



SOLIDProof

Bring trust into your projects

**Blockchain Security | Smart Contract Audits | KYC
Development | Marketing**

MADE IN GERMANY

Throne

AUDIT

SECURITY ASSESSMENT

12. August, 2023

FOR



SolidProof_io



@solidproof_io

Introduction	4
Disclaimer	4
Project Overview	5
Summary	5
Social Medias	5
Audit Summary	6
File Overview	7
Imported packages	8
Audit Information	9
Vulnerability & Risk Level	9
Auditing Strategy and Techniques Applied	10
Methodology	10
Overall Security	11
Upgradeability	11
Ownership	12
Ownership Privileges	13
Minting tokens	13
Burning tokens	15
Blacklist addresses	16
Fees and Tax	17
Lock User Funds	18
Components	19
Exposed Functions	19
StateVariables	19
Capabilities	20
Inheritance Graph	21
Centralization Privileges	22
Audit Results	24
Critical issues	24
High issues	24



Medium issues	24
Low issues	25
Informational issues	26



Introduction

[SolidProof.io](#) is a brand of the officially registered company MAKE Network GmbH, based in Germany. We're mainly focused on Blockchain Security such as Smart Contract Audits and KYC verification for project teams.

Solidproof.io assess potential security issues in the smart contracts implementations, review for potential inconsistencies between the code base and the whitepaper/documentation, and provide suggestions for improvement.

Disclaimer

[SolidProof.io](#) reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team. SolidProof.io do not cover testing or auditing the integration with external contract or services (such as Unicrypt, Uniswap, PancakeSwap etc'...)

SolidProof.io Audits do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technology proprietors. SolidProof Audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

SolidProof.io Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology. Blockchain technology and cryptographic assets present a high level of ongoing risk. SolidProof's position is that each company and individual are responsible for their own due diligence and continuous security. SolidProof in no way claims any guarantee of the security or functionality of the technology we agree to analyze.

Project Overview

Summary

Project Name	Throne
Website	https://www.throne.exchange/
About the project	A Majestic Native DEX Reigning over the BASE Ecosystem. Fortified by the Community, Empowered by V3 Protocol.
Chain	Base
Language	Solidity
Codebase Link	https://basescan.org/token/0x798aCF1BD6E556F0C3cd72e77b3d169D26a28ab5#code
Commit	N/A
Unit Tests	Not Provided

Social Medias

Telegram	https://t.me/ThroneDEX
Twitter	https://twitter.com/ThroneDEX
Facebook	N/A
Instagram	N/A
Github	https://github.com/ThroneDEX
Reddit	N/A
Medium	N/A
Discord	https://discord.gg/d752m7dmeH
Youtube	N/A
TikTok	N/A
LinkedIn	N/A



Audit Summary

Version	Delivery Date	Changelog
v1.0	12. August 2023	<ul style="list-style-type: none"> • Layout Project • Automated- /Manual-Security Testing • Summary

Note - The following audit report presents a comprehensive security analysis of the smart contract utilized in the project. This analysis did not include functional testing (or unit testing) of the contract/s logic. We cannot guarantee 100% logical correctness of the contract as it was not functionally tested by us.





File Overview

The Team provided us with the files that should be tested in the security assessment. This audit covered the following files listed below with an SHA-1 Hash.

File Name	SHA-1 Hash
contracts/interfaces/IOATH.sol	044ea29e13ed92d66be4f2f34ab984cfd9de6838
contracts/OATH.sol	cd01c33fa80c691622046db5429139e1da8b9480

Please note: Files with a different hash value than in this table have been modified after the security check, either intentionally or unintentionally. A different hash value may (but need not) be an indication of a changed state or potential vulnerability that was not the subject of this scan.

Imported packages

Used code from other Frameworks/Smart Contracts (direct imports).

Dependency / Import Path	Count
@openzeppelin/contracts/access/Ownable.sol	1
@openzeppelin/contracts/token/ERC20/ERC20.sol	1
@openzeppelin/contracts/token/ERC20/IERC20.sol	1
@openzeppelin/contracts/utils/math/Math.sol	1
@openzeppelin/contracts/utils/math/SafeMath.sol	1
contracts/interfaces/IOATH.sol	1

Note for Investors: We only audited contracts mentioned in the scope above. All contracts related to the project apart from that are not a part of the audit, and we cannot comment on its security and are not responsible for it in any way

Audit Information

Vulnerability & Risk Level

Risk represents the probability that a certain source threat will exploit vulnerability and the impact of that event on the organization or system. The risk Level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 - 10	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
High	7 - 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon as possible.
Medium	4 - 6.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 - 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 - 1.9	A vulnerability that have informational character but is not effecting any of the code.	An observation that does not determine a level of risk

Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to check the repository for security-related issues, code quality, and compliance with specifications and best practices. To this end, our team of experienced pen-testers and smart contract developers reviewed the code line by line and documented any issues discovered.

We check every file manually. We use automated tools only so that they help us achieve faster and better results.

Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
 - a. Reviewing the specifications, sources, and instructions provided to SolidProof to ensure we understand the size, scope, and functionality of the smart contract.
 - b. Manual review of the code, i.e., reading the source code line by line to identify potential vulnerabilities.
 - c. Comparison to the specification, i.e., verifying that the code does what is described in the specifications, sources, and instructions provided to SolidProof.
2. Testing and automated analysis that includes the following:
 - a. Test coverage analysis determines whether test cases cover code and how much code is executed when those test cases are executed.
 - b. Symbolic execution, which is analysing a program to determine what inputs cause each part of a program to execute.
3. Review best practices, i.e., review smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on best practices, recommendations, and research from industry and academia.
4. Concrete, itemized and actionable recommendations to help you secure your smart contracts.



Overall Security

Upgradeability

Contract is not an upgradeable



Deployer cannot update the contract with new functionalities

Description

The contract is not an upgradeable contract. The deployer is not able to change or add any functionalities to the contract after deploying.

Comment

N/A



Ownership

The ownership is not renounced

✗ The owner is not renounce

Description

The owner has not renounced the ownership that means that the owner retains control over the contract's operations, including the ability to execute functions that may impact the contract's users or stakeholders. This can lead to several potential issues, including:

- Centralizations
- The owner has significant control over contract's operations

Example

We assume that you have funds in the contract and it has been audited by any security audit firm. Now the audit has passed. After that, the deployer can upgrade the contract to allow him to transfer the funds you purchased without any approval from you. This has the consequence that your funds can be taken by the creator.

Comment

At the date of 12. August 2023 where the audit was created, the owner address was 0x6d14bdae774042d0ad862b8d1f23ce9f35036844.

Note - If the contract is not deployed, consider the ownership not renounced. Moreover, if there are no ownership functionalities, ownership is automatically considered renounced.



Ownership Privileges

These functions can be dangerous. Please note that abuse can lead to financial loss. We have a guide where you can learn more about these Functions.

Minting tokens

Minting tokens refer to the process of creating new tokens in a cryptocurrency or blockchain network. This process is typically performed by the project's owner or designated authority, who has the ability to add new tokens to the network's total supply.

Contract owner cannot mint new tokens

✓ The owner cannot mint new tokens

Description

The owner is not able to mint new tokens once the contract is deployed.

Comment

The owner cannot mint new tokens directly. In the "emitAllocations" function the "_mint" function is executed to mint "masterV2Share" and "masterV3Share" to the contract address and also the "treasuryShare" to the "treasuryAddress".

File, Line/s: OATH, L192-L194

Codebase:

```

156     function emitAllocations() public {
157         uint256 circulatingSupply = totalSupply();
158         uint256 currentBlockTimestamp = _currentBlockTimestamp();
159
160         uint256 _lastEmissionTime = lastEmissionTime; // gas saving
161         uint256 _maxSupply = elasticMaxSupply; // gas saving
162
163         // if already up to date or not started
164         if (currentBlockTimestamp <= _lastEmissionTime || _lastEmissionTime == 0) {
165             return;
166         }
167
168         // if max supply is already reached or emissions deactivated
169         if (_maxSupply <= circulatingSupply || emissionRate == 0) {
170             lastEmissionTime = currentBlockTimestamp;
171             return;
172         }
173
174         uint256 newEmissions = currentBlockTimestamp.sub(_lastEmissionTime).mul(emissionRate);
175
176         // cap new emissions if exceeding max supply
177         if (_maxSupply < circulatingSupply.add(newEmissions)) {
178             newEmissions = _maxSupply.sub(circulatingSupply);
179         }
180
181         // calculate master and treasury shares from new emissions
182         uint256 masterV2Share = newEmissions.mul(masterV2Allocation).div(ALLOCATION_PRECISION);
183
184         uint256 masterV3Share = newEmissions.mul(masterV3Allocation).div(ALLOCATION_PRECISION);
185
186         // sub to avoid rounding errors
187         uint256 treasuryShare = newEmissions.sub(masterV2Share).sub(masterV3Share);
188
189         lastEmissionTime = currentBlockTimestamp;
190
191         // add master shares to its claimable reserve
192         masterV2Reserve = masterV2Reserve.add(masterV2Share);
193         masterV3Reserve = masterV3Reserve.add(masterV3Share);
194         // mint shares
195         mint(address(this), masterV2Share);
196         mint(address(this), masterV3Share);
197         mint(treasuryAddress, treasuryShare);
198
199         emit AllocationsDistributed(masterV2Share, masterV3Share, treasuryShare);
200     }

```



Burning tokens

Burning tokens is the process of permanently destroying a certain number of tokens, reducing the total supply of a cryptocurrency or token. This is usually done to increase the value of the remaining tokens, as the reduced supply can create scarcity and potentially drive up demand.

Contract owner cannot burn tokens

 **The owner cannot burn tokens**

Description	The owner is not able burn tokens without any allowances.
Comment	N/A



Blacklist addresses

Blacklisting addresses in smart contracts is the process of adding a certain address to a blacklist, effectively preventing them from accessing or participating in certain functionalities or transactions within the contract. This can be useful in preventing fraudulent or malicious activities, such as hacking attempts or money laundering.

Contract owner cannot blacklist addresses



The owner cannot blacklist addresses

Description

The owner is not able blacklist addresses to lock funds.

Comment

N/A



Fees and Tax

In some smart contracts, the owner or creator of the contract can set fees for certain actions or operations within the contract. These fees can be used to cover the cost of running the contract, such as paying for gas fees or compensating the contract's owner for their time and effort in developing and maintaining the contract.

Contract owner cannot set fees more than 25%



The owner cannot levy unfair taxes

Description	The owner is not able to set the fees above 25%
Comment	N/A

Lock User Funds

In a smart contract, locking refers to the process of restricting access to certain tokens or assets for a specified period of time. When tokens or assets are locked in a smart contract, they cannot be transferred or used until the lock-up period has expired or certain conditions have been met.

Owner cannot lock the contract



The owner cannot lock the contract

Description

The owner is not able to lock the contract by any functions or updating any variables.

Comment

The owner can set the "emissionRate" to 0. That causes that the "emitAllocations" cannot be called anymore.

File, Line/s: OATH, L326-L334, L156

Codebase based on the comment above:

```
326     function updateEmissionRate(uint256 emissionRate_↑) external onlyOwner {
327         require(emissionRate_↑ <= MAX_EMISSION_RATE, "OATH:updateEmissionRate: can't exceed maximum");
328
329         // apply emissions before changes
330         emitAllocations();
331
332         emit UpdateEmissionRate(emissionRate, emissionRate_↑);
333         emissionRate = emissionRate_↑;
334     }
```

```
156     function emitAllocations() public {
157         uint256 circulatingSupply = totalSupply();
158         uint256 currentBlockTimestamp = _currentBlockTimestamp();
159
160         uint256 _lastEmissionTime = lastEmissionTime; // gas saving
161         uint256 _maxSupply = elasticMaxSupply; // gas saving
162
163         // if already up to date or not started
164         if (currentBlockTimestamp <= _lastEmissionTime || _lastEmissionTime == 0) {
165             return;
166         }
167
168         // if max supply is already reached or emissions deactivated
169         if (_maxSupply <= circulatingSupply || emissionRate == 0) {
170             lastEmissionTime = currentBlockTimestamp;
171             return;
172         }
173
174         uint256 newEmissions = currentBlockTimestamp.sub(_lastEmissionTime).mul(emissionRate);
175
176         // cap new emissions if exceeding max supply
177         if (_maxSupply < circulatingSupply.add(newEmissions)) {
178             newEmissions = _maxSupply.sub(circulatingSupply);
179         }
180
181         // calculate master and treasury shares from new emissions
182         uint256 masterV2Share = newEmissions.mul(masterV2Allocation).div(ALLOCATION_PRECISION);
183
184         uint256 masterV3Share = newEmissions.mul(masterV3Allocation).div(ALLOCATION_PRECISION);
185
186         // sub to avoid rounding errors
187         uint256 treasuryShare = newEmissions.sub(masterV2Share).sub(masterV3Share);
188
189         lastEmissionTime = currentBlockTimestamp;
190
191         // add master shares to its claimable reserve
192         masterV2Reserve = masterV2Reserve.add(masterV2Share);
193         masterV3Reserve = masterV3Reserve.add(masterV3Share);
194         // mint shares
195         mint(address(this), masterV2Share);
196         mint(address(this), masterV3Share);
197         mint(treasuryAddress, treasuryShare);
198
199         emit AllocationsDistributed(masterV2Share, masterV3Share, treasuryShare);
200     }
```

External/Public functions

External/public functions are functions that can be called from outside of a contract, i.e., they can be accessed by other contracts or external accounts on the blockchain. These functions are specified using the function declaration's external or public visibility modifier.

State variables

State variables are variables that are stored on the blockchain as part of the contract's state. They are declared at the contract level and can be accessed and modified by any function within the contract. State variables can be defined with a visibility modifier, such as public, private, or internal, which determines the access level of the variable.

Components

 Contracts	 Libraries	 Interfaces	 Abstract
1	0	1	0


Exposed Functions

This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.

 Public	 Payable
23	0





External	Internal	Private	Pure	View
19	21	0	0	7

StateVariables

Total	 Public
18	18



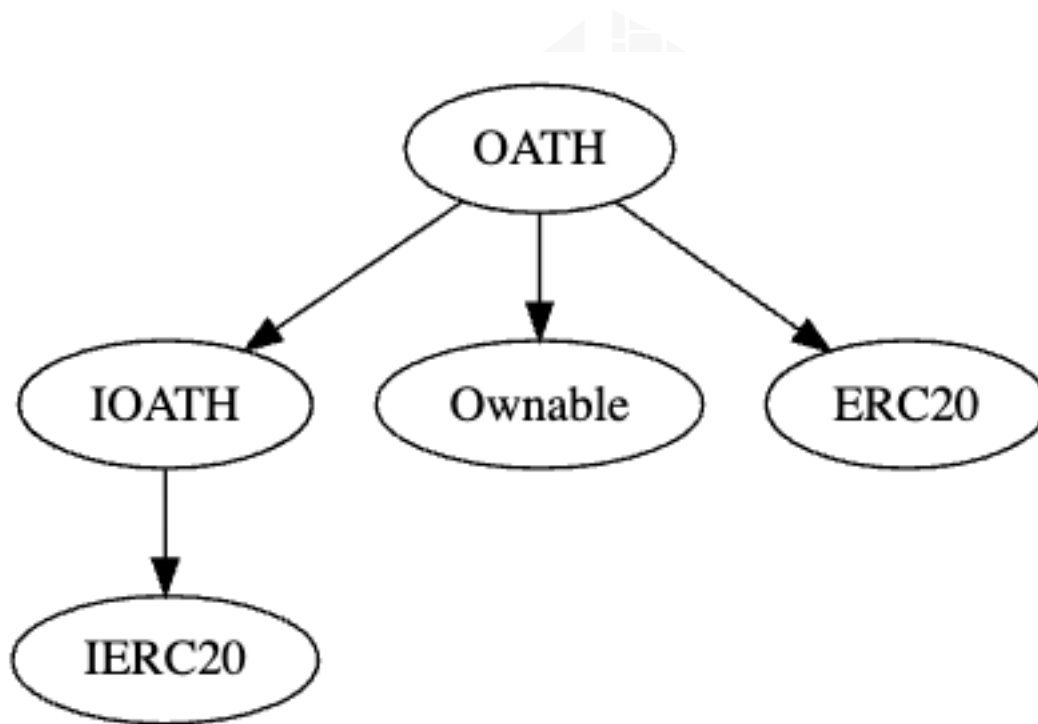
Capabilities

Solidity Versions observed	 Experimenta l Features	 Can Receive Funds	 Uses Assembl y	 Has Destroyable Contracts
<code>^0.8.10</code>				



Inheritance Graph

An inheritance graph is a graphical representation of the inheritance hierarchy among contracts. In object-oriented programming, inheritance is a mechanism that allows one class (or contract, in the case of Solidity) to inherit properties and methods from another class. It shows the relationships between different contracts and how they are related to each other through inheritance.



Centralization Privileges

Centralization can arise when one or more parties have privileged access or control over the contract's functionality, data, or decision-making. This can occur, for example, if the contract is controlled by a single entity or if certain participants have special permissions or abilities that others do not.

In the project, there are authorities that have access to the following functions:

File	Privileges
1. OATH.sol	<ul style="list-style-type: none"> • OnlyOwner <ul style="list-style-type: none"> • updateMasterV2Addresses • updateMasterV3Addresses • initializeEmissionStart • updateAllocations • updateEmissionStart • updateMaxSupply • updateMaxWallet • updateTreasuryAddress • setExcludeMaxWallet • disableMaxWallet • onlyMasterV2 <ul style="list-style-type: none"> • claimMasterV2Rewards • onlyMasterV3 <ul style="list-style-type: none"> • claimMasterV3Rewards

Recommendations

To avoid potential hacking risks, it is advisable for the client to manage the private key of the privileged account with care. Additionally, we recommend enhancing the security practices of centralized privileges or roles in the protocol through a decentralized mechanism or smart-contract-based accounts, such as multi-signature wallets.

Here are some suggestions of what the client can do:

- Consider using multi-signature wallets: Multi-signature wallets require multiple parties to sign off on a transaction before it can be executed, providing an extra layer of security e.g. Gnosis Safe
- Use of a timelock at least with a latency of e.g. 48-72 hours for awareness of privileged operations
- Introduce a DAO/Governance/Voting module to increase transparency and user involvement



- Consider Renouncing the ownership so that the owner cannot modify any state variables of the contract anymore. Make sure to set up everything before renouncing.



Audit Results

Critical issues

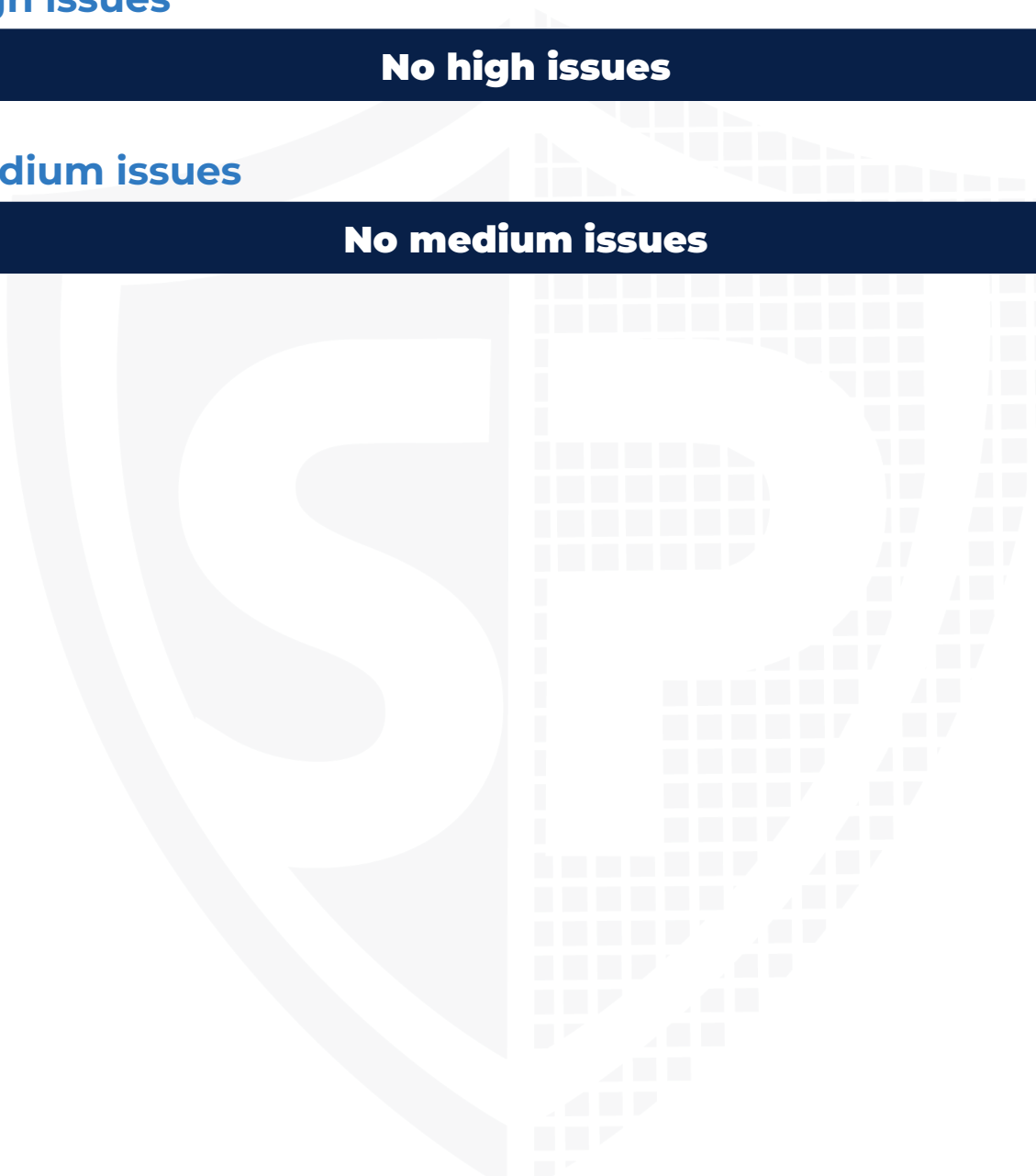
No critical issues

High issues

No high issues

Medium issues

No medium issues



Low issues

#1 | Updating masterV2/masterV3 to an EOA

File	Severity	Location	Status
Main	Low	L264, L278	Open

Description - The owner can set the masterV2/masterV3 to an EOA. It is recommended to check whether the new address is a contract. The owner can set his address to the "masterV2Address"/"masterV3Address" to claim the "masterV2Reserve"/"masterV3Reserve".

Remediation - Additionally, it is recommended to include the old addresses back to the "isExcludedFromMaxWallet".

#2 | Wrong event emit

File	Severity	Location	Status
Main	Low	L245	Open

Description - While claiming the masterV3Rewards, the "ClaimMasterV2Rewards" will be emitted.

Remediation - Emit the "claimMasterV3Rewards".

#3 | Centralized privileges

File	Severity	Location	Status
Main	Low	L264, L278, L367, L306	Open

Description - The owner can set the master addresses to an EOA address. It is possible that he can set the address to his address to get the allocations. Anyway, this is also possible by setting the master shares to an amount of 0. That causes the treasuryAddress, which the owner can also set, will get 100% of the emissions.

Informational issues

#1 | Dead address check

File	Severity	Location	Status
Main	Informational	L367	Open

Description - Check the dead address for the “treasuryAddress_” address. Otherwise, the treasury shares will be minted to the dead address.

Remediation - Add a dead address “require” statement before setting the “treasuryAddress”.

#2 | Floating Pragma

File	Severity	Location	Status
Main	Informational	L9	Open

Description - The contracts should be deployed with the same compiler version and flag that they have been tested thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using other versions.

Remediation - Get rid of the “^” sign to use the version of 0.8.10 explicitly.

#3 | Check zero value

File	Severity	Location	Status
Main	Informational	L189, L192	Open

Description - The variable “masterV2Allocation” can be set to 0 value. That causes the minting will be executed in the “emitAllocations”. Analog to it is the “masterV3Allocation”. If both variables are set to 0, all new emissions will be transferred to the “treasuryAddress”.

Remediation - Verify that the variable is not 0 before calling the minting.

#4 | Duplication of code

File	Severity	Location	Status
Main	Informational	L207, L230	Open

Description - Instead of duplicating a function, you can split the function that contains the changes. In this case, the change is in the reserves (“masterV2Reserve”/“masterV3Reserve”) in the claiming functions. We

recommend moving this function into its function to save code and duplications. The more duplications, the more errors can happen.

#5 | SafeMath library is unnecessary

File	Severity	Location	Status
Main	Informational	L6	Open

Description - The SafeMath library is unnecessary in the contract because the pragma version is above 0.8.x. That version handles the overflow/underflow issues by default.

Remediation - If you are going to remove the SafeMath library, ensure to replace the SafeMath library functions with raw mathematical operations.



Legend for the Issue Status

Attribute or Symbol	Meaning
Open	The issue is not fixed by the project team.
Fixed	The issue is fixed by the project team.
Acknowledged(ACK)	The issue has been acknowledged or declared as part of business logic.





**Blockchain Security | Smart Contract Audits | KYC
Development | Marketing**

MADE IN GERMANY