HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Medieval Empires
Date: November 14th, 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 Medieval Empires
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU
Type	ERC20 token; Tokens sale
Platform	EVM
Network	Polygon
Language	Solidity
Methodology	Link
Website	https://www.medievalempires.com/
Changelog	25.10.2022 - Initial Review 14.11.2022 - 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 Medieval Empires (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/moon-gaming/mee-governance-token/tree/b8540e7b88bd8877398a599e5d62c9294a981b0e
Commit	b8540e7b88bd8877398a599e5d62c9294a981b0e
Docs/Whitepaper	Whitepaper
Docs/Functional	Whitepaper
Docs/Technical	Technical description
Contracts	File: ./contracts/token-distribution/SaleRounds.sol SHA3: d6411de43d4fba2026712f225ac0f3f14f41edd0075bf9a0418f36e3365cb392 File: ./contracts/token-distribution/TokenDistribution.sol SHA3: 9a0b5a3e8d9ddb954b4c38592f4080d9539d30a9db824edb8f07417681fdfd2f File: ./contracts/token/GovernanceToken.sol SHA3: ad832d958984e8f256c0f667a6f10cc602a0595a46a297b2bd702b116e94f708 File: ./contracts/utils/GameOwner.sol SHA3: c7a2c3c25bfa0032474538c359cf7e99396ddad979391247df87c323205a7542

Second review scope

Repository	https://github.com/moon-gaming/mee-governance-token
Commit	d941f0f37baf5890ac07769a016b7d0a4eb911ec
Docs/Whitepaper	Whitepaper
Docs/Functional	Whitepaper
Docs/Technical	Technical description
Contracts	File: ./contracts/token-distribution/SaleRounds.sol SHA3: 5da6f7aee4e2655caae6e31cf426ed7aacb1e84ade569e2001745335b87bceba

	File: ./contracts/token-distribution/TokenDistribution.sol SHA3: e78cc6ea3eb8811480d961d80e34bb323608dbff5031e7fad3d74e04ce1c5f56 File: ./contracts/token/GovernanceToken.sol SHA3: 77fa5ec9917d71bf37d97c04195e0e28235be6b3220682ab7ad63fcca7f634d5 File: ./contracts/utils/GameOwner.sol SHA3: b17feeb4041115d3126ff88f80470a35e3e1510bae84c070e1427967d000a901

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

- Functional documentation is good.
- Technical documentation has not been updated with the new changes.

Code quality

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

- The code does not follow the official Solidity style guidelines.
- The development environment is configured.

Test coverage

Test coverage of the project is **74.36%** (branch coverage).

- Deployment and basic user interactions are covered with tests.
- Some negative cases are not covered.
- Some functions are not tested.
- Interactions by several users are not tested thoroughly.

Security score

As a result of the audit, the code contains **1** high, **1** medium and **3** low severity issues. The security score is **4** 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: **5.8**.

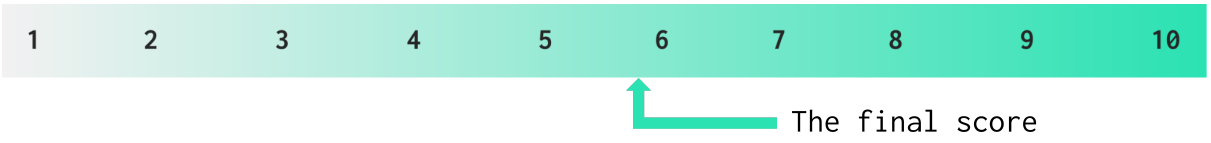

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
21 October 2022	13	1	3	3
14 November 2022	3	1	1	0

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.	Passed
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.	Not Relevant
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.	Not Relevant
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.	Failed
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.	Failed
Stable Imports	**Custom**	The code should not reference draft contracts, which may be changed in the future.	Passed

System Overview

Medieval Empire is a play to earn game offering a governance tokens sale with the following contracts:

- *GovernanceToken* – simple ERC-20 token with a max supply fixed at deployment.
  It has the following attributes:
  - Name: AoE Governance Token
  - Symbol: MEE
  - Decimals: 18
  - Max supply: 3b tokens.
- *SaleRounds* – a contract that holds the logic of the token sales for each round.
- *TokenDistribution* – a utility contract about round types.
- *GameOwner* - a contract that handles the gameOwner role.

Privileged roles

- The game owner can add or delete addresses to the distribution lists for *private* and *seed* rounds. This is necessary to handle off-chain management.
- Vesting has to be started by the Game Owner.

Risks

- The technical documentation says: "*if you already joined a reservation, you cannot join any other rounds and you are going to get an error like 'User has already registered for a different round'*". This verification is done when the user tries to join the reservation but not when the game owner adds the user to the whitelists. Therefore, a user can be on the whitelist for multiple rounds.
- *advisor, exchanges, play and earn*, *public*, *team*, *treasury* and *social* rounds tokens allocation are done to one address (for each round). This means that the members of the team can not claim their tokens by themself. The owner of the *teamWalletAddress* is then responsible for sending the tokens to the members of the team. This can create conflicts in the case of a member of the team leaving the project. The same applies for advisors.
- The *public* sale is not in the scope of this audit. Allocated tokens are sent to an external address, but no verification can be made about what happens then.
- **The project owners indirectly or directly control the entirety of the token supply and vesting.**

Findings

■■■■ Critical

1. Requirements Violation

The vesting and cliff periods for several rounds (*SEED*, *PRIVATE*, *PUBLIC*, *ADVISOR*, *EXCHANGES* and *PLAYANDEARN*) are in days when they should be in months, according to the documentation.

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

Recommendation: The code should not violate requirements provided by the Customer.

Status: Fixed (d941f0f37baf5890ac07769a016b7d0a4eb911ec)

2. Requirements Violation

During the seed and the private rounds, users can reserve and claim tokens without paying.

Path: ./contracts/SaleRounds.sol

Recommendation: The code should not violate requirements provided by the Customer.

Status: Fixed (d941f0f37baf5890ac07769a016b7d0a4eb911ec)

3. Requirements Violation, Funds Lock

The token allocation for *ADVISOR* is not implemented. They can neither reserve nor claim their tokens.

Path: ./contracts/SaleRounds.sol

Recommendation: The code should not violate requirements provided by the Customer.

Status: Fixed (d941f0f37baf5890ac07769a016b7d0a4eb911ec)

■■■ High

1. Highly Permissive Role Access

The game owner can set the cliff time for any round. This should not be possible as the cliff is determined in the whitepaper and should not be modifiable.

Path: ./contracts/SaleRounds.sol : function setCliffTime()

Recommendation: Remove mentioned functionality from the contract.

Status: Fixed (d941f0f37baf5890ac07769a016b7d0a4eb911ec)

2. Requirements Violation

TREASURY token distribution cliff is set to 4 months when the documentation sets it to 2 months.

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

Recommendation: The code should not violate requirements provided by the Customer.

Status: Fixed (d941f0f37baf5890ac07769a016b7d0a4eb911ec)

3. **Requirements Violation**

   EXCHANGES token distribution cliff is set to 3 months in the whitepaper. The distribution in the constructor follows this requirement, but tokens are minted without any checks breaking this rule. Comments in the code says "*NO VESTING TIME SO DIRECT MINTING*"

   Path: ./contracts/SaleRounds.sol : constructor(), function initialReserveAndMint()

   Recommendation: The code should not violate requirements provided by the Customer.

   Status: Fixed (d941f0f37baf5890ac07769a016b7d0a4eb911ec)

4. **Requirements Violation**

   The vesting granularity for several rounds (*SEED*, *PRIVATE*, *PUBLIC*, *PLAYANDEARN*, *EXCHANGES* and *TREASURY*) are in months when they should be in days, according to the documentation.

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

   Recommendation: The code should not violate requirements provided by the Customer.

   Status: New

Medium

1. **Missing Events Emit on Changing Important Values**

   It is recommended to emit events after changing important values. This will make it easy for everyone to notice such changes.

   Paths: **./**contracts/GameOwner.sol : function setGameOwnerAddress()

   ./contracts/SaleRounds.sol : function reserveTokens(), setCliffTime()

   Recommendation: Implement event emits after changing contract values.

   Status: Fixed (d941f0f37baf5890ac07769a016b7d0a4eb911ec)

2. **Documentation Contradiction**

   The project should be consistent and contain no self-contradictions.

   Technical documentation has not been updated after implementing modifications in the code.

   Path: Technical Documentation

Recommendation: Update technical documentation to follow the new implementation.

Status: New

Low

1. Requirements Noncompliance

Max supply is 3b tokens according to the whitepaper, 8b according to the technical documentation. The code implements 3b.

Path: ./contracts/SaleRounds.sol

Recommendation: Modify the technical documentation to align with the whitepaper and the code.

Status: Fixed (d941f0f37baf5890ac07769a016b7d0a4eb911ec)

2. Requirements Noncompliance

The technical documentation says: "*Multiple rounds cannot be active at the same time, only 1 round is allowed at a time.*" But if rounds can be set as active, there is no option to set them to inactive when the round has finished.

Path: ./contracts/SaleRounds.sol

Recommendation: As rounds have to remain active for tokens to be claimable after vesting period, the documentation should be changed. Another option is to allow tokens to be claimed when the distribution has started, even if the round is inactive. In this case, there should be an option to set the round as inactive and a check that no rounds are active before activating a new round.

Status: Fixed (d941f0f37baf5890ac07769a016b7d0a4eb911ec)

3. Floating Pragma

The project uses floating pragmas ^0.8.17.

Paths: ./contracts/GovernanceToken.sol

./contracts/SaleRounds.sol

./contracts/TokenDistribution.sol

./contracts/GameOwner.sol

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

Status: Fixed (d941f0f37baf5890ac07769a016b7d0a4eb911ec)

4. Functions that Can Be Declared External

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

Paths: **./**contracts/SaleRounds.sol : function initialReserveAndMint(), setActiveRound(), addAddressForDistribution(), deleteAddressForDistribution(), getAddressList(), reserveTokens(), mintTokensForPublic(), claimTokens(), getTotalClaimedForAllRounds(), getTotalRemainingForAllRounds(), getTotalRemainingForSpecificRound(), getTotalPending(), setCliffTime(), getCliffTime()

./contracts/GameOwner.sol : function setGameOwnerAddress(), getGameOwnerAddress()

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

Status: Fixed (d941f0f37baf5890ac07769a016b7d0a4eb911ec)

5. **Missing Zero Address Validation**

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

Paths: ./contracts/GameOwner.sol : constructor, function setGameOwnerAddress()

./contracts/SaleRounds.sol : function initialReserveAndMint()

Recommendation: Implement zero address checks.

Status: Fixed (d941f0f37baf5890ac07769a016b7d0a4eb911ec)

6. **Assert Violation**

Properly functioning code should never reach a failing assert statement.

Path: ./contracts/SaleRounds.sol : function initialReserveAndMint()

Recommendation: Add a check for the correct walletAddresses array.

Status: Fixed (d941f0f37baf5890ac07769a016b7d0a4eb911ec)

7. **Unindexed Events**

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

Path: ./contracts/SaleRounds.sol : events : ReservEvent, ClaimEvent.

Recommendation: Use the “indexed” keyword for the event parameters.

Status: Fixed (d941f0f37baf5890ac07769a016b7d0a4eb911ec)

8. **Boolean Equality**

Boolean constants can be used directly and do not need to be compared to true or false.

Path: ./contracts/SaleRounds.sol : function setActiveRound(), isEligibleToReserveToken(), isRoundActive()

Recommendation: Remove boolean equality.

Status: Fixed (d941f0f37baf5890ac07769a016b7d0a4eb911ec)

9. Misleading Error Messages

In function *reserveTokensInternal*, the require error message “total remaining seed amount is not enough” may apply for other rounds than seed.

Path: ./contracts/SaleRounds.sol : function getBalanceToRelease()

Recommendation: Refactor the message in require conditions to fit code behavior.

Status: Fixed (d941f0f37baf5890ac07769a016b7d0a4eb911ec)

10. Redundant Require Statement

In function *getBalanceToRelease* require statement (unClaimedBalance >= 0) is redundant due to uint being always >= 0.

This can lead to higher Gas taxes.

Path: ./contracts/SaleRounds.sol : function getBalanceToRelease()

Recommendation: Remove redundant code.

Status: Fixed (d941f0f37baf5890ac07769a016b7d0a4eb911ec)

11. Unused Variable

The variable MAX_UINT256 is never used.

Path: ./contracts/GovernanceToken.sol

Recommendation: Remove unused variable.

Status: Fixed (d941f0f37baf5890ac07769a016b7d0a4eb911ec)

12. Unused Function

The functions created but not used in the project should be deleted. This will make a more Gas efficient contract.

Path: ./contracts/GameOwner.sol : function isGameOwnerAddress()

Recommendation: Remove unused function or use it in the modifier onlyGameOwner() which uses the same code.

Status: Fixed (d941f0f37baf5890ac07769a016b7d0a4eb911ec)

13. Style Guide Violation

Inside each contract, use the following order:

1. Type declarations
2. State variables
3. Events
4. Modifiers
5. Functions

Functions should be grouped according to their visibility and ordered:

1. constructor
2. receive function (if exists)
3. fallback function (if exists)
4. external
5. public
6. internal
7. private

Within a grouping, place the view and pure functions last.

Paths: ./contracts/SaleRounds.sol

./contracts/GovernanceToken.sol

Recommendation: Follow the official Solidity guidelines.

Status: New

14. Unused Function

The function *mintTokenForPublicSale* is private and is never called. This function is not needed anymore as tokens for public sale are minted in *initialReserveAndMint*.

Path: ./contracts/SaleRounds.sol : function mintTokenForPublicSale()

Recommendation: Remove unused function.

Status: New

15. Duplicate Code

The modifiers *isInvestRound* and *claimableRound* are duplicated.

The modifier *isEligibleToReserveToken* is a combination of *isInvestRound* and *onlyGameOwner*. As it is used with *isInvestRound*, it is redundant and should be replaced by *onlyGameOwner*.

Path: ./contracts/SaleRounds.sol : isEligibleToReserveToken, isInvestRound, claimableRound.

Recommendation: Remove duplicate code.

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.