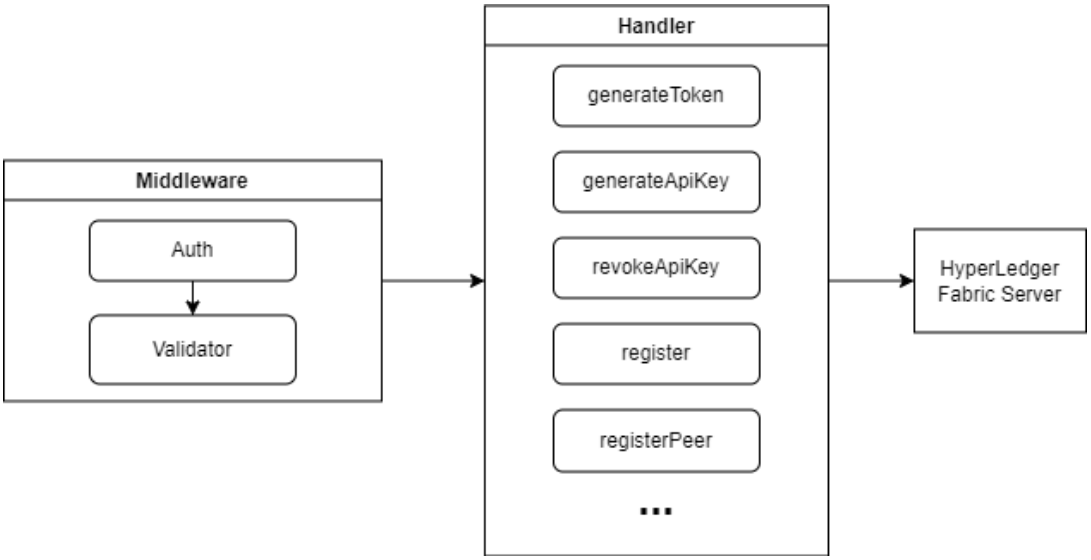
5.2 Results

5.2.1 Overview on Busy Technology system

Architecture and business flows

In this audit project, CyStack focuses on finding vulnerabilities in the four repositories under the Busy Technology project: BusyAPI, BusyChaincode, BusyEvents and BusyNetwork.

BusyAPI is written in Node.js and will be deployed as a docker container. It performs interactions between client applications and the BusyChaincode. The business flow of BusyAPI is illustrated in the following graph:



BusyChaincode is the core of Busy Technology. It is developed with Hyperledger Fabric framework, which is a platform for distributed ledger solutions, underpinned by a modular architecture delivering high degrees of confidentiality, resiliency, flexibility and scalability. BusyChaincode is written in Go. BusyChaincode contains 5 main programs, which could be understood as smart contracts in Hyperledger: Busy.go, BusyToken.go, BusyNFT.go, BusyMessenger.go and BusyVoting.go. Functions in these program are listed below:

1. Busy.go
    - Init
    - CreateUser
    - CreateStakingAddress
    - GetBalance
    - GetUser
    - GetTokenIssueFee
    - SetTokenIssueFee
    - IssueToken
    - Transfer
    - GetTotalSupply
    - Burn
    - MultibeneficiaryVestingV1
    - MultibeneficiaryVestingV2
    - GetLockedTokens
    - AttemptUnlock
    - UpdateTransferFee
    - GetTokenDetails
    - GetStakingInfo
    - Claim

- ClaimAll
- FetchStakingAddress
- Unstake
- GetCurrentPhase
- GetCurrentFee
- GetBusyAddress
- AuthenticateUser

2. BusyToken.go
    - Mint
    - MintBatch
    - BurnBatch
    - TransferFrom
    - BatchTransferFrom
    - IsApprovedForAll
    - SetApprovalForAll
    - BalanceOf
    - BalanceOfBatch
    - GetTokenInfo
    - UpdateTokenMetaData
3. BusyNFT.go
    - Mint
    - Transfer
    - GetCurrentOwner
    - UpdateNFTMetaData
4. BusyMessenger.go
    - SendMessage
    - UpdateMessagingFee
    - GetMessagingFee
5. BusyVoting.go
    - CreatePool
    - CreateVote
    - DestroyPool
    - QueryPool
    - PoolHistory
    - PoolConfig
    - UpdatePoolConfig

BusyEvents handles transaction logging and tracking for future explorer web application.

BusyNetwork is a repository, which contains shell code for starting the network of Busy Technology with peers and orderers.

Access control

No issues were found in BusyAPI access control. Roles and permissions are reported and and confirmed by Busy Technology s.r.o..

5.2.2 Static analysis

npm-audit and Snyk found 24 vulnerabilities, including 7 critical, 15 high and 2 medium in 677 scanned packages during the first security audit.

CyStack proceeded a seucurity retest on Busy Technology system and confirmed that 20 issues were resolved by Busy Technology team. Only 4 High issues remain due to the version of **protobufjs**. This problem could not be solved because the package **fabric-client** defined it as a fixed dependency.

5.2.3 Manual reviews

The project Busy Technology is developed with good coding and security practices. No issues with severity higher than Low have been found. These issues are mostly related to missing of input validation.

All of the issues are described in details in next section - Vulnerabilities Details.

Vulnerability Details

1. Missing input validation for the variable *token* in the function *busyVoting:CreatePool*

ID	#busytech-003
Category	CWE-929 - Injection
Description	In the function **CreatePool**, parameter **token** is not validated before it is passed to **burnCoins** function. This leads to incorrect behaviors in function **addTotalSupplyUTXO** in the function **burnCoins**. If the burnt tokens are not **BUSY**, **BUSY** tokens are still burnt by the function **ctx.GetStub().PutState()** called in **burnCoins**, but the total supply of **token** reduces instead. The function **burnCoins** works correctly only when the input value of **token** is the same as the value of **BUSY_COIN_SYMBOL**.
Severity	LOW
CVSS 3.0 base score	CVSS:3.0/AV:L/AC:H/PR:H/UI:N/S:C/C:N/I:L/A:L (3.9)
Target	BusyChaincode/busyVoting.go:98
Status	Fixed
Remediation	Check if the value of the variable **token** equals to the value of **BUSY_COIN_SYMBOL** before executing the function **burnCoins** in the function **CreatePool**. Else, use the hardcoded value **BUSY_COIN_SYMBOL** instead of using the parameter **token** in the function **CreatePool**.

Step to reproduce

The codelines where the issue occurs:

```
…
18  func (bv *BusyVoting) CreatePool(ctx contractapi.TransactionContextInterface, walletid
    ↪  string, PoolName string, PoolDescription string, token string) (*Response, error)
    ↪  {
…
…       err = burnCoins(ctx, defaultAddress, votingConfig.PoolFee, token)
…
154 }
155
```

From the line 439 to 446 in the function **burnCoins**, we can tell that the state is updated correctly only when the value of **token** equals to **BUSY_COIN_SYMBOL**:

```
…
431 func burnCoins(ctx contractapi.TransactionContextInterface, address string, coins
    ↪ string, token string) error {
432     minusOne, _ := new(big.Int).SetString("-1", 10)
433     bigTxFee, _ := new(big.Int).SetString(coins, 10)
434     err := addTotalSupplyUTXO(ctx, token, new(big.Int).Set(bigTxFee).Mul(minusOne,
        ↪ bigTxFee))
435     if err != nil {
436         return err
437     }
438
439     utxo := UTXO{
440         DocType: "utxo",
441         Address: address,
442         Amount:  bigTxFee.Mul(bigTxFee, minusOne).String(),
443         Token:   BUSY_COIN_SYMBOL,
444     }
445     utxoAsBytes, _ := json.Marshal(utxo)
446     err = ctx.GetStub().PutState(fmt.Sprintf("voting~%s~%s~%s",
        ↪ ctx.GetStub().GetTxID(), address, BUSY_COIN_SYMBOL), utxoAsBytes)
447     if err != nil {
448         return err
449     }
450     return nil
451 }
…
```

2. Missing input validation for the variable *token* in the function *busyVoting:CreateVote*

ID	#busytech-004
Category	CWE-929 - Injection
Description	In the function **CreateVote**, parameter **token** is not validated before it is passed to **burnCoins** function. This leads to incorrect behaviors in function **addTotalSupplyUTXO** in the function **burnCoins**. If the burnt tokens are not **BUSY**, **BUSY** tokens are still burnt by the function **ctx.GetStub().PutState()** called in **burnCoins**, but the total supply of **token** reduces instead. The function **burnCoins** works correctly only when the input value of **token** is the same as the value of **BUSY_COIN_SYMBOL**.
Severity	LOW
CVSS 3.0 base score	CVSS:3.0/AV:L/AC:H/PR:H/UI:N/S:C/C:N/I:L/A:L (3.9)
Target	BusyChaincode/busyVoting.go:245
Status	Fixed
Remediation	Check if the value of the variable **token** equals to the value of **BUSY_COIN_SYMBOL** before executing the function **burnCoins** in the function **CreateVote**. Else, use the hardcoded value **BUSY_COIN_SYMBOL** instead of using the parameter **token** in the function **CreateVote**.

Step to reproduce

The codelines where the issue occurs:

```
…
155  func (bv *BusyVoting) CreateVote(ctx contractapi.TransactionContextInterface, walletid
     ↪  string, votingaddress string, amount string, voteType string, token string)
     ↪  (*Response, error) {
…
245      err = burnCoins(ctx, defaultAddress, amount, token)
…
317  }
…
```

From the line 439 to 446 in the function **burnCoins**, we can tell that the state is updated correctly only when the value of **token** equals to **BUSY_COIN_SYMBOL**:

```
…
431 func burnCoins(ctx contractapi.TransactionContextInterface, address string, coins
    ↪ string, token string) error {
432     minusOne, _ := new(big.Int).SetString("-1", 10)
433     bigTxFee, _ := new(big.Int).SetString(coins, 10)
434     err := addTotalSupplyUTXO(ctx, token, new(big.Int).Set(bigTxFee).Mul(minusOne,
        ↪ bigTxFee))
435     if err != nil {
436         return err
437     }
438
439     utxo := UTXO{
440         DocType: "utxo",
441         Address: address,
442         Amount:  bigTxFee.Mul(bigTxFee, minusOne).String(),
443         Token:   BUSY_COIN_SYMBOL,
444     }
445     utxoAsBytes, _ := json.Marshal(utxo)
446     err = ctx.GetStub().PutState(fmt.Sprintf("voting~%s~%s~%s",
        ↪ ctx.GetStub().GetTxID(), address, BUSY_COIN_SYMBOL), utxoAsBytes)
447     if err != nil {
448         return err
449     }
450     return nil
451 }
…
```

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	Pass
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	Pass
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.