HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: IMPT.io
Date: Oct 17th, 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 IMPT.io
Approved By	Noah Jelich | Lead Solidity SC Auditor at Hacken OU
Type	ERC20 token
Platform	EVM
Network	Ethereum
Language	Solidity
Methods	Manual Review, Automated Review, Architecture Review
Website	https://impt.io
Timeline	12.09.2022 - 17.10.2022
Changelog	14.09.2022 - Initial Review 17.10.2022 - Second Review

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

Table of contents

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

Introduction

Hacken OÜ (Consultant) was contracted by IMPT.io (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/pixelplex/impt-smartcontracts
Commit:
75b5c9f6b8cd280f58a71068e1450d0bba97b405
Documentation:
Whitepaper (partial functional requirements provided)

Functional requirements: Document provided

Integration and Unit Tests: Yes
Contracts:
File: ./contracts/access/Operator.sol
SHA3: de3f0a9d15a74b060a08918b0801a1697e51553ff7932f32de3521ef31d9c515

File: ./contracts/access/TwoStageOwnable.sol
SHA3: c463151e935b9ab2aea53a631935121afbe0a4d9ad68ea64a78a5c5ce180d646

File: ./contracts/IMPT.sol
SHA3: 72c66a38d99a49c07e4d773bb6da6a7210de418b784c08f532b2bae58b867732

File: ./contracts/mocks/MockedVesting.sol
SHA3: 5c0273ec45e715002baee44a0367ebc8b9a39a5ea4b1c37f138ca22d79b13036

File: ./contracts/mocks/TwoStageOwnableMock.sol
SHA3: a565c964bb7b58e531281028f795685ca44fdc925d516ea98acc60f489cfa00d

File: ./contracts/Vesting.sol
SHA3: ffd047f70a810325f7c00a95a5b8b4c7f71634fc4ebc47315cca1ecd0e12ad20

Second review scope

Repository:
https://github.com/anonymous-001-1/IMPT
Commit:
39d3c96afaefe7f8ea6dd898dd9d281b21433ec6
Documentation:
Whitepaper (partial functional requirements provided)

Functional requirements: Document provided

Integration and Unit Tests: Yes
Contracts:
File: ./contracts/IMPT.sol
SHA3: 126f8e8d0c78a4beadb89ab45e63eaf041aec1b083fb085f567c48d5067aa7d6

File: ./contracts/TwoStageOwnable.sol

SHA3: f117929396c8a8ef59254e7231def6a5a7af9b0faa1c93e0cd76f3480e9010ce

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 scoring methodology.

Documentation quality

The total Documentation Quality score is **5** out of **10**.

- Technical requirements are not provided.
- Functional requirements are provided.

Code quality

The total Code Quality score is **8** out of **10**.

- The code follows best practices.
- The code partially follows official language style guides.
- Development environment is configured.

Test coverage

Test coverage of the project is **93.1%**.

- Deployment and basic user interactions are covered with tests.
- Interactions by several users are tested thoroughly.

Security score

As a result of the audit, the code contains **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: **8.9**.

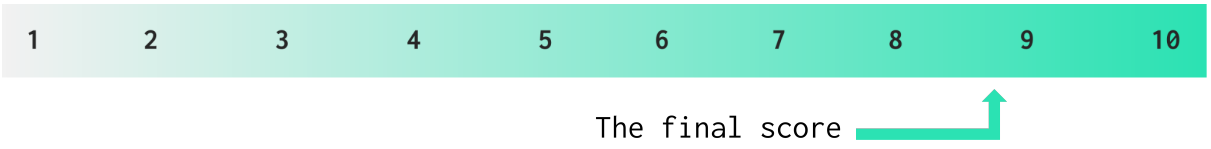

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
14 September 2022	6	2	2	0
14 October 2022	1	0	0	0

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

Checked Items

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

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Not Relevant
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Passed
Access Control & Authorization	CWE-284	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	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Passed
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Not Relevant
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless required.	Passed
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed

Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Not Relevant
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Passed
Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155	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	Not Relevant
Shadowing State Variable	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant
Incorrect Inheritance Order	SWC-125	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	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Passed
Presence of unused variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Passed
EIP standards violation	EIP	EIP standards should not be violated.	Passed
Assets integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions.	Passed
User Balances manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	**Custom**	Smart contract data should be consistent all over the data flow.	Passed
Flashloan Attack	**Custom**	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	**Custom**	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	**Custom**	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	**Custom**	Style guides and best practices should be followed.	Failed
Requirements Compliance	**Custom**	The code should be compliant with the requirements provided by the Customer.	Passed
Environment Consistency	**Custom**	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	**Custom**	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	**Custom**	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	**Custom**	The code should not reference draft contracts, that may be changed in the future.	Passed

System Overview

IMPT.io is a project that includes IMPT tokens and Vesting contracts:

- *IMPT* – simple ERC-20 token that mints initial supply to the given list of addresses during deployment. Additional minting is not allowed.
  It has the following attributes:
  - Name: IMPT
  - Symbol: IMPT.io
  - Decimals: 18
  - Total supply: 3b tokens.
- TwoStageOwnable - an abstract contract for adding and additional process of owner changing. The additional process is that the owner creates an order to assign a new owner, and the new owner can accept or not respond to that order.

Privileged roles

- The owner of the IMPT.io can:
  - pause and unpause the contract

Risks

- The deployer can provide two lists that contain addresses and amounts. In the constructor of IMPT, it sends specified amounts to addresses. As a result, the deployer can change the total supply of IMPT tokens during deployment.
- Vesting and Operator contracts are not in the scope of the audits after the first one, so the secureness of those contracts cannot be guaranteed.

Findings

Critical

No critical severity issues were found.

High

1. Potential Out-of-Gas Exception

In the constructor of the IMPT, the deployer provides two lists to mint the initial supply to the specific addresses. However, there is no check for the length of the provided lists.

This issue may cause the deployment failure with a run-out of Gas exception.

Paths: ./contract/IMPT.sol: constructor()

Recommendation: Implement a length check for the recipients_ and amounts_ parameters of IMPT constructor to avoid “run out of Gas” exception.

Status: Fixed (39d3c96afaefe7f8ea6dd898dd9d281b21433ec6e)

Medium

No medium severity issues were found.

Low

1. Variable Shadowing

In the constructor of the IMPT *decimals* variable shadows ERC20’s decimals() function.

Path: ./contracts/IMPT.sol : constructor()

Recommendation: Rename related variables/arguments.

Status: Fixed (39d3c96afaefe7f8ea6dd898dd9d281b21433ec6e)

2. Functions that Can Be Declared External

In order to save Gas, public functions that are never called in the contract should be declared external.

Paths: ./contracts/TwoStageOwnable.sol: nominatedOwner()

Recommendation: Use the external attribute for functions never called from the contract.

Status: Fixed (39d3c96afaefe7f8ea6dd898dd9d281b21433ec6e)

3. Missing Zero Address Validation

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

This can lead to unwanted external calls to 0x0.

Paths: ./contracts/TwoStageOwnable.sol: _nominateNewOwner()

Recommendation: Implement zero address checks.

Status: Fixed (39d3c96afaefe7f8ea6dd898dd9d281b21433ec6e)

4. **Style Guides Violation**

   The provided projects should follow the official guidelines.

   Paths: ./contracts/TwoStageOwnable.sol

   Recommendation: Follow the official Solidity guidelines.

   Status: Reported

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.