HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Infinityness
Date: May 23th, 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 Infinityness
Approved By	Evgeniy Bezuglyi | SC Department Head at Hacken OU
Type	Staking; Vesting
Platform	Hedera
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Website	-
Timeline	13.05.2022 - 23.05.2022
Changelog	18.05.2022 - Initial Review 23.05.2022 - 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 Infinityness (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/buidler-labs/headstarter-contracts
Commit:
7e4fd68
Technical Documentation:
Type: Business Logic and Technical Documentation
Link: https://github.com/buidler-labs/headstarter-contracts#readme
JS tests: Yes
Deployed Contracts Addresses: No
Contracts:

File: ./contracts/BasicPool.sol
SHA3: 3192dc87db624e4141159ab83f1cc0cd8d1396c47feea9ed5effe77e14063217

File: ./contracts/extensions/VestingPool.sol
SHA3: fa400bbcf5e3daefeff680397d84ec96f2d882499c56c375e3dc78d5cae47f50

File: ./contracts/extensions/AllowlistPool.sol
SHA3: cbd59685b19376637c0ae926ae564cced979550c88047a8249a6fa7f1616992c

File: ./contracts/IDOPoolWithVesting.sol
SHA3: 777b34d870c91a5e7f5dd52e20515374f0abc41cb798d0c44865f6ff177712b8

File: ./contracts/mocks/PoolWithVesting.sol
SHA3: cf7b73207d0be3d80ccd55693e67fa819dca16a101fbafef78097394755e4138

File: ./contracts/extensions/MultiPartyWithdrawPool.sol
SHA3: c1cbc703e9a9f26fcda98684bacf0e44310fc6964f9e1768c1469312d7213c47

File: ./contracts/extensions/MerklePool.sol
SHA3: 4f70bc742e95c8a9eecedf6c7e47f3cba5d814b3cfb24113e63efa4bebb4fa21

File: ./contracts/mocks/PoolMultiPartyWithdraw.sol
SHA3: 61e9fd5bb7630d25b6971428d9d6589d0f04fba6855c3a9e008eb961e91b1f89

File: ./contracts/IDOPool.sol
SHA3: b4756ec7ff7be74319be5ff0226164d93f4bd9bcd29d2ca786025febf1a1dd4b

File: ./contracts/mocks/PoolAllowlist.sol
SHA3: 47bf7e2fb744f34ea4bbf23b03531c05bb707290bd54ea2041f10ef430b4f62f

File: ./contracts/interfaces/IPool.sol
SHA3: 6f631d45860f4cb44e892a14b983ea80c6e90b20c1a53bc38d5df1fecf2dbb48

File: ./contracts/mocks/PoolAllowlist.sol
SHA3: 47bf7e2fb744f34ea4bbf23b03531c05bb707290bd54ea2041f10ef430b4f62f

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

File: ./contracts/mocks/PoolMerkle.sol
SHA3: 7dc09da3a0190d3f7ba87c6f1ddc7bf9e21db7a07289023fbff8420f9761b573

Second review scope

Repository:
https://github.com/buidler-labs/headstarter-contracts
Commit:
7284d2a
Technical Documentation:
Type: Business Logic and Technical Documentation
Link: https://github.com/buidler-labs/headstarter-contracts#readme

JS tests: Yes
Deployed Contracts Addresses: No
Contracts:

File: ./contracts/BasicPool.sol
SHA3: bb433ba3e10fa50d58f40a9b1364661fa86aab621ef651909ccc47feff1fe3b1

File: ./contracts/extensions/VestingPool.sol
SHA3: 374808973b963785c6a2c386bb3e106dd2f7324d76fa9cb57a20860e5a322600

File: ./contracts/extensions/AllowlistPool.sol
SHA3: 7e6fdbc199fdf698d6091d7c7172adb23bb80515897b7175e2b5be5035148453

File: ./contracts/IDOPoolWithVesting.sol
SHA3: c3894a9d86ac32beb6a459f15f05ebc1a784f794f8576cbe4a635e50957122ac

File: ./contracts/mocks/PoolWithVesting.sol
SHA3: c4d63d94a830034b1b07469bbfae419e9b8c32fa9c149b3c15e1dfe8f475da58

File: ./contracts/extensions/MultiPartyWithdrawPool.sol
SHA3: b125fb65eee9661b1397ed7fe17968ac0f3e6ae08b230c3993d4a5c35d3e64f1

File: ./contracts/extensions/MerklePool.sol
SHA3: f5b97d0db507e5e44cf4da9e6a9586b92dccdcc7ceb4960067be64428f5a180e

File: ./contracts/mocks/PoolMultiPartyWithdraw.sol
SHA3: 44ba07bfb1c5dcec5637f70637c62bf001cdb4bc949379e5a872a800609d9f1e

File: ./contracts/IDOPool.sol
SHA3: fd581a1bd3fb02cfc8bb7e70a59654957800f42edf62561a178399838362ceb7

File: ./contracts/mocks/PoolAllowlist.sol
SHA3: af4fb73aeca1525fc6a77de4ddab4b3123d36f63a869eeb467d27d6508538539

File: ./contracts/interfaces/IPool.sol
SHA3: 8e7bf445eeb903d3b0f3ccf37311293d6ece1d7404de5ad0737b4ff729bd5fc4

File: ./contracts/mocks/PoolAllowlist.sol
SHA3: af4fb73aeca1525fc6a77de4ddab4b3123d36f63a869eeb467d27d6508538539

File: ./contracts/mocks/PoolMerkle.sol
SHA3: 09c3560d2916191eedf035464b205b0e2a084841760f1dd9afb3b218f1c60446

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

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

Code quality

The total CodeQuality score is **10** out of **10**. Tests were provided.

Architecture quality

The architecture quality score is **10** out of **10**.
The Customer provided pre-populated config and environment details to start from the box.

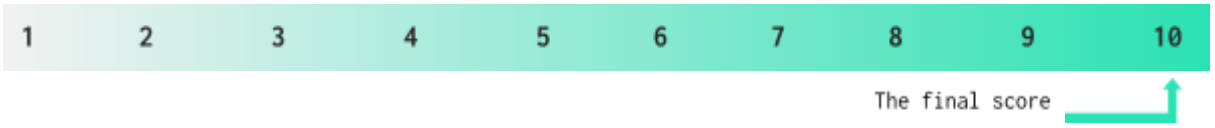
Security score

As a result of the audit, the code contains **1** medium and **1** low severity 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**.



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

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	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.	Passed
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Failed
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 destroyed until it has funds belonging to users.	Passed
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Uninitialized Storage Pointer	SWC-109	Storage type should be set explicitly if the compiler version is < 0.5.0.	Not Relevant
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Not Relevant
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.	Passed
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless it is 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.	Passed
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	Signed messages should always have a unique id. A transaction hash should not be used as a unique id.	Passed
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.	Passed
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.	Not Relevant
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.	Not Relevant
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.	Passed
Requirements Compliance	**Custom**	The code should be compliant with the requirements provided by the Customer.	Passed
Repository Consistency	**Custom**	The repository should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
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

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

System Overview

Headstarter - is pool/vesting system with the following contracts:

- *BasicPool.sol* – basic pool contract with abilities to join the pool, claim tokens eventually.
  It has the following attributes:
  - Token address: ERC20 Token
  - Token claim max.: Maximum amount of tokens can be claimed by the user.
  - Start date: date when pull will be opened
  - End date: date when pull will be closed
  - Redeem date: date when tokens can be claimed
- *AllowlistPool.sol* - contract, which adds facilities to whitelist users for participation in the pool.
- *MerklePool.sol* - contract, which adds facilities to whitelist users based on the "*Merkle*" algorithm, meaning whitelisted users will not be saved on the blockchain, only their total hash.
- *MultiPartyWithdrawPool.sol* - contract which adds facilities where the raised funds can be withdrawn by multiple parties.
- VestingPool.sol - contract, which adds vesting facilities so that user's tokens will be claimed portionally according to specified intervals.
- IDOPool.sol - contract based on *BasicPool* with *AllowlistPool* facilities. Combination of these 2 contracts.
- IDOPoolWithVesting.sol - contract based on *IDOPool* with *VestingPool* facilities. Combination of these 2 contracts.

Privileged roles

- The owner can:
  - transfer ERC20 tokens to the pool
  - withdraw the raised funds
  - set pool schedule: start date, end date, redeem date
  - set IDO details: max tokens per user, price per token
  - add/remove users from whitelist
  - initiate withdrawal raised funds for all parties
  - set vesting schedule

Notice

- The code is designed specifically for hedera and should not be on other networks where bigger variables size is commonly used.

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

Findings

Critical

No critical issues were found.

High

1. **Potential DoS.**

   The function iterates over all shareholders and withdraws tokens.

   Gas consumption can differ a lot between different transactions. Possible DoS if the number of shareholders is large enough. Moreover, the holder's address can be a contract, which consumes lots of Gas when sending tokens to it.

   Contracts:MultiPartyWithdrawPool.sol

   Function: withdraw

   Recommendation: Use Pull over Push pattern or limit the amount of data that can be processed in one translation.

   Status: Fixed (Revised commit: 7284d2a)

2. **Highly permissive owner access**

   In vesting extension, the owner has the ability to change the release percentages and release interval durations whenever he/she wants. Moreover, the owner is allowed to change the start, end, and redeem dates in the BasicPool contract.

   This may lead to a delay in promised release.

   Contract: VestingPool.sol, BasicPool.sol

   Function: setVestingSchedule, setPoolSchedule

   Recommendation: Do not allow those parameters to be rescheduled when a pool is started.

   Status: Fixed (Revised commit: 7284d2a)

3. **Unchecked call return value**

   The return value of the call to the precompiled HTS(Hedera Token Service) contract is not checked.

   If a user joins the pool only one time and the call to the HTS contract fails, the user will not be associated with the token and will not be able to receive the tokens.

   Contract: BasicPool.sol

   Function: _associateToken

   Recommendation: Check two conditions and accept both as a success. If the HTS associateToken response code for the status is 22 (SUCCESS),

or it is 194(TOKEN_ALREADY_ASSOCIATED_TO_ACCOUNT). For other statuses, return false and revert the transaction.

Status: Fixed (Revised commit: 7284d2a)

Medium

1. Typo in error message

The error says "Invalid internal" whereas "Invalid interval" should be.

Contracts:VestingPool.sol

Function: _setVestingSchedule

Recommendation: Fix typo.

Status: New

Low

1. Typo in Documentation

_amountToClaim method does not exist.

Recommendation: Most probably right method name is *_getAmountToClaim*

Status: Fixed (Revised commit: 7284d2a)

2. Typo in Documentation

`*/scripts/merkle-generator*` missing in the repository.
`generate the Merkle Root` link leads nowhere.

Recommendation: Add script and fix the link.

Status: Fixed (Revised commit: 7284d2a)

3. Floating pragma

The contracts use floating pragma ^0.8.7.

Contract: all

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

Status: Fixed (Revised commit: 7284d2a)

4. Outdated Compiler Version

Using an outdated compiler version can be problematic, especially if publicly disclosed bugs and issues affect the current compiler version.

Contract: all

Recommendation: Use a recent version of the Solidity compiler.

Status: Reported

5. **Style guide violation.**

   Contracts do not fully follow the Solidity code style guide.

   Contracts: all

   Recommendation: Follow the official Solidity code style guide.

   Status: Fixed (Revised commit: 7284d2a)

6. **Requirements incompliance.**

   IDOPool contract refers to a pausable contract, but the code implementation demonstrates the opposite.

   The comment line is misleading.

   Contract: IDOPool.sol

   Function: @notice comment line

   Recommendation: Remove the comment stating it is pausable.

   Status: Fixed (Revised commit: 7284d2a)

7. **The confusing function name.**

   The function is named as getter function, but it mutates data.

   Contract: BasicPool.sol

   Function: _getAmountToClaim

   Recommendation: rename the function.

   Status: Fixed (Revised commit: 7284d2a)

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

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 audit makes no statements or warranties on the security of the code. It also cannot be considered a 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.

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 audit cannot guarantee the explicit security of the audited smart contracts.