SOLIDProof
Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC
Development | Marketing

MADE IN GERMANY

Manga Fi

AUDIT

SECURITY ASSESSMENT

29. August, 2023

FOR

SolidProof_io   @solidproof_io

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	Manga Fi
Website	www.mangafi.org
About the project	MangaFi is a DeFi protocol that grows the value of its collection of DeFi fan coin assets and generates real yield for its users. The core of MangaFi is a yield-bearing & fully-backed reserved currency called HONO. HONO is backed by an ETH vault that grows by accumulating revenue from real yield-generation resources.
Chain	Ethereum Network(eth)
Language	Solidity
Codebase	HONO:https://etherscan.io/address/0x8F5ac7F3cfeFfB6EB638A8aDd2d32661F82C03FD#code Stakinghono:https://etherscan.io/address/0xf39D0162e3d03fD95ea8355611cD331B3E068B18#code xHono:https://etherscan.io/address/0x74e4c18eb9d7390f6bf7bc477d94b51608366048#code LockPool:https://etherscan.io/address/0x0df24d02681cffbd740ec2cb8d719b1cb3c5c83c#code
Commit	N/A
Unit Tests	Not Provided

Social Medias

Telegram	N/A
Twitter	https://twitter.com/MangafiOfficial
Facebook	N/A
Instagram	https://www.instagram.com/mangafiofficial/
GitHub	N/A
Reddit	N/A
Medium	N/A
Discord	N/A
YouTube	N/A
TikTok	https://www.tiktok.com/@mangafiofficial
LinkedIn	N/A

Audit Summary

Version	Delivery Date	Change Log
v1.0	25. August 2023	· Layout Project
		· Automated/ Manual-Security Testing
		· Summary
v1.1	29. August 2023	· Reaudit

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

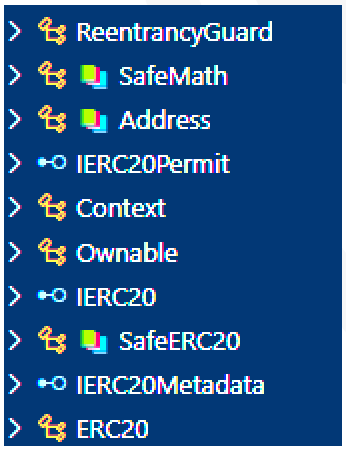
The Team provided us with the files that should be tested in the security assessment. This audit covered the following files listed below with an SHA-1 Hash.

File Name	SHA-1 Hash
contracts/LockPool.sol	8d1104e38b0115c380d8b72993b7b1d1f1bd0b8d
contracts/xHONO.sol	54bbc06f9c2f62582e09327fde3f6ccc8b25faf6
contracts/HONO.sol	cd11688d88e3ea54be2df0f3c463face07d816e6
contracts/Staking-hono.sol	cd65c59078ceaa6beee947119150e1e17ad95723

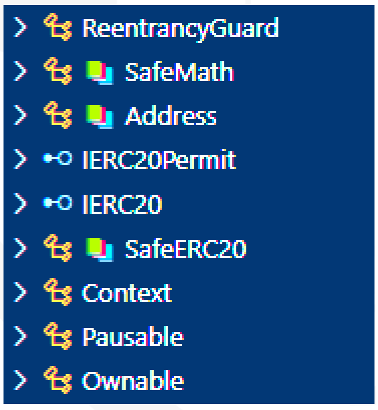
Please note: Files with a different hash value than in this table have been modified after the security check, either intentionally or unintentionally. A different hash value may (but need not) be an indication of a changed state or potential vulnerability that was not the subject of this scan.

Imported packages

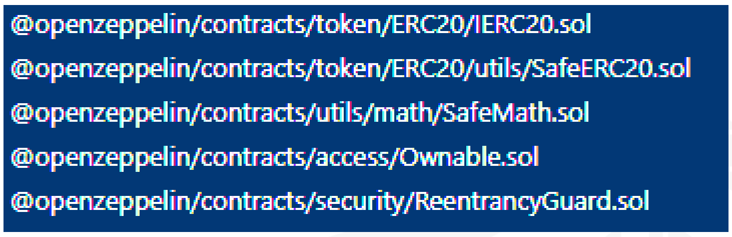
Used code from other Frameworks/Smart Contracts.



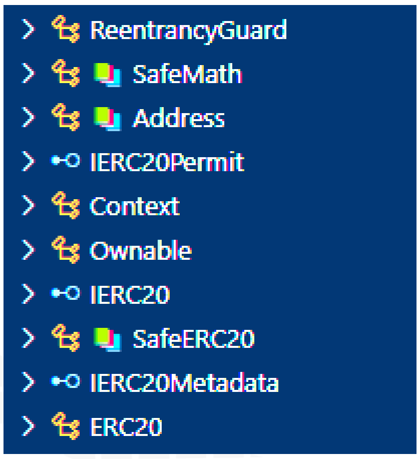
ReentrancyGuard
SafeMath
Address
IERC20Permit
Context
Ownable
IERC20
SafeERC20
IERC20Metadata
ERC20



ReentrancyGuard
SafeMath
Address
IERC20Permit
IERC20
SafeERC20
Context
Pausable
Ownable



@openzeppelin/contracts/token/ERC20/IERC20.sol
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol
@openzeppelin/contracts/utils/math/SafeMath.sol
@openzeppelin/contracts/access/Ownable.sol
@openzeppelin/contracts/security/ReentrancyGuard.sol



- ReentrancyGuard
- SafeMath
- Address
- IERC20Permit
- Context
- Ownable
- IERC20
- SafeERC20
- IERC20Metadata
- ERC20

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.

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'sstate. They are declared at the contract level and can be accessed and modified by any function within the contract. State variables can be needed within visibility modifier, such as public, private or internal, which determines the access level of the variable.

Components

📝Contracts	📚Libraries	🔍Interfaces	🎨Abstract
6	9	9	10

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
102	4

External	Internal	Private	Pure	View
62	266	12	45	66

StateVariables

Total	🌐Public
52	28

Capabilities

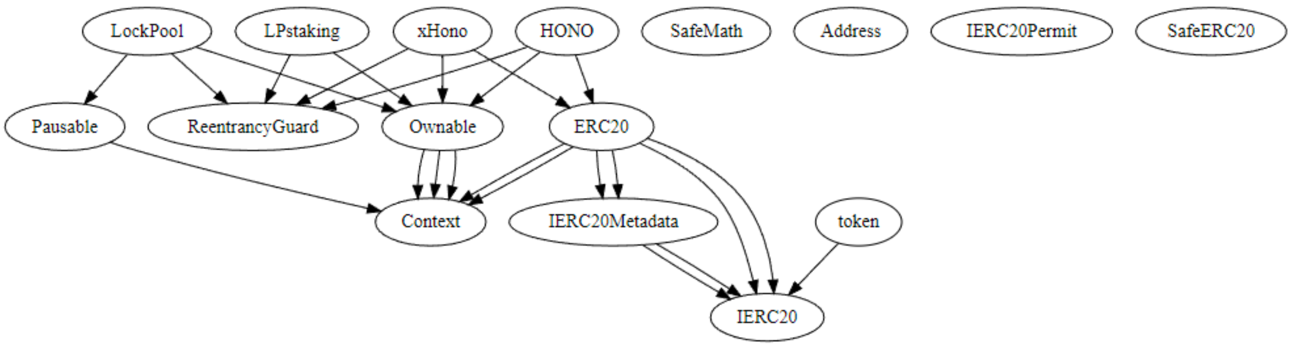
Solidity Versions observed	🧪 Experimental Features	💰 Can Receive Funds	🖥 Uses Assembly	💣 Has Destroyable Contracts
`^0.8.0` `^0.8.1`	`----------`	`Yes`	`Yes` (3 asm blocks)	`----------`

📤 Transfers ETH	⚡ Low-Level Calls	👥 Delegate Call	🧮 Uses Hash Functions	🔖 ECRecover	🌀 New/Create/Create2
`yes`		`yes`			

♻️ TryCatch	Σ Unchecked
	`yes`

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 methodsfrom another class. It shows the relationships between different contracts and how they are related to each other through inheritance.



Audit Information

Vulnerability & Risk Level

Risk represents the probability that a certain source threat will exploit the 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 aspossible.
Medium	4 – 6.9	A vulnerability that could affect the desired outcome of executingthe 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 andsmart 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 codeand 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 upgradable	✅ 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 ownership is not renounced**

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	N/A
Comment	N/A

Note – *The contract cannot be considered as renounced till it is not deployed or having some functionality that can change the state of the contract.*

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.

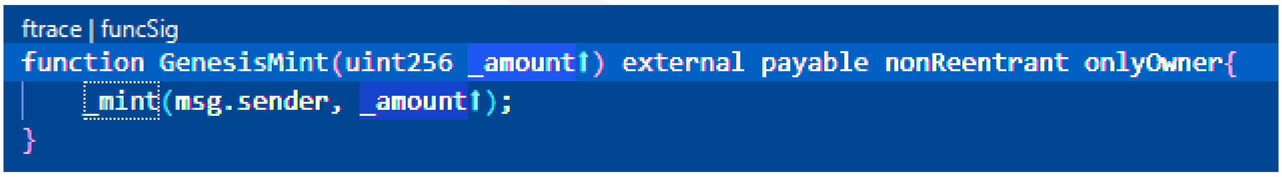
Minting tokens

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 can mint new tokens	❌ **The owner can mint new tokens.**
Description	Owners who have the ability to mint new tokens can reward themselves or other stakeholders, who can then sell the newly minted tokens on a cryptocurrency exchange to raise funds. However, there is a risk that the owner may abuse this power, leading to a decrease in trust and credibility in the project or platform. If stakeholders perceive that the owner is using their power to mint new tokens unfairly or without transparency, it can result in decreased demand for the token and a reduction in its value.
Comment	The owner can mint tokens after paying the eth and according to the amount of eth the tokens will be minted through the mint function and can receive his eth back by burn their own tokens, but this contract also contains one genesis mint functionality where the owner has the ability to mint unlimited amount of tokens. Also, In the xHono contract minter has the ability to mint unlimited number of tokens to any external account.

File/Line(s): L1198-1200

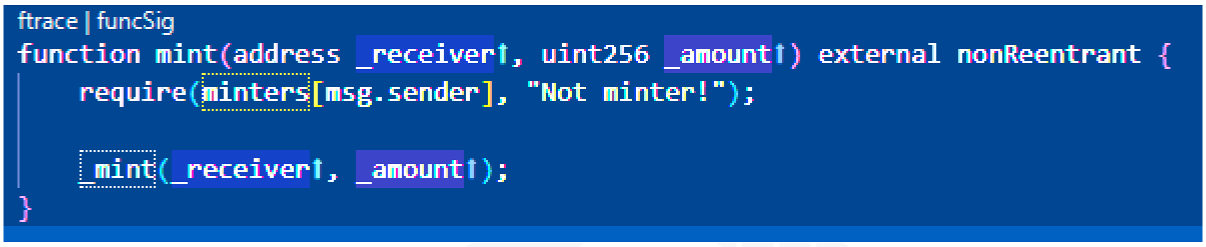
Codebase: HONO.sol



```
function GenesisMint(uint256 _amount↑) external payable nonReentrant onlyOwner{
    _mint(msg.sender, _amount↑);
}
```

File/Line(s): L1368-1372

Codebase: xHono.sol



```
ftrace | funcSig
function mint(address _receiver, uint256 _amount) external nonReentrant {
    require(minters[msg.sender], "Not minter!");

    _mint(_receiver, _amount);
}
```

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 indeveloping and maintaining the contract.

Contract owner cannot set fees more than 25%.	✅ **The owner cannot set fees more than 25%**
Description	The owner is not able to set the fees above 25%.
Comment	N/A

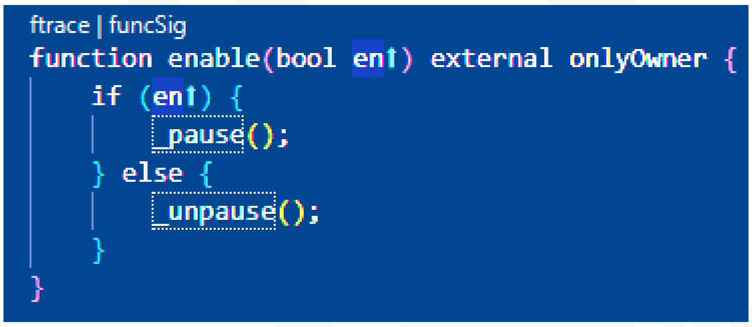
Lock User Funds

In a smart contract, locking refers to the process of restricting access to certain tokens or assets for a specified period of time. When token 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.

Contract owner can lock user funds.	❌ **The owner can lock user funds.**
Description	Locking the contract means that the owner is able to lockany funds of addresses that they are not able to transferbought tokens anymore.
Comment	This contract contains the 'pausable' functionality that can pause the locking and unlocking functions for unlimited period of time and because of this if the user has locked their tokens they won't be able to unlock them until the owner un-pause the token.

File/Line(s): L1049-1055

Codebase: LockPool.sol



ftrace | funcSig

```
function enable(bool ent) external onlyOwner {
    if (ent) {
        _pause();
    } else {
        _unpause();
    }
}
```

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
HONO.sol	➢ The owner can update any address as an arbitrager. ➢ The owner can add any external account as a minter. ➢ The owner can mint unlimited number of tokens. ➢ The owner, arbitrager and minter can redeem the amount of eth by burning of tokens.
xHono.sol	➢ The owner can add or remove the minters in the contract. ➢ The minter can mint an unlimited number of tokens to any external address.
LockPool.sol	➢ The owner can add any token address in the contract that will be used in the contract. ➢ The owner can update any address as DAO address for the contract. ➢ The owner can update any address to exchange pool address for the contract. ➢ The owner can pause and un-pause the locking and unlocking functionality in the contract for an unlimited period of time. ➢ The owner can update any number into the fixed rate for the receiver1 and fixed rate receiver2. ➢ The owner can update any external account as receiver 1 and receiver 2 in the contract. ➢ The owner can claim the stuck tokens to any external account. Also, the owner can claim the contract’s own tokens from the contract.
Staking-hono.sol	➢ The owner or lpEngine can add the user info for staking. ➢ The owner or lpEngine can withdraw the user’s token to his account from staking in this contract. ➢ The owner can add any external address as a stake

pool in this contract.

- The owner can update any arbitrary value as a referrer amount in the contract.
- The owner can add or remove any external address as a lpEngine in the contract.
- The owner can recover eth balance from the contract.
- The owner can claim the stuck tokens in the contract. Also, the owner can claim the contract's own tokens from the contract.

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 Result

#1 | Owner can mint unlimited number of tokens.

File	Severity	Location	Status
HONO.sol	Medium	L1198-1200	ACK
xHono.sol	Medium	L1368-1372	ACK

Description – The owner has the ability to mint unlimited number of tokens which can manipulate the supply of tokens. Also, In the xHono contract, minter can mint unlimited number of tokens to any external account including his own address.

Remediation – It is recommended that no tokens should be minted after the initial deployment of tokens.

Alleviation – Team is responsible for maintaining the price on the market to keep it pegged to treasury value as per document: https://docs.mangafi.org/hono-mechanics/chakra.

Hence the team is required to retain ownership of the contract.

#2 | The owner can change the lock token address.

File	Severity	Location	Status
LockPool.sol	Medium	L1037-1039	ACK

Description – The token which will be used as a lock token should not be changed because if it is done then the users who have locked the tokens won't be able to withdraw the token that they locked because the state variable "token" has been modified so, the transfer call will go to some other contract.

Alleviation – LockPool is mainly used for storing xHONO (governance) for voting emissions as per documentation: https://docs.mangafi.org/xhono-token/vote-and-claim.

The team requires access to pause the contract if necessary if rules need to be changed in the future and the contract needs to be redeployed.

#3 | The Owner can lock functions for an unlimited period.

File	Severity	Location	Status
LockPool.sol	Medium	L1049-1055	ACK

Description – The owner can pause the locking and unlocking functionality of the contract for an unlimited period of time as this can lock the user funds if the owner does not un-pause the token.

Remediation – It is recommended that the 'pausable' functionality should not be present in the contract and if it is there then there must be some threshold for the locking period. Also, The unlock functionality should not be paused anyway as this can lock the user's funds.

Alleviation – LockPool is mainly used for storing xHONO (governance) for voting emissions as per documentation: https://docs.mangafi.org/xhono-token/vote-and-claim.

The Team requires access to pause the contract if necessary if rules need to be changed in the future and the contract needs to be redeployed.

#4 | User cannot withdraw tokens.

File	Severity	Location	Status
Staking-hono.sol	Medium	L214-216	ACK

Description – The amount of tokens that staked by the user cannot be withdrawn by them, only the owner and LP address have the ability of tokens to withdraw which is not recommended.

Remediation – Add some functionality in the contract where the users can also have the ability to withdraw the staked tokens.

Audit comment – The project team confirmed that "instead of asking people to stake LP token, we auto stake for them when they add LP. hence the deposit() and withdraw() function is actually called from another contract that manages their LP" and that contract is not provided with us and we are not responsible for any security issues related to it.

Alleviation – User never send any token to this contract. They add LP in a different contract, which then call this contract to update user's stake. User reward will be send to them when they remove or add more LP.

#5 | Owner can drain locked tokens.

File	Severity	Location	Status
LockPool.sol	Medium	L1106-1108	ACK

Description – The owner has the ability to drain the locked tokens. The number of tokens is not subtracted from the total staked balance, and it is directly transferred to the user's wallets which is not recommended.

Remediation – Add the functionality where the amount of the staked tokens will be subtracted first then the rest of the amount will be sent to the user.

Alleviation – This is so that we can handle migration to a new code base if changes to the contract needs to be made.

#6 | Missing events arithmetic.

File	Severity	Location	Status
HONO.sol	Low	L1191-1193, L1194-1196	ACK
xHono.sol	Low	L1364-1366, L1368-1372	ACK
LockPool.sol	Low	L1037-1039, L1041-1043, L1045-1047, L1089-1092, L1093-1096	ACK
Staking-hono.sol	Low	L223-225, L227-230	ACK

Description – Emit all the critical parameter changes.

#7 | Missing state visibility.

File	Severity	Location	Status
HONO.sol	Low	L1189	ACK

Description – There is no visibility present for the state variable which is not recommended.

Remediation – Add ‘public’ or ‘private’ while initializing any variable in the contract.

#8 | Missing zero or dead address check.

File	Severity	Location	Status
HONO.sol	Low	L1191-1193, L1194-1196, L1235-1238	ACK
xHono.sol	Low	L1364-1366, L1368-1372	ACK
LockPool.sol	Low	L1037-1039, L1041-1043, L1045-1047, L1089-1092, L1093-	ACK

		1096	
Staking-hono.sol	Low	L214-216, L219-221, L227-229	ACK

Description – It is recommended to check that the address cannot be set to a zero or dead address.

#9 | The owner can claim the contract's own tokens.

File	Severity	Location	Status
LockPool.sol	Low	L1106-1108	ACK
Staking-hono.sol	Low	L241-244	ACK

Description – It is recommended that the contract's owner should not have the authority to claim the contract's own tokens of the contract.

Remediation – Add a 'require' check so that the owner should not be able to claim the contract's own tokens.

#10 | Missing zero check.

File	Severity	Location	Status
LockPool.sol	Low	L1057-1087	ACK

Description – Add a 'require' check that the value must be greater than zero.

#11 | Missing 'require' check.

File	Severity	Location	Status
HONO.sol	Low	L1227-1229	ACK

Description – The value which will be deleted must be greater than zero.

Remediation – Add a 'require' check that the value must be greater than zero otherwise the functionality will fail, and it will throw an error.

#12 | Unnecessary code.

File	Severity	Location	Status
Staking-hono.sol	Low	L80	ACK

Description – The msg.value is added to 'totalETHdistributed' and that value is not being used anywhere. I would recommend you remove irrelevant code to reduce the gas fees.

#13 | Incorrect naming convention.

File	Severity	Location	Status
Staking-hono.sol	Informational	L227-229	ACK

Description – Incorrect naming convention in the parameters.

Remediation – It is recommended that the variable and parameters name should be relevant and provide the correct information. For more details, please refer to this documentation (https://solidity.readthedocs.io/en/v0.4.25/style-guide.html#naming-conventions).

#14 | NatSpec Documentation missing.

File	Severity	Location	Status
HONO.sol	Informational	--	ACK
xHono.sol	Informational	--	ACK
LockPool.sol	Informational	--	ACK
Staking-hono.sol	Informational	--	ACK

Description – If you started to comment on your code, also comment on all other functions, variables, etc.

Note for Investor– The Team has acknowledged the issue and declared it as a part of the business logic which means that the issue that was mentioned by us is still in the code and needs to be looked at carefully before investment. Also, we are not responsible for any loss of funds caused because of the above issues that we have mentioned. Moreover, as the contract imports Uniswap functions then there is a possibility that the owner may be able to drain liquidity using the Uniswap Interface to Swap Contract balance for ETH.

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.

Solid Proofed

Blockchain Security | Smart Contract Audits | KYC Development | Marketing

MADE IN GERMANY