HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: MetaTime
Date: 12 Nov, 2023

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 MetaTime
Approved By	Luis Buendia | Senior Solidity SC Auditor at Hacken OÜ
Tags	MetaChain
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://metatime.com/
Changelog	27.10.2023 - Initial Review 12.11.2023 - Remediation 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 MetaTime (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

MetaTime pools ... The files in the scope*:*

- **Distributor** - This contract is designed to facilitate the distribution of coins over a specified period of time. It allows users to claim their share of coins based on the distribution rate and period length.
- **LiquidityPool** - This contract is a smart contract used for managing a liquidity pool. It allows the owner to deposit coins into the pool and withdraw them as needed.
- **StrategicPool** - This contract is designed for managing a strategic pool of coins. It allows the owner of the contract to burn coins from the pool using a formula or without using a formula.
- **TokenDistributor** - This contract is a Solidity smart contract designed for distributing coins among users over a specific period of time. It allows the contract owner to set claimable amounts for users before the claim period starts and enables users to claim their coins during the distribution period. Any remaining coins after the claim period can be swept by the contract owner.
- **TokenDistributorWithNoVesting** - The TokenDistributorWithNoVesting contract is designed for distributing coins during no vesting sales. It allows the contract owner to set claimable amounts for users and enables users to claim their coins within a specified claim period. Additionally, any remaining coins can be swept from the contract by the owner after the claim period ends.
- **Trigonometry** - Library contract implemented for trigonometry calculations used in the burn with formula function.
- **PoolFactory** - Contract that deploys Distributor and TokenDistributor contracts.

Privileged roles

- The owner of each contract can set and change multiple configuration values and also access restricted functionalities.

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**.

- High level overview documentation has been created.
- Technical descriptions have been provided and NatSpec is extensive.

Code quality

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

- The code is structured and readable.
- The Gas model is optimized.

Test coverage

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

- Deployment and some basic user interactions are covered with tests.
- The extended code coverage was done after the security review.

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.

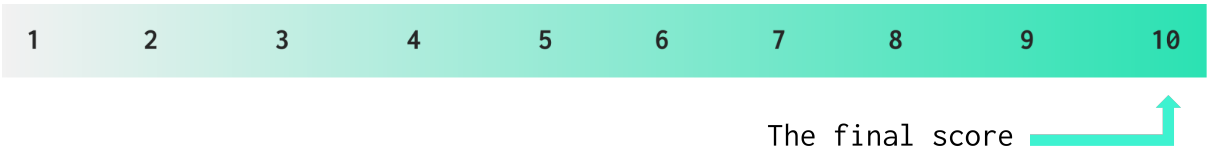

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
27 October 2023	1	0	0	0
12 November 2023	0	0	0	0

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

Risks

- The protocol is centralized and depends on a multisignature account to operate the reviewed contracts. Nevertheless, the MetaTime team heavily protects these accounts and always operates in the best interests of their community.

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

No medium severity issues were found.

Low

L01. Lack of parameter validation on *Distributor* initialization

Impact	Low
Likelihood	Low

The *Distributor* contract does not validate the parameters *lastClaimTime*, *claimableAmount* and *leftClaimableAmount*. Although these parameters are introduced by the owner, there are several wrong value combinations that result in unexpected reverts during the contract execution.

Path: ./contracts/core/Distributor.sol : isParamsValid()

Recommendation: Validate the parameters to avoid arriving at undesired states. Remove any unused parameters from the modifier.

Found in: 692f9732352da63878e983a3be663dffbec39c01

Status: Fixed (Revised commit: 501767f)

Remediation: The fix removed unnecessary validations on the *isParamsValid* modifier and added validations on the function *initialize* for other required parameters.

Informational

I01. Use Custom Errors

Custom errors from Solidity 0.8.4 are cheaper than revert strings (cheaper deployment cost and runtime cost when the revert condition is met). Source Custom Errors in Solidity: Starting from Solidity v0.8.4, there is a convenient and gas-efficient way to explain to users why an operation failed through the use of custom errors. Until now, you could already use strings to give more information about

failures (e.g., revert(‘‘Insufficient funds.‘‘);), but they are rather expensive, especially when it comes to deployment cost, and it is difficult to use dynamic information in them.

Path: ./contracts/core/Distributor.sol

./contracts/core/LiquidityPool.sol

./contracts/core/StrategicPool.sol

./contracts/core/TokenDistributor.sol

./contracts/core/TokenDistributorWithNoVesting.sol

Recommendation: Consider replacing strings for custom errors.

Found in: 692f9732352da63878e983a3be663dffbec39c01

Status: Acknowledged

Remediation: Given the current developed testing framework the required time to adopt these changes is not acceptable.

I02. Initialized Variable to Default Value

Initializing variables to default value executes an extra order that is not required.

Path: ./contracts/core/Distributor.sol : calculateClaimableAmount()

./contracts/core/StrategicPool.sol

./contracts/vesting/TokenDistributor.sol

Recommendation: Consider avoiding initializing variables to default value.

Found in: 692f9732352da63878e983a3be663dffbec39c01

Status: Fixed (Revised commit: 501767f)

Remediation: The recommended initialized variables to default values were removed.

I03. Return value on *_withdraw* function of LiquidityPool contract not used

The _withdraw function returns a boolean value that is never used on the transferFunds function that calls it.

Path: ./contracts/core/prize-pool/StakePrizePoolV2.sol: _liquidate()

Recommendation: Consider using the returned value or removing it if necessary.

Found in: 692f9732352da63878e983a3be663dffbec39c01

Status: Fixed (Revised commit: be33499)

Remediation: The unnecessary return value was removed.

I04. Use two step ownership transfer pattern

The two step ownership transfer pattern ensures a more robust approach to change the ownership of the contract. The contracts included on the genesis file do not require this pattern.

Path: ./contracts/core/Distributor.sol

./contracts/core/TokenDistributor.sol

./contracts/core/TokenDistributorWithNoVesting.sol

./contracts/utils/PoolFactory.sol

Recommendation: Consider using the two step ownership transfer pattern to leverage the security of the relevant contracts.

Found in: 692f9732352da63878e983a3be663dffbec39c01

Status: Fixed (Revised commit: 1a3d338)

Remediation: The two step owner transfer was implemented on the recommended contracts using the *Ownable2Step* contract from OpenZeppelin.

I05. PoolFactory implementation contracts can be set to zero or incorrect values leaving the contract non functional

The initialize function of the PoolFactory contract sets the global state address variables for the implementation contracts to clone. However, the initialize function does not perform any validation on the received input and the contract does not contain any setter function that allows to change these values if required.

Recommendation: Consider creating a setter for these values or performing a minimum validation during the initialization function.

Found in: 692f9732352da63878e983a3be663dffbec39c01

Status: Fixed (Revised commit: 8e9eb69)

Remediation: The contract validates that the introduced parameters are not zero.

I06. Change return value to avoid extra SLOAD

The current implementation of the functions createDistributor and createTokenDistributor use a global variable twice when it is not required.

Recommendation: Consider returning the variable and adding the plus plus sign at the beginning to avoid an extra SLOAD. Ex:

```
return distributorCount++;
```

Found in: 692f9732352da63878e983a3be663dffbec39c01

Status: **Fixed** (Revised commit: 2220e61)

Remediation: The unnecessary code lines were removed.

I07. lastBurnedAmount of StrategicPool is redundant

The global state variable *lastBurnedAmount* of the *StrategicPool* contract is never used and the value is emitted on an event.

Recommendation: Consider removing the variable to save Gas.

Found in: 692f9732352da63878e983a3be663dffbec39c01

Status: **Fixed** (Revised commit: a79f951)

Remediation: The unnecessary global state variable was removed.

I08. Consider controlling casting overflow on StrategicPool

The functions *burn* and *burnWithFormula* of the StrategicPool contract perform cast from unsigned integer to signed integer. Although it is unlikely to happen, it is possible that the value in that casting overflows resulting in a negative number.

Recommendation: Consider controlling the values after the casting before using them.

Found in: 692f9732352da63878e983a3be663dffbec39c01

Status: **Fixed** (Revised commit: a79f951)

Remediation: The casting was controlled using *SafeCast* library from OpenZeppelin.

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

I09. Missing NatSpec documentation for *leftClaimableAmount*

The parameter leftClaimableAmount does not contain NatSpec documentation.

Recommendation: Consider writing the natspec documentation for the missing variable.

Found in: 692f9732352da63878e983a3be663dffbec39c01

Status: Fixed (Revised commit: 21b9328)

Remediation: The missing NatSpec was added to the contract.

I10. Owner Can Renounce Itself

The owner can renounce itself creating a problem to operate with high privileged functions.

Recommendation: Consider erasing the renounce owner function.

Found in: 692f9732352da63878e983a3be663dffbec39c01

Status: Mitigated

Remediation: Given the multisignature wallet, it is highly unlikely to arrive in this scenario.

I11. Centralization Risk

The protocol is heavily centralized. This is not a risk by itself. However, it is important to notice that any unauthorized access to the owner accounts can jeopardize the protocol stability.

Recommendation: Use multisignature wallet for privileged accounts or any additional protection mechanism over the privileged account.

Found in: 692f9732352da63878e983a3be663dffbec39c01

Status: Mitigated

Remediation: The project uses a multisignature wallet and complies with best security practices.

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/Metatime-Technology-Inc/pool-contracts/
Commit	692f9732352da63878e983a3be663dffbec39c01
Whitepaper	https://docs.metatime.com/home/
Requirements	N/A
Technical Requirements	N/A
Contracts	File: contracts/core/Distributor.sol SHA3: f1c2984f0202819d71af11148e0ce158740e4788841517ed90656c1212842cb3 File: contracts/core/LiquidityPool.sol SHA3: 18ea7ade67eb6896a64cb39a19765d503b687d5a450564535e38cede2fe9d9ab File: contracts/core/StrategicPool.sol SHA3: 5cd958c4e194c95204b8770ceda6249535e2abde187d255e10ed5ae4847d0040 File: contracts/core/TokenDistributor.sol SHA3: a02cece531848032274c774d684e77f7fbce1d5b6f7e0a2e706071626534db87 File: contracts/core/TokenDistributorWithNoVesting.sol SHA3: 85d0635d7e55b52f7194af17284b3a2bda63b422d4cfa35e05ee9e46626df5b5 File: contracts/utils/PoolFactory.sol SHA3: 07f1aca93e2f8fa80a85b9e2dd6f0e72fe1b0f593e48e4674fae1705b3d7c69e File: contracts/libs/Trigonometry.sol SHA3: 260290f6fc7484cbf53b745e7a26b07a32988ef7e5fbd84f7c18ad296cffd3cf

Remediations review scope

Repository	https://github.com/Metatime-Technology-Inc/pool-contracts/
Commit	73fc62c5410f868b6d2c24307be93d2401787221
Whitepaper	https://docs.metatime.com/home/
Requirements	N/A
Technical Requirements	N/A
Contracts	File: contracts/core/Distributor.sol SHA3: 96a337ad1fd387264dc76294abdf6e7ece0dd4703f61bbb9421a707487ade84b File: contracts/core/LiquidityPool.sol SHA3: c335c07ae9179f28936b54fe3b43e4a53eaf4f973a86f8ff424bb28c25722a5b File: contracts/core/StrategicPool.sol SHA3: cdf6e5d2f206d4a36c2762b82f785ffc391e707a01d2459fb41f9cb4bfd01567 File: contracts/core/TokenDistributor.sol SHA3: 29567e9262730e817b1f004a92bbf05a93380b82b7e88bce3f659ddcfab4a05c File: contracts/core/TokenDistributorWithNoVesting.sol SHA3: 7e33a0ce1398547a879d30bbdfa9144aa252c8a712a8402d9ad828916016b3ff File: contracts/utils/PoolFactory.sol SHA3: 69adbeacc741d5c4ad374e90e63400406434e1d5f652a81b9f10f1d2b72bb105 File: contracts/libs/Trigonometry.sol SHA3: 202a89b61c152023b757c1509a0b577cc681177c1e9b1c2f6d1f09756a2ae855