HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: ZeroSix
Date: 06 June, 2023

This report may contain confidential information about IT systems and the intellectual property of the Customer, as well as information about potential vulnerabilities and methods of their exploitation.

The report can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

Document

Name	Smart Contract Code Review and Security Analysis Report for ZeroSix
Approved By	Paul Fomichov | Lead Solidity SC Auditor at Hacken OU
Type	ERC1155 token; Certification
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://zerosix.co/
Changelog	01.05.2023 - Initial Review 24.05.2023 - Second Review 06.06.2023 - Third Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by ZeroSix (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

System Overview

ZeroSix is an extended and modified ERC-1155 implementation which is named ERC-1888. System explanation can be found in the following contracts:

- *Registry* – an extended ERC-1155 contract that can certify a token and assign an issuer to give the entire control of the related token id.
  A certificate is basically a token id (ERC-1155) that can be minted an infinite amount by only its issuer to any address. The issuer can not be changed later once it is issued. Every certificate has an expiration time and users cannot transfer or claim tokens once it is expired.
- *Issuer* – a management contract that helps requesting/approving workflows for issuing ERC-1888 certificates. Issuer contract is the owner of the Registry contract. ERC-1888 operations are controlled here.
- RegistryExtended – an extended version of Registry contract with multiple batch issuing and multiple batch transfers. It inherits the Registry contract.
- TokenAccount – a modified IERC1155Receiver contract that acts as an intermediary that receives ERC1155 tokens and forwards them to a specified wallet address.
- CommonConstants – a basic contract that is inherited by TokenAccount to store constant variables.

Privileged roles

- The owner of the *RegistryExtended contract can:*
  - issue an ERC-1155 token id to itself (msg.sender)
  - handle batch issues and batch multiple issues
- The owner of the Issuer contract can:
  - set a private issuer
  - approve requested certification requests
  - issue a certification without needing certification request
  - mint more volume to existing certificates

Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**.

- Functional requirements are provided.
- Technical description is provided.

Code quality

The total Code Quality score is **10** out of **10**.

- The project follows the official Solidity style guide.

Test coverage

Code coverage of the project is **100%** (branch coverage).

- Deployment and basic user interactions are covered with tests.

Security score

As a result of the audit, the code contains no issues. The security score is **10** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **10**. The system users should acknowledge all the risks summed up in the risks section of the report.

1
2
3
4
5
6
7
8
9
10
The final score

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
01 May 2023	8	1	0	0
24 May 2023	0	0	0	0
06 June 2023	0	0	0	0

Risks

No potential risks were found.

Checked Items

We have audited the Customers' smart contracts for commonly known and specific vulnerabilities. Here are some items considered:

Item	Description	Status	Related Issues
Default Visibility	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed	
Integer Overflow and Underflow	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed	
Outdated Compiler Version	It is recommended to use a recent version of the Solidity compiler.	Passed	
Floating Pragma	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed	
Unchecked Call Return Value	The return value of a message call should be checked.	Passed	
Access Control & Authorization	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed	
SELFDESTRUCT Instruction	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant	
Check-Effect-Interaction	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed	
Assert Violation	Properly functioning code should never reach a failing assert statement.	Passed	
Deprecated Solidity Functions	Deprecated built-in functions should never be used.	Passed	
Delegatecall to Untrusted Callee	Delegatecalls should only be allowed to trusted addresses.	Not Relevant	
DoS (Denial of Service)	Execution of the code should never be blocked by a specific contract state unless required.	Passed	

Race Conditions	Race Conditions and Transactions Order Dependency should not be possible.	Passed	
Authorization through tx.origin	tx.origin should not be used for authorization.	Passed	
Block values as a proxy for time	Block numbers should not be used for time calculations.	Not Relevant	
Signature Unique Id	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery. EIP-712 should be followed during a signer verification.	Not Relevant	
Shadowing State Variable	State variables should not be shadowed.	Passed	
Weak Sources of Randomness	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant	
Incorrect Inheritance Order	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed	
Calls Only to Trusted Addresses	All external calls should be performed only to trusted addresses.	Passed	
Presence of Unused Variables	The code should not contain unused variables if this is not justified by design.	Passed	
EIP Standards Violation	EIP standards should not be violated.	Passed	
Assets Integrity	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed	
User Balances Manipulation	Contract owners or any other third party should not be able to access funds belonging to users.	Passed	
Data Consistency	Smart contract data should be consistent all over the data flow.	Passed	

Flashloan Attack	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Not Relevant	
Token Supply Manipulation	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the Customer.	Passed	
Gas Limit and Loops	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Passed	
Style Guide Violation	Style guides and best practices should be followed.	Passed	
Requirements Compliance	The code should be compliant with the requirements provided by the Customer.	Passed	
Environment Consistency	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed	
Secure Oracles Usage	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant	
Tests Coverage	The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed	
Stable Imports	The code should not reference draft contracts, which may be changed in the future.	Passed	

Findings

■■■■ Critical

No critical severity issues were found.

■■■ High

No high severity issues were found.

■■ Medium

M01. Uninitialized Implementation

Impact	Medium
Likelihood	Medium

It is not recommended to leave an implementation contract uninitialized. An uninitialized implementation contract can be taken over by an attacker.

Path: ./packages/traceability/issuer/contracts/Issuer.sol

Recommendation: Invoke the *_disableInitializers()* function in the constructor to automatically lock the contract when it is deployed.

Found in: 4b407ff68

Status: Fixed (Revised commit: e55d2e832c22878d0171a58c5c3ff1846c5e3bbf)

■ Low

L01. Outdated Solidity Version

Impact	Low
Likelihood	Medium

Using an outdated compiler version can be problematic, especially if publicly disclosed bugs and issues affect the current compiler version. The contracts in the project have the Solidity version 0.8.4.

Paths: ./packages/traceability/issuer/contracts/Issuer.sol

./packages/traceability/issuer/contracts/Registry.sol

./packages/traceability/issuer/contracts/RegistryExtended.sol

./packages/traceability/issuer/contracts/ERC1888/IERC1888.sol

./packages/trade/exchange-token-account/contracts/TokenAccount.sol

./packages/trade/exchange-token-account/contracts/Common.sol

Recommendation: Use a contemporary and the same compiler version for all contracts.

Found in: 4b407ff68

Status: Fixed (Revised commit: e55d2e832c22878d0171a58c5c3ff1846c5e3bbf)

L02. Floating Pragma

Impact	Low
Likelihood	Medium

Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Paths:

./packages/trade/exchange-token-account/contracts/TokenAccount.sol

./packages/trade/exchange-token-account/contracts/Common.sol

Recommendation: Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment.

Found in: 4b407ff68

Status: Fixed (Revised commit: e55d2e832c22878d0171a58c5c3ff1846c5e3bbf)

L03. Missing Validation

Impact	Medium
Likelihood	Medium

The length of the inputs, _data and _owners are not checked against the possible mistake of entering different lengths by accident.

This can lead to creating request with zero address or empty data.

Path:./packages/traceability/issuer/contracts/Issuer.sol: requestCertificationForBatch()

Recommendation: Implement checks to validate that both 2 input arrays are in the same size and addresses are not empty.

Found in: 4b407ff68

Status: Fixed (Revised commit: e55d2e832c22878d0171a58c5c3ff1846c5e3bbf)

L04. Missing Validation

Impact	Low

Likelihood	Medium

The length of the _expirationDates is not checked against the possible mistake of entering different lengths by accident.

This can lead to creating request with zero address or empty data.

Path: ./packages/traceability/issuer/contracts/Registry.sol: batchIssue()

Recommendation: Implement check to validate that the length of the expiration dates is same with the others.

Found in: 4b407ff68

Status: Fixed (Revised commit: e55d2e832c22878d0171a58c5c3ff1846c5e3bbf)

L05. Redundant Implementation

Impact	Low
Likelihood	Medium

In *safeTransferAndClaimFrom* function, the given token(s) is first transferred from ‘_from’ to ‘_to’ address and then it is burned.

There is no need to transfer them if they are going to be burned after.

Redundant implementations make the code look more sophisticated and hard to understand.

Path: ./packages/traceability/issuer/contracts/Registry.sol: safeTransferAndClaimFrom()

Recommendation: Instead of sending the tokens to address ‘_to’, burn them from the address ‘_from’ directly.

Found in: 4b407ff68

Status: Mitigated (Revised commit: e55d2e832c22878d0171a58c5c3ff1846c5e3bbf) (Customer stated that the reason of this implementation is to transfer tokens from the platform wallet to custodial wallet of the user firstly and then burning them to achieve the traceability of retirement or claiming functionality on the platform.)

L06. Reading Array Length in a Loop

Impact	Low
Likelihood	Medium

Array length should be saved in a local variable instead of being computed in each loop cycle during the condition check.

Path:./packages/traceability/issuer/contracts/Issuer.sol: requestCertificationForBatch()

Recommendation: Save the array length in a variable and use that variable in the for loop condition.

Found in: 4b407ff68

Status: Fixed (Revised commit: e55d2e832c22878d0171a58c5c3ff1846c5e3bbf)

L07. Functions That Can Be Declared External

Impact	Low
Likelihood	Medium

“public” functions that are never called by the contract should be declared “external” to save Gas.

Notice: it is also applicable to the “initialize” function in upgradable contracts. There is no magic in declaring them public if the contract is not inherited.

Path: ./packages/traceability/issuer/contracts/Issuer.sol: initialize(), setPrivateIssuer(), getCertificationRequest(), issue(), issueBatch()

Recommendation: Change functions' visibilities to external.

Found in: 4b407ff68

Status: Fixed (Revised commit: e55d2e832c22878d0171a58c5c3ff1846c5e3bbf)

L08. Missing Zero Address Validation

Impact	Low
Likelihood	Medium

Address parameters are being used without checking against the possibility of 0x0.

This can lead to unwanted external calls to 0x0.

Path: ./packages/trade/exchange-token-account/contracts/TokenAccount.sol: constructor()

Recommendation: Implement zero address checks.

Found in: 4b407ff68

Status: Fixed (Revised commit: e55d2e832c22878d0171a58c5c3ff1846c5e3bbf)

Informational

I01. Redundant Function Declaration

requestCertification function is redundant since there is already another function *requestCertificationFor* to do the same operation. Users can set their own addresses(as msg.sender) as input for *requestCertificationFor* function and there is no need for a second function for this basic feature.

Similar redundant implementation is found in *issue* and *issueBatch* functions. These functions are basically calling the *requestCertificationFor* and *approveCertificationRequest* functions with order. This can be controlled and applied manually from the web side by calling *requestCertificationFor* and *approveCertificationRequest functions*.

Path: ./packages/traceability/issuer/contracts/Issuer.sol: requestCertification(), requestCertificationFor(), approveCertificationRequest()

Recommendation: Remove the redundant function *requestCertification*.

Found in: 4b407ff68

Status: Fixed (Revised commit: e55d2e832c22878d0171a58c5c3ff1846c5e3bbf)

I02. Misleading Contract Name

ERC1888 interface is not named as *IERC1888* although it is not a contract but an interface.

Contract name should represent the contract logic and should not mislead it.

Path: ./packages/traceability/issuer/contracts/ERC1888/IERC1888.sol

Recommendation: Change contract name to fit the logic.

Found in: 4b407ff68

Status: Fixed (Revised commit: e55d2e832c22878d0171a58c5c3ff1846c5e3bbf)

I03. Style Guide Violation

Some parts of the code violate the style guide standards.

The provided projects should follow the official guidelines.

Especially pay attention to ‘Maximum Line Length’.

Path: ./packages/traceability/issuer/contracts/ERC1888/IERC1888.sol

Recommendation: Follow the official Solidity guideline. https://docs.soliditylang.org/en/v0.8.13/style-guide.html

Found in: 4b407ff68

Status: Fixed (Revised commit: e55d2e832c22878d0171a58c5c3ff1846c5e3bbf)

I04. Redundant Code Block

Variables with public visibility do not need a getter function.

This increases Gas usage on deployment as the compiled bytecode will be bigger.

Path: ./packages/traceability/issuer/contracts/Issuer.sol: getRegistryAddress(), getPrivateIssuesAddress()

Recommendation: Remove redundant code.

Found in: 4b407ff68

Status: Fixed (Revised commit: e55d2e832c22878d0171a58c5c3ff1846c5e3bbf)

I05. Variables That Can Be Declared Immutable

Impact	Low
Likelihood	Medium

wallet address can be declared immutable because it is never changed after being declared.

Redundant/mistaken declarations cause unnecessary Gas consumption.

Path: ./packages/trade/exchange-token-account/contracts/TokenAccount.sol

Recommendation: Declare the variable as immutable.

Found in: 4b407ff68

Status: Fixed (Revised commit: e55d2e832c22878d0171a58c5c3ff1846c5e3bbf)

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Appendix 1. Severity Definitions

When auditing smart contracts Hacken is using a risk-based approach that considers the potential impact of any vulnerabilities and the likelihood of them being exploited. The matrix of impact and likelihood is a commonly used tool in risk management to help assess and prioritize risks.

The impact of a vulnerability refers to the potential harm that could result if it were to be exploited. For smart contracts, this could include the loss of funds or assets, unauthorized access or control, or reputational damage.

The likelihood of a vulnerability being exploited is determined by considering the likelihood of an attack occurring, the level of skill or resources required to exploit the vulnerability, and the presence of any mitigating controls that could reduce the likelihood of exploitation.

Risk Level	High Impact	Medium Impact	Low Impact
High Likelihood	Critical	High	Medium
Medium Likelihood	High	Medium	Low
Low Likelihood	Medium	Low	Low

Risk Levels

Critical: Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.

High: High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.

Medium: Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.

Low: Major deviations from best practices or major Gas inefficiency. These issues won't have a significant impact on code execution, don’t affect security score but can affect code quality score.

Impact Levels

High Impact: Risks that have a high impact are associated with financial losses, reputational damage, or major alterations to contract state. High impact issues typically involve invalid calculations, denial of service, token supply manipulation, and data consistency, but are not limited to those categories.

Medium Impact: Risks that have a medium impact could result in financial losses, reputational damage, or minor contract state manipulation. These risks can also be associated with undocumented behavior or violations of requirements.

Low Impact: Risks that have a low impact cannot lead to financial losses or state manipulation. These risks are typically related to unscalable functionality, contradictions, inconsistent data, or major violations of best practices.

Likelihood Levels

High Likelihood: Risks that have a high likelihood are those that are expected to occur frequently or are very likely to occur. These risks could be the result of known vulnerabilities or weaknesses in the contract, or could be the result of external factors such as attacks or exploits targeting similar contracts.

Medium Likelihood: Risks that have a medium likelihood are those that are possible but not as likely to occur as those in the high likelihood category. These risks could be the result of less severe vulnerabilities or weaknesses in the contract, or could be the result of less targeted attacks or exploits.

Low Likelihood: Risks that have a low likelihood are those that are unlikely to occur, but still possible. These risks could be the result of very specific or complex vulnerabilities or weaknesses in the contract, or could be the result of highly targeted attacks or exploits.

Informational

Informational issues are mostly connected to violations of best practices, typos in code, violations of code style, and dead or redundant code.

Informational issues are not affecting the score, but addressing them will be beneficial for the project.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Initial review scope

Repository	https://github.com/482solutions/Zero6
Commit	4b407ff
Whitepaper	-
Requirements	-
Technical Requirements	Link
Contracts	File: packages/traceability/issuer/contracts/Issuer.sol SHA3: 4220f3a122bd89bd20eb2f3333ff54b99fbba0f5cce59daa57fd398e0d571f3a File: packages/traceability/issuer/contracts/Registry.sol SHA3: 893ef0650381caf6f0ded5cb9e5f1c723240bc29d79e1f4eb75c50ec2f32fc87 File: packages/traceability/issuer/contracts/RegistryExtended.sol SHA3: cfb3b07c272fae16d90046fdb2f40565e29748e9ae784f394f642f796ab0449d File: packages/traceability/issuer/contracts/ERC1888/IERC1888.sol SHA3: 5194440eeccaba0c5a3169bea6f7387d75d2e21293427da7563b69211189d790 File: packages/trade/exchange-token-account/contracts/Common.sol SHA3: 58fd2d8be9214f939a2f3428892a8843536c5934b9c0da805e446871407af7e3 File: packages/trade/exchange-token-account/contracts/TokenAccount.sol SHA3: 1731e32b3d7dadd6d596a25be13452a977e560c235ec201c43d3f122bddd194b

Second review scope

Repository	https://github.com/482solutions/Zero6
Commit	e55d2e832c22878d0171a58c5c3ff1846c5e3bbf
Whitepaper	-
Requirements	ZeroSix-SC-Audit-Functional-Requirements
Technical Requirements	Link
Contracts	File: ./packages/traceability/issuer/contracts/Issuer.sol SHA3: 73d4cc97b5050fd773247ced2d22902b75b37df4e1de364096cec734c29d18a2 File: ./packages/traceability/issuer/contracts/Registry.sol SHA3: 6d7c91e982566caf141bc7274debd879dab87c08c190e40d023756c47565f4cf File: ./packages/traceability/issuer/contracts/RegistryExtended.sol

	SHA3: ac37f4ae72b8178c69e6891229236391519 33c87207e167a0d65ec19a9eb3c7b File: ./packages/traceability/issuer/contracts/ERC1888/IERC1888.sol SHA3: 2d4c7d9fcaa309c854a84ab05ce28b5e00cfaa7914c13545c549569d67cef73a File: ./packages/trade/exchange-token-account/contracts/Common.sol SHA3: d0733f901a32aa9c48aba94669cbdf9c8624c0cd1866f7a5c2f32fa6f4bfe995 File: ./packages/trade/exchange-token-account/contracts/TokenAccount.sol SHA3: 506f65a899ee7849b4f50f77e3a5d045a78c34a6fba619430e7e511275e60fdb

Third review scope

Repository	https://github.com/482solutions/Zero6
Commit	da0ae264338f188276e9f6c385667676b00a3e1e
Whitepaper	-
Requirements	ZeroSix-SC-Audit-Functional-Requirements
Technical Requirements	Link
Contracts	File: ./packages/traceability/issuer/contracts/Issuer.sol SHA3: 1139c75c7260ea66b8dadf1883b652b48ccdb7e0d039e7df8004c660e65c654b File: ./packages/traceability/issuer/contracts/Registry.sol SHA3: 226d5aa2e0deeaeb0b78fe193fc2bab22ebb2fcb4c038ba945639874907737ee File: ./packages/traceability/issuer/contracts/RegistryExtended.sol SHA3: fc4343d875d4a8a3134f9d20c8fd5c95b3f0f629f094a2e69b21778851974a75 File: ./packages/traceability/issuer/contracts/ERC1888/IERC1888.sol SHA3: e1576ba53f5b3a10741a49b18d6e3d0c981b06ce0b57ef7f8d76101a6011b624 File: ./packages/trade/exchange-token-account/contracts/Common.sol SHA3: d0733f901a32aa9c48aba94669cbdf9c8624c0cd1866f7a5c2f32fa6f4bfe995 File: ./packages/trade/exchange-token-account/contracts/TokenAccount.sol SHA3: 2e7c6aff1d3ab7f1e6b383fc320956155c9a1c088d05f9521c549a9a08caa073