HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Genie Swap
Date: 29 September, 2023

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

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 Genie Swap
Approved By	Paul Fomichov | Lead Solidity SC Auditor at Hacken OÜ
Tags	ERC20 token; Launchpad
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://genieswap.com/
Changelog	06.09.2023 - Initial Review 29.09.2023 - 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 Genie Swap (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

The Genie Swap audited project comprises Launchpad contracts, which offer a foundation for token launches, enhanced by multi-level-marketing (MLM) features. This MLM functionality enables the allocation of commissions to individuals involved in promotional efforts, including special addresses like launchpad partners and designated addresses.
The MLM system organizes users into a tree structure. Every user has a referrer, except the topmost user. The tree helps in determining commission distributions.

- Hierarchy: As you traverse up the tree (from user to referrer), rewards are computed based on MLM levels (ranging from Level 1 to Level n).
- Designated Address Rewards: Once all MLM levels are exhausted, if a referrer is a designated address, it earns a designated address share.
- Partner Benefits: If a referrer is a partner, they receive both the MLM level share (if they are within the MLM levels) and the partner share.
- Single Reward for Designated Addresses: Only the first designated address identified during the upward traversal gets a reward.
- Double Incentive for Partners: Partners always receive their partner-share, irrespective of their tree position. Moreover, they can also earn MLM rewards based on the outlined rules.

The files in the scope:

- **MyLaunchpadToken.sol** - The ERC20 token that is used in the Launchpad.
- **LaunchpadToken.sol** - Interface for MyLaunchpadToken.sol.
- **Escrow.sol** - Responsible for holding and releasing assets for beneficiaries, which also ensures certain conditions are met.
- **Launchpad.sol** - Facilitates token launches and sales on a launchpad. It enables users to mint Launchpad tokens in exchange for payment tokens. The contract is the central component of the system.
- **Manager.sol** - Responsible for the management of various aspects of the system, from token exchange rates to partners and designated addresses.
- **MLM.sol** - Handles the multi-level marketing (MLM) functionality of the system. This contract is responsible for onboarding users,

calculating and distributing rewards based on referrals, and keeping track of the MLM structure.

- **Types.sol** - The library that defines various data structures (structs) used across the project.
- **Utility.sol** - The library that provides utility functions to be used across other contracts in the project.
- **Error.sol** - The function that stores various error definitions to be used in reverted calls.
- **LaunchpadFactory.sol** - The factory contract for deploying and managing Launchpad contract instances.

Privileged roles

- Owner of the MLM/Manager contract: Can pause minting of tokens in Launchpad instances, set the commission mode for MLM, set the token exchange rate, set the start time of the Launchpad if its not started yet, set the end time of the Launchpad if it is not ended yet, remove partners and designated addresses, manage designated addresses, manage partner addresses, and set MLM levels.
- Deployer of LaunchpadToken: Can use the LaunchpadFactory to deploy Launchpad instances. Can also be an Ownable ERC20 tokens owner.

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 **10** out of **10**.

- Functional requirements are provided.
- Technical description is provided.
- Description of the development environment is present.

Code quality

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

- NatSpec is consistent.
- Style Guide is followed.
- The development environment is configured.

Test coverage

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

- Deployment and basic user interactions are covered with tests.
- Negative cases coverage is present.
- Interactions by several users are tested thoroughly.

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.

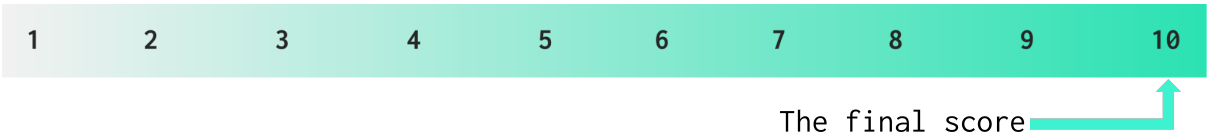

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
6 September 2023	3	4	1	0
29 September 2023	0	0	0	0

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

Risks

- The Launchpad contract relies on external contracts (third party tokens) and their integrity can not be certified. Admin and users should make sure the tokens used in the Launchpad are safe and can not bring issues to investors' funds.

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 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.	Not Relevant	
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	

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

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 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.	Passed	
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	

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

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. Contracts shouldn't rely on values that can be changed in the same transaction.	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	

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 high severity issues were found.

Medium

M01. Checks-Effects-Interactions Pattern Violation

Impact	Medium
Likelihood	Medium

Some of the state variables are updated after the external calls.

As explained in Solidity Security Considerations, it is best practice to follow the checks-effects-interactions pattern when interacting with external contracts to avoid reentrancy-related issues.

Path: ./contracts/LaunchpadFactory.sol : deployLaunchpad()

Recommendation: Follow the checks-effects-interactions pattern when interacting with external contracts.

Found in: 358803d

Status: Fixed (revised commit: 090f9f4)

M02. Requirements Violation: Contradicting NatSpec

Impact	Low
Likelihood	High

In some functions of the Manager.sol contract, the implementations are contradicting their NatSpec descriptions.

- In the *managePartnerAddress()* function, events are in fact emitted, contradicting the NatSpec.
- In the *getDesignatedAddresses()* and *getPartners()* functions, any address can make the call, contradicting the NatSpec.

This may lead to unexpected behavior.

Path: ./contracts/lib/Manager.sol : managePartnerAddress(), getDesignatedAddresses(), getPartners()

Recommendation: Fix the mismatch between the NatSpec and implementation.

Found in: 358803d

Status: Fixed (revised commit: 090f9f4)

M03. Requirements Violation: Incomplete Pause Mechanism Implementation

Impact	High
Likelihood	Low

The contract Manager extends the Pausable contract from OpenZeppelin. It implements the *pause()* function, but there is no *unpause()* function in the contract.

When paused, the function isActive() returns false and that blocks the functionalities of the Launchpad contract, that extends the Manager contract and uses the onlyWhenActive() modifier.

Path: ./contracts/lib/Manager.sol

Recommendation: Implement a "unpause" feature or change the purpose, naming and documentation of the *pause()* function.

Found in: 358803d

Status: Fixed (revised commit: 090f9f4)

M04. Requirements Violation: Insufficient Balance

Impact	High
Likelihood	Low

In the function *_release()* from the Escrow contract, there is the following code:

```
/* As we pay out commissions in the launchpad token, this might happen */
if (token.balanceOf(address(this)) < amount) {
    revert FatalEscrowError();
}
```

The comment explicits that issues may happen with mixed balances and invalid calculations, resulting in insufficient balance to pay out launchpad participants.

Path: ./contracts/lib/Escrow.sol: _release()

Recommendation: Improve code in order to make it impossible to have such issues with insufficient balance.

Found in: 358803d

Status: Fixed (revised commit: 090f9f4)

Low

L01. Inefficient Gas Model

Impact	Low
Likelihood	Medium

In some functions, there are loops that iterate over *storage* variables, causing Gas usage to be higher.

Path: ./contracts/lib/Manager.sol: removePartner(), removeDesignatedAddress(), manageDesignatedAddress(), managePartnerAddress()

Recommendation: Use memory instead of storage for read purposes.

Found in: 358803d

Status: Fixed (revised commit: 090f9f4)

L02. Redundant Check

Impact	Low
Likelihood	Medium

In the removeDesignatedAddress() function, the following checks are done:

```
if (da == address(0)) {
    revert InvalidAddress();
}
```

```
if (!_knownDesignatedAddresses[da]) {
    revert InvalidAddress();
}
```

The first check is useless as the second check asserts that the address is set. The 0x0 address will never be a valid key for the *_knownDesignatedAddresses* mapping as when calling the manageDesignatedAddress() function, it is not possible to set the 0x0 address as a designated address.

Path: ./contracts/lib/Manager.sol: removeDesignatedAddress()

Recommendation: Remove redundant code.

Found in: 358803d

Status: Fixed (revised commit: 090f9f4)

L03. Ambiguous Error Message

Impact	Medium
Likelihood	Low

In the *managePartnerAddress()* function, the error *OutOfBoundary()* is being used for both cases: when the *_partners* would exceed the maximum size and if the *totalPartnerShares* would exceed the maximum total shares.

This can cause confusion to the final user, not knowing what exactly was the issue – either exceeding the maximum partners amount or the maximum total shares amount.

Path: ./contracts/lib/Manager.sol: managePartnerAddress()

Recommendation: Improve error emission in order to be descriptive and not cause confusion.

Found in: 358803d

Status: Fixed (revised commit: 090f9f4)

Informational

I01. Floating Pragma

The project uses floating pragmas *^0.8.9*.

This may result in the contracts being deployed using the wrong pragma version, which is different from the one they were tested with. For example, they might be deployed using an outdated pragma version, which may include bugs that affect the system negatively.

Path: ./contracts/*.sol

Recommendation: Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment. Consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen.

Found in: 358803d

Status: Fixed (revised commit: 090f9f4)

I02. Solidity Style Guide Violation

Contract readability and code quality are influenced significantly by adherence to established style guidelines. In Solidity programming, there exist certain norms for code arrangement and ordering. These guidelines help to maintain a consistent structure across different contracts, libraries, or interfaces, making it easier for developers and auditors to understand and interact with the code.

The suggested order of elements within each contract, library, or interface is as follows:

- Type declarations
- State variables
- Events
- Modifiers
- Functions

Functions should be ordered and grouped by their visibility as follows:

- Constructor
- Receive function (if exists)
- Fallback function (if exists)
- External functions
- Public functions
- Internal functions
- Private functions

Within each grouping, view and pure functions should be placed at the end.

Furthermore, following the Solidity naming convention and adding NatSpec annotations for all functions are strongly recommended. These measures aid in the comprehension of code and enhance overall code quality.

Paths:
./contracts/interfaces/LaunchpadToken.sol: LaunchpadToken → ILaunchpadToken (naming convention)
./contracts/lib/Escrow.sol
./contracts/lib/Launchpad.sol
./contracts/lib/Manager.sol
./contracts/lib/Manager.sol: MlmUserExists → mlmUserExists (naming convention)
./contracts/lib/MLM.sol
./contracts/LaunchpadFactory.sol

Recommendation: Consistent adherence to the official Solidity style guide is recommended. This enhances readability and maintainability of the code, facilitating seamless interaction with the contracts. Providing comprehensive NatSpec annotations for functions and following Solidity's naming conventions further enrich the quality of the code.

Found in: 358803d

Status: Fixed (revised commit: 090f9f4)

I03. Missing Variable Explicit Visibility

Some variable visibilities are not explicitly set. This results in harder code maintenance and readability.

Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.

Paths:
./contracts/LaunchpadFactory.sol : _launchpadTokens,
./contracts/lib/Manager.sol : _mlmCommissionMode

Recommendation: Set the visibilities of variables explicitly.

Found in: 358803d

Status: Fixed (revised commit: 090f9f4)

I04. Missing Variable Explicit uint Size

Some `uint` variable sizes are not explicitly set. This results in harder code maintenance and readability.

`uint` variable sizes should be set explicitly.

Paths:
./contracts/lib/Escrow.sol : Released.amount, _holdings, releasable()(uint), _release()(uint), _release.amount

./contracts/lib/Launchpad.sol : TokensMinted.amount, TokensMinted.tokenExchangeRate, mint(amountPaymentToken)(amountLaunchpadToken), _mint(amountPaymentToken)(amountLaunchpadToken), mint().totalPaymentTokenCommission, _mint().paymentTokenCommission, _mint().launchpadTokenCommission, _mint().remainingPaymentToken, getPrice()(price), getPrice().answer, getPrice().timeStamp, quota(amountPaymentToken)(payout), quota().usdValuePaymentToken, quota().launchpadTokenDecimals, quota().base, topUpLaunchpadTokens(topUpAmount), recoverLaunchpadToken().amount, release()(uint)

./contracts/lib/Manager.sol : PartnerChanged.share, DesignatedAddressChanged.share, StartChanged.start, EndChanged.end, MlmLevelsChanged.levels, BASIS, MAX_LAUNCHPAD_TIME, MAX_TOKEN_EXCHANGE_RATE, MAX_MLM_LEVELS, MAX_NUM_PARTNERS, MAX_NUM_DESIGNATED_ADDRESSES, MAX_MLM_TREE_DEPTH, MAX_TOTAL_SHARE_PARTNERS, MAX_TOTAL_SHARE_MLM_LEVELS, MAX_TOTAL_SHARE_DESIGNATED_ADDRESS, _tokenExchangeRate, _start, _end, _designatedAddressesShare, _mlmLevels, setTokenExchangeRate(tokenExchangeRate), getTokenExchangeRate()(uinit), setStart(start), setEnd(end), removePartner.indexToDelete, removeDesignatedAddress.indexToDelete, manageDesignatedAddress(share), getStart()(uint), getEnd()(uint), managePartnerAddress(share), managePartnerAddress().totalPartnerShares, setMlmLevels(mlmLevels), setMlmLevels.totalRate, getMlmLevels()(uint[])

./contracts/lib/MLM.sol : _getCommissions(amountPaymentToken, amountLaunchpadToken), _getCommissions.rewardCounter, _getCommissions.treeDepth, _getCommissions.share

./contracts/lib/Types.sol : Commission.amount, CommissionPaid.amount, PartnerConfig.share, DesignatedAddressConfig.share, LaunchpadConfig,

Recommendation: Set the *uint* sizes explicitly.

Found in: 358803d

Status: Reported (contract Manager.sol contains “uint” variables)

I05. Redundant Modifier Usage

In some occurrences of the *nonReentrant* modifier, the usage is redundant, as the functions do not perform state updates.

This will result in unoptimized Gas usage.

Paths:
./contracts/lib/Launchpad.sol : topUpLaunchpadTokens(), recoverLaunchpadToken()

Recommendation: Remove redundant modifier call.

Found in: 358803d

Status: Fixed (revised commit: 090f9f4)

I06. Unindexed Events Emissions

Some events emitted in the system are not using indexed variables.

Having indexed parameters in the events makes it easier to search for these events using indexed parameters as filters.

Paths:
./contracts/lib/Escrow.sol : Released → asset
./contracts/lib/MLM.sol : UserOnboarded → user
./contracts/lib/Types.sol : CommissionPaid

Recommendation: Use the *indexed* keyword to the relevant event parameters

Found in: 358803d

Status: Fixed (revised commit: 090f9f4)

I07. Public Functions That Should Be External

Functions that are meant to be exclusively invoked from external sources should be designated as *external* rather than *public*. This is essential to enhance both the Gas efficiency and the overall security of the contract.

Path: ./contracts/lib/Manager.sol : isPending(), getMlmLevels()

Recommendation: Transition the relevant functions, which are exclusively utilized by external entities, from their current *public* visibility setting to the *external* visibility setting.

Found in: 358803d

Status: Fixed (revised commit: 090f9f4)

I08. Contracts That Should Be Interface

Contracts without any logic and only type definition should be interfaces.

Path: ./contracts/lib/Error.sol

Recommendation: Transform the Error contract into an interface.

Found in: 358803d

Status: Fixed (revised commit: 090f9f4)

I09. Missing Events for Critical Value Updates

Events should be emitted after sensitive changes take place, to facilitate tracking and notify off-chain clients following the contract's activity.

Path: ./contracts/lib/Manager.sol: setTokenExchangeRate()

Recommendation: Consider emitting events in said functions.

Found in: 358803d

Status: Fixed (revised commit: 090f9f4)

I10. Redundant Variable Assignment

When updating mappings, instead of reassigning the variable value in order to reset its value, it is cheaper to use the delete statement instead of reassigning the default value.

Path: ./contracts/lib/Manager.sol:

- removePartner() → _knownPartners[partner]
- removeDesignatedAddress() → __knownDesignatedAddresses[da], _designatedAddressesShare[da]

Recommendation: Use the *delete* keyword instead of reassigning the variable value.

Found in: 358803d

Status: Fixed (revised commit: 090f9f4)

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 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.

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

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.

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

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/PROJECT-ONRAMP/launchpad-contracts
Commit	358803d6e92c45dda120ca87efefe7bcb23f1af0
Whitepaper	Link
Requirements	Link
Technical Requirements	Link
Contracts	File: contracts/LaunchpadFactory.sol SHA3: 5ed2969c8d6d339dba5dee9d0f25d88b2a6bd49f3f84acd8a9696ebf2df0b3db File: contracts/examples/MyLaunchpadToken.sol SHA3: 628cd616f67c40df8dcad57932267b348478d3fbde4c4abaa60a546302ecba10 File: contracts/interfaces/LaunchpadToken.sol SHA3: f6a3347a13455139b1d2dd46a50354591d97cdfb273757557599a863cac23fb1 File: contracts/lib/Error.sol SHA3: 6ae802b18d2f863b8109b3594dc29cce0bab74fa8ba282112e12df3e6a86a7d9 File: contracts/lib/Escrow.sol SHA3: bcff40c7438519f498cfe70d90ed74bb4b8223fa05648785670a58864a29af9c File: contracts/lib/Launchpad.sol SHA3: 3433e9bfb96bb13eb0bc3c634f52f01d960e0465c642762c5f38e2ada1cf3ce2 File: contracts/lib/Manager.sol SHA3: 87f7651c410599efcf59538b5a7b2994b354f8963c75a3c15a7508a3bbe37f88 File: contracts/lib/MLM.sol SHA3: 89eec2f0f85fd53d04fb9789953354864ac07c399f551e900ed56664bab49546 File: contracts/lib/Types.sol SHA3: 216169eaaa3c61d0417201bd9095299053b354cc204ef235d3f640d4086665e8 File: contracts/lib/Utility.sol SHA3: 0d8311276efe42050101f54e1cc3a4829cc21577b8f14a1b83430084587a8258

Second review scope

Repository	https://github.com/PROJECT-ONRAMP/launchpad-contracts
Commit	090f9f4496364eb95c3c415892b5e0e78b3db7d9
Whitepaper	Link
Requirements	Link

Technical Requirements	Link
Contracts	File: contracts/LaunchpadFactory.sol SHA3: 2cac10d7322787e2ef1d50dace6caa30a922fb7829ac96e4f0859b95b7c05893 File: contracts/examples/MyLaunchpadToken.sol SHA3: 13c9968afbc3c93d1fe31f7be5a6c8d1869660e073a152371f2af6a41a0d9033 File: contracts/interfaces/ILaunchpadToken.sol SHA3: f56f73aac0a731e8323d2aa7665eda725590c7c383eac88bf0b8fdac2a5dca07 File: contracts/lib/Error.sol SHA3: 278d44b423655fb4fc8ef5f2f207c4936cf04f5ba2ae4df3a87af4ecb4594cda File: contracts/lib/Escrow.sol SHA3: 90541ac9d9c1780bb8d9a240338d6bc40d203cf3c6bc18a456d3b688b7a390f4 File: contracts/lib/Launchpad.sol SHA3: 5b4586791e304e780a96711674b348f9b7f199bdadfabab97b16a750b084a119 File: contracts/lib/Manager.sol SHA3: 80cb17ae56641e7e27eeb61a0e4c89f5029293cc3e02dc3e1688c53f63a01cef File: contracts/lib/MLM.sol SHA3: 49f6284e289171daceedb137d82bb229e5e8a087a534be3197b81f5d41831f36 File: contracts/lib/Utility.sol SHA3: 72b40388b336176a7965769832ddcbd679e23e486a05cebc23e60d09d22e23bc File: contracts/lib/Types.sol SHA3: c54efb9b2a00787771d16c2076f5b8dfbfaec68ae01e6ec0464e59e2a4f7e06c