HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: ITHEUM
Date: August 10th, 2022

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for Itheum Limited
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU
Type	Claims portal, vesting, rewards, airdrops
Platform	Elrond
Network	Elrond
Language	Rust
Methods	Manual Review, Automated Review, Architecture review
Website	https://www.itheum.io/
Timeline	18.07.2022 - 08.08.2022
Changelog	21.07.2022 - Initial Review 10.08.2022 - Second Review

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Introduction 4

Scope 4

Severity Definitions 6

Executive Summary 7

Checked Items 8

System Overview 11

Findings 12

Disclaimers 16

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by Itheum Limited (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.

Scope

The scope of the project is smart contracts in the repository:

Initial review scope

Repository:
https://github.com/Itheum/itheumcore-elrond-sc-claims
Commit:
6ebbe1f271753b7a61fb0625c10648c23ccebb8a
Technical Documentation:
Type: Technical description
https://itheum.notion.site/Elrond-Claims-Contract-b4d52fc1f3f7452f864e847601650f6c
Type: Functional requirements
https://itheum.notion.site/Elrond-Claims-Contract-b4d52fc1f3f7452f864e847601650f6c
Integration and Unit Tests: Yes
Contracts:
File: ./src/events.rs
SHA3: cbc915b3c348b2098137347208b415b623738bb2ff741a4f900675f444e0446a
File: ./src/lib.rs
SHA3: a2b58563c57c575485ff9f9a299376c9678ef6e40b7c98e6253903e79541c9b1
File: ./src/views.rs
SHA3: e1b4904739fdfbfb108e7db512f785748bf020610aeb3ca16e363dc1f2c2ee4c
File: ./src/storage.rs
SHA3: 069edf1c04cbad58bfc17e5f4d9b8ed7e0144756de8f42f80c2df754437ca591

Second review scope

Repository:
https://github.com/Itheum/itheumcore-elrond-sc-claims
Commit:
086b7e4c7329db725358a0b8c45ee73d7dcb5f8a
Technical Documentation:
Type: Technical description
https://itheum.notion.site/Elrond-Claims-Contract-b4d52fc1f3f7452f864e847601650f6c
Type: Functional requirements
https://itheum.notion.site/Elrond-Claims-Contract-b4d52fc1f3f7452f864e847601650f6c
Integration and Unit Tests: Yes
Contracts:
File: ./src/events.rs
SHA3: 95633b5b400026c8c08749a839a441dc4e7d83426a1a1a0cb733c6319df0b9e3
File: ./src/lib.rs
SHA3: fea97bb7944cb98fe047d6ee56aef3d9436c98fbc90aabec4ae8a0e929650bb6
File: ./src/views.rs
SHA3: d0bdc78e30401790f78e9049ef74e3e63be051b26a2e440ff731dd72d64bbe11
File: ./src/storage.rs
SHA3: 0f52e6d07b476e30f2a93e974b7753f9bb73b53ddb79fee01b61bd1c92f7b271

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

```
File: ./src/requirements.rs
SHA3: 8c2917415b0e6964856662e9fe9377839a7db8a058f033c8d2c955de437faa25
```

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions.
Medium	Medium-level vulnerabilities are important to fix; however, they cannot lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Executive Summary

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

Documentation quality

The Customer provided superficial and internal functional and technical requirements. The total Documentation Quality score is **10** out of **10**.

Code quality

The total CodeQuality score is **10** out of **10**.

Architecture quality

The architecture quality score is **10** out of **10**.

Security score

As a result of the second audit, the code contains only **1** low severity issue. 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: **10.0**.

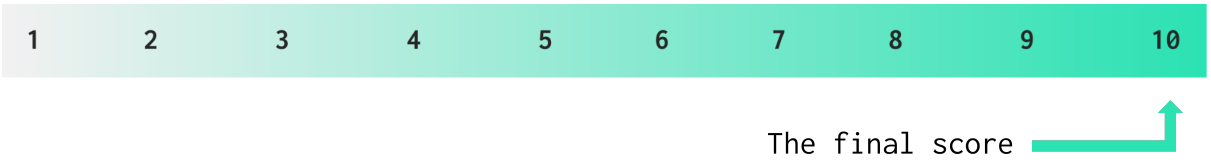

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
21 July 2022	6	1	0	0
08 August 2022	1	0	0	0

Checked Items

We have audited provided smart contracts for commonly known and more specific vulnerabilities. Here are some of the items that are considered:

Item	Description	Status
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 Rust compiler.	Passed
Floating Pragma	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Not Relevant
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 Rust 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 it is 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.	Passed
Signature Unique Id	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifier should always be used. All parameters from the signature should be used in signer recovery	Passed
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.	Not Relevant
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.	Not Relevant
Assets integrity	Funds are protected and cannot be withdrawn without proper permissions.	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 100%, 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, that may be changed in the future.	Passed

System Overview

Itheum is the world's 1st decentralized, cross-chain data brokerage platform. Itheum has the following contracts:

- *ClaimsContract* – A simple “upgradeable contract” that holds a mapping from addresses and a “claim type” to a “claim amount” and a “claim add date”. A “claim type” is an u32 taking values between 0 and 2. The “claim amount” is a BigUint, which represents the amount of Itheum they can take out. The “claim add date” is a timestamp on when the claim for the address and “claim type” was last modified in the smart contract.

Privileged roles

- *The Owner* of the smart contract - can manually put in a new “claim amount” for an address and a “claim type”.
- *Itheum Token Owner* - the owner of the Itheum token on Elrond.
- *Itheum Token* is a ESDT token on Elrond.
- *DEX DApp* - Itheum DEX to interact with this contract using its own wallet.

Risks

- In case the contract owner keys leak, an attacker can get access to all funds that belong to *Claims* contract and will be able to send them to any address.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Findings

Critical

No critical severity issues were found.

High

No critical severity issues were found.

Medium

1. **Requirements incompliance.**

   The documentation states that the contract should implement the following feature: ‘Owner of Claims Contract and Owners of Token should be different’. This feature is not implemented.

   File: ./src/lib.rs

   Contract: ClaimsContract

   Function: set_reward_token

   Recommendation: Either implement the missing logic or remove the corresponding statement from the documentation.

   Status: Fixed (Revised commit: 086b7e4c7329db725358a0b8c45ee73d7dcb5f8a)

Low

1. **Zero valued transactions.**

   Happens when a function tries to send zero funds to a target. Remove_claim function does not contain zero value validation for ‘amount’ parameter.

   The function remove_claim can execute a zero-valued transaction if ‘amount’ is zero.

   This can lead to a transaction with zero value to be sent. Missing a zero value check can lead to unnecessary storage updates and emitting events because nothing was changed.

   File: ./src/lib.rs

   Contract: ClaimsContract

   Function: remove_claim

   Recommendation: Add zero validation for ‘amount’ parameter inside ‘remove_claim’ function.

   Status: Fixed (Revised commit: 086b7e4c7329db725358a0b8c45ee73d7dcb5f8a)

2. Missing parameter zero value check.

Remove_claims function does not contain zero value validation for the ‘tuple.2’ value inside the loop.

Missing a zero value check can lead to unnecessary storage updates and emitting events because nothing was changed.

File: ./src/lib.rs

Contract: ClaimsContract

Function: remove_claims

Recommendation: Add zero validation for ‘tuple.2’ inside ‘remove_claims’ function.

Status: Fixed (Revised commit: 086b7e4c7329db725358a0b8c45ee73d7dcb5f8a)

3. Duplicate code.

Duplication of code may lead to unnecessary Gas consumption. There are ‘require’ statement duplications in add_claim, add_claims, remove_claim, remove_claims, harvest_claims .

Code duplication can lead to big size wasm code uploaded to the network and unnecessary Gas consumption.

File: ./src/lib.rs

Contract: ClaimsContract

Functions: dd_claim, add_claims, remove_claim, remove_claims, harvest_claims

Recommendation: Extract require statement with the error message: ‘Reward token is not set’ to a separate method.

Status: Fixed (Revised commit: 086b7e4c7329db725358a0b8c45ee73d7dcb5f8a)

4. Unnecessary reading from storage.

When Claim Contract is paused, there is no reason to read data about reward token value from storage.

This can lead to more Gas consumption when the contract is paused.

File: ./src/lib.rs

Contract: ClaimsContract

Function: harvest_claim

Recommendation: Move require statement with message ‘Contract is paused’ before reading the reward token value from storage.

Status: Fixed (Revised commit: 086b7e4c7329db725358a0b8c45ee73d7dcb5f8a)

5. **Misleading method name.**

   Method name should represent the method logic and should not mislead it.

   'Pause' method implements logic that contradicts its name.

   This makes code harder to read.

   File: ./src/lib.rs

   Contract: ClaimsContract

   Function: pause

   Recommendation: Change method name to fit the logic or split pause and unpause functionality to 2 different methods.

   Status: Fixed (Revised commit: 086b7e4c7329db725358a0b8c45ee73d7dcb5f8a)

6. **Missing event emitting.**

   Events for critical state changes should be emitted for tracking things off-chain. 'Harvest_claim' method does not emit events for collected rewards type in case if 'claim_type' parameter was None.

   This can lead to a lack of event data on the main network for analytics.

   File: ./src/lib.rs

   Contract: ClaimsContract

   Function: harvest_claim

   Recommendation: Add information to event 'all_claims_collected_event' about rewards for any type or emit 3 different events for each reward type when 'claim_type' parameter has None value.

   Status: Fixed (Revised commit: 086b7e4c7329db725358a0b8c45ee73d7dcb5f8a)

7. **"*#![feature]*" attribute may not be used on the stable release channel.**

   Feature attributes are only allowed on the nightly release channel. Stable or beta compilers will not comply. This can lead to a lack of event data on the main network for analytics.

   The file lib.rs contains *#![feature]* attribute, if needed the feature, make sure to use a nightly release of the compiler (but be warned that the feature may be removed or altered in the future).

   File: ./src/lib.rs

Recommendation: Consider removing #![feature(proc_macro_quote)] and quote import from lib.rs.

Status: New

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed by the best industry practices at the date 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 to 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, Consultant cannot guarantee the explicit security of the audited smart contracts.