



SOLIDProof

Bring trust into your projects

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

MADE IN GERMANY

Unicow

AUDIT

SECURITY ASSESSMENT

21. September, 2023

FOR



SolidProof_io



@solidproof_io



Introduction	3
Disclaimer	3
Project Overview	4
Summary	4
Social Medias	4
Audit Summary	5
File Overview	6
Imported packages	6
Components	7
Exposed Functions	7
Capabilities	8
Inheritance Graph	9
Audit Information	10
Vulnerability & Risk Level	10
Auditing Strategy and Techniques Applied	11
Methodology	11
Overall Security	12
Upgradeability	12
Ownership	13
Ownership Privileges	14
Minting tokens	14
Burning tokens	15
Blacklist addresses	16
Fees and Tax	17
Lock User Funds	18
Centralization Privileges	21
Audit Results	23

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	Unicow
Website	https://unicow.org/
About the project	Unicow is the world's 1st, specially designed transferable passive income pool that gives extra rewards to pool owners. We introduced NFT to show all information of each pool including owner info, TVL, pool level, reward rates, and so on. Each NFT will present each reward pool.
Chain	Basescan
Language	Solidity
Codebase	NftManager: https://basescan.org/address/0x51403ed41c73174effc541cc7bbf783b6602d2ca#code Resource: https://basescan.org/address/0x117908E5fbC5F77DAF07D4263bA79eE810b092f4#code Masterchef: https://basescan.org/address/0xE57CBe0aca2FB8edf2B6C68b99688d208C6Ef0D9#code Treasury: https://basescan.org/address/0x52fa4d0E30303F9d28e90a845bE520760A209432#code
Commit	N/A
Unit Tests	Not Provided

Social Medias

Telegram	https://t.me/unicow_finance
Twitter	https://twitter.com/UnicowNFT
Facebook	N/A
Instagram	N/A
GitHub	N/A
Reddit	N/A
Medium	N/A
Discord	https://discord.gg/unicownft
YouTube	N/A
TikTok	N/A
LinkedIn	N/A



Audit Summary

Version	Delivery Date	Change Log
v1.0	22. August 2023	<ul style="list-style-type: none"> · Layout Project · Automated/ Manual-Security Testing · Summary
v1.5	21. September 2023	<ul style="list-style-type: none"> · 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/contracts/NftManager.sol	bb5a0f9669dee106828fa17c30192daaa4b850d2
contracts/contracts/Treasury.sol	9766f339d0e869797db9a429f7d8d249c0c55a23
contracts/contracts/Resource.sol	fb3084a3350ca1b19c56adb463bc752e74d91c4c
contracts/contracts/Masterchef.sol	e79e531a53b8e6a555e84569b638d51e2f025559

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(local imports).

```
@openzeppelin/contracts/utils/Address.sol
@openzeppelin/contracts/utils/Strings.sol
@openzeppelin/contracts/access/Ownable.sol
@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol
@openzeppelin/contracts/security/ReentrancyGuard.sol
./interfaces/IResource.sol
./interfaces/ITreasury.sol
./interfaces/INFTDescriptor.sol
```

```
@openzeppelin/contracts/access/Ownable.sol
@openzeppelin/contracts/token/ERC20/IERC20.sol
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol
./interfaces/IResource.sol
./interfaces/IUniswapV2Factory.sol
./interfaces/IUniswapV2Router02.sol
./interfaces/IUniswapV2Pair.sol
```



```
@openzeppelin/contracts/access/Ownable.sol
@openzeppelin/contracts/token/ERC20/ERC20.sol
@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol
@openzeppelin/contracts/utils/Address.sol
@openzeppelin/contracts/utils/math/SafeMath.sol
./interfaces/ITreasury.sol
```

-  Context
-  Ownable
-   SafeMath
-   Address
-  IBEP20
-  ITreasury
-   SafeBEP20

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. This contract imports some local contracts which are not audited by us, and we will not be responsible for any security cause for those contracts.

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 needed within visibility modifier, such as public, private or internal, which determines the access level of the variable.

Components

 Contracts	 Libraries	 Interfaces	 Abstract
4	3	2	2


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
84	4











External	Internal	Private	Pure	View
53	117	9	17	30

StateVariables

Total	 Public
47	40



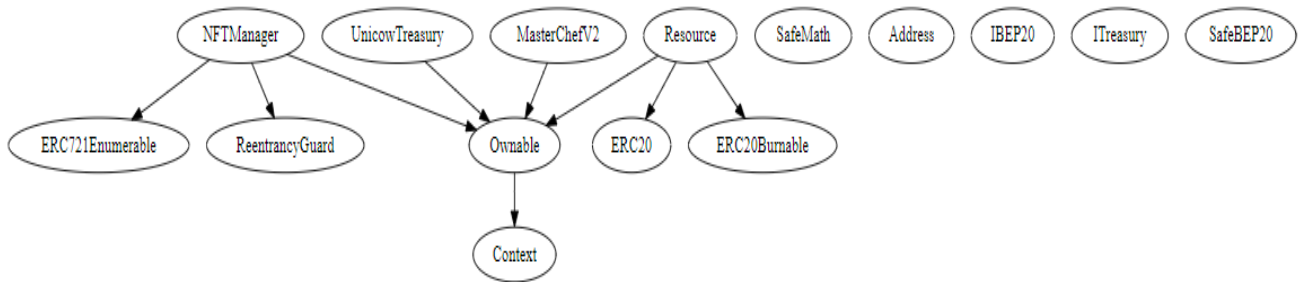
Capabilities

<div>Solidity Versions observed</div>	<div> Experimental Features</div>	<div> Can Receive Funds</div>	<div> Uses Assembly</div>	<div> Has Destroyable Contracts</div>	
<div><div>^0.8.21</div><div>0.7.0</div></div>	<div><div>-----</div></div>	<div><div>Yes</div></div>	<div><div>yes</div><div>(2 asm blocks)</div></div>	<div><div>-----</div></div>	
<div><div> Transfer s ETH</div></div>	<div><div> Low-Level Calls</div></div>	<div><div> Delegate Call</div></div>	<div><div> Uses Hash Functions</div></div>	<div><div> ECRrecover</div></div>	<div><div> New/Create/Create2</div></div>
<div><div>yes</div></div>	<div></div>	<div><div>yes</div></div>	<div></div>	<div></div>	



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.



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 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 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 cannot mint new tokens

✓ The owner cannot 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 treasury contract has the ability to mint new tokens. Also, this minting will going to be done in the staking functionality of the contract. And that tokens will be transfer to the referrer as a reward.

File/ Line(s): L82-84
Codebase: Resource.sol

```
function mintByTreasury(address account1, uint256 amount1) external onlyTreasury {
    super._mint(account1, amount1);
}
```

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

In some cases, burning tokens can be used as a tactic by the owner or developers to manipulate the token's value. If the owner or developers burn a significant number of tokens without transparency or justification, it can cause stakeholders to lose trust in the project and lead to a decrease in demand for the token.

Comment

The treasury contract can burn the tokens from the account during the staking in the contract.

File/ Line(s): L86-88

Codebase: Resource.sol

```
function burnByTreasury(address account!, uint256 amount!) external onlyTreasury {
|   super._burn(account!, amount!);
| }
}
```




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 set fee more than 25%.**

Description

The owner cannot set fees more than 25% in the contract.

Comment

The sell fees can be set not more than 25%.

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 cannot lock functions.



The owner cannot lock functions intentionally.

Description

This contract contains functions which will be locked until the startTime is not set by the owner, here are the following functions:

- 1 – createNFT
- 2 – stakeTokens
- 3 – cashoutReward
- 4 – cashoutAll
- 5 – compoundAll

The owner is able to adjust the startTime as long as the current block.timestamp is below the startTime. Otherwise, once it is started (block.timestamp is higher than the startTime) the owner is not able to change it afterward.

Comment

The functionalities will be locked in the contract by default until the startTime is not set by the owner. It is recommended that the start time should be started once the contract is deployed.

File/ Line(s): 73-76

codebase: NftManager.sol

```
modifier whenStarted() {
    require(startTime < block.timestamp, "NFTManager: Not started");
    _;
}
```

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
NftManager.sol	<ul style="list-style-type: none"> ➤ The owner can set the start time only once. ➤ The owner can update the treasury contract only once. ➤ The owner can update the mint price for whitelisted and un-whitelisted account of not more than 2 tokens. ➤ The owner can update the processing fees not more than 10%. ➤ The owner can whitelist addresses from fees. ➤ The owner can claim the stuck tokens.
Resource.sol	<ul style="list-style-type: none"> ➤ The Treasury contract can whitelist addresses from fees. ➤ The owner can set the treasury wallet address. ➤ The Treasury contract can mint an unlimited number of tokens. ➤ The Treasury contract can burn tokens from other wallets without any allowance. ➤ The owner can claim stuck tokens.
Masterchef.sol	<ul style="list-style-type: none"> ➤ The owner can update the start time after the initial deployment for the reward of tokens. ➤ The owner can change the treasury token after the initial deployment. ➤ The owner can update the rewards per second to any arbitrary value which means he can set it to zero then no rewards will be sent to the user. ➤ The owner can set any arbitrary value on the bonus multiplier for the token including zero. ➤ The owner can add the liquidity pair to the contract. ➤ The owner can update the resource allocation



Treasury.sol

- point.
- The owner can claim the ethers from the contract.
- The NFT manager or masterchef contract can swap tokens manually.
- The NFT manager or masterchef address can mint unlimited amount of reward tokens to any address including its own address.
- The NFT manager and masterchef address can burn an unlimited amount of tokens from any address.
- The owner can exclude wallets from fees.
- The owner can add liquidity into this contract manually.
- The owner can toggle allowance for the Uniswap router in this contract.
- The owner can enable and disable swapping.
- The owner can set any arbitrary value which is greater than zero as a minimum swap amount required for swapping.
- The owner can set any address as a few wallets address including zero or dead.
- The owner can claim stuck tokens.

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 burn tokens from external accounts without allowance.

File	Severity	Location	Status
Resource.sol	High	L87-89	Fixed

Description – The treasury contract has the ability to burn unlimited number of tokens from other wallets through this the supply of the tokens can be manipulated and user can lose their funds.

Remediation – It is recommended that no external wallets can burn tokens without any allowance. Add an allowance functionality to avoid these circumstances.

Alleviation – The treasury contract can only burn the specific amount of tokens with the help of the stake token function and the minting of tokens will be done to the referrer accounts.

#2 | Owner can mint an unlimited number of tokens.

File	Severity	Location	Status
Resource.sol	Medium	L83-85	Fixed

Description – The owner can mint an unlimited number of tokens which can manipulate the supply of tokens also the prices of the tokens can be manipulated due to unlimited minting.

Remediation – It is recommended that no minting should be done after the initial deployment of the token.

Alleviation – The treasury contract can mint the tokens to the referrer account as a reward for staking the tokens.

#3 | Owner can lock the token.

File	Severity	Location	Status
NftManager.sol	Medium	L73-76	Fixed

Description – This contract contains the pausable functionality which can lock the function for an unlimited period of time. The contract owner can pause minting, staking, and reward functionalities for an unlimited period which can result in the user is not able to mint tokens and will not be able

to stake the minted tokens and get rewards. Here, the contract owner can change the startTime which can pause some functionalities in the token for that period of time and if the value is so high then it can be locked for unlimited period of time.

Remediation – Remove the ‘pausable’ functionality from the contract as this can lock your token.

#4 | Logical error.

File	Severity	Location	Status
NftManager.sol	Medium	L221	ACK

Description – The amount of fees is calculated from the reward but not subtracted from the reward. So, because of this, the total value locked in the token is calculated without the subtracted value.

Remediation – Add the reward and fee calculation in order to get the optimum results.

#5 | Owner can set fees more than 100%.

File	Severity	Location	Status
NftManager.sol	Medium	L415-418	Fixed

Description – The owner can add any arbitrary value excluding zero as a processing fee in the contract. The fee calculation of $(\text{reward} * \text{processingFee}) / 100$ will be more than the reward itself, meaning the fees are higher than the actual rewards. This will result in a miscalculation of the costs sent to the treasury while compounding. The same in the "compoundReward" function.

Remediation – Add a ‘require’ check that fees cannot be more than 25% in the contract in any circumstances.

Alleviation – The fees cannot be set now at more than 10%.

#6 | Owner can claim contract’s own tokens.

File	Severity	Location	Status
OwnerRecovery.sol	Medium	L12-18	Fixed

Description – The contract owner can claim contract's own token from the contract which is not recommended as this contract contains the staking functionality in the contract if the contract owner claim the contract's own tokens then the user can lose their funds.

Remediation – Add a 'require' check that the owner cannot claim the contract's own token in any scenario.

#7 | Owner can drain liquidity.

File	Severity	Location	Status
Treasury.sol	Medium	L246-248	Fixed

Description – The owner can claim the stuck balance from the token and the liquidity of the token will be sent to the contract address. Because of this the owner has the ability to drain the liquidity from the contract.

Remediation – Remove the lost ether functionality or add liquidity to the dead address to avoid the above circumstances.

Alleviation – The contract liquidity is added to address zero.

#8 | Owner can mint tokens in their own wallet.

File	Severity	Location	Status
NftManager.sol	Low	L111-131	ACK

Description – The owner can use the airdrop functionality to mint the NFTs to his own wallet without paying any fees.

Remediation – Add a 'require' check that the owner cannot be able to mint in his own account.

#9 | Missing events arithmetic.

File	Severity	Location	Status
NftManager.sol	Low	L395-398	ACK
Resource.sol	Low	L95-108	ACK
Masterchef.sol	Low	L733-735, L737-745, L747-752, L754-756, L758-760, L764-780, L783-790	ACK
Treasury.sol	Low	L52-77, L229-232,	ACK

		L221-227, L244-249	
--	--	--------------------	--

Description – Emit all the critical parameter changes.

#10 | Floating pragma solidity version.

File	Severity	Location	Status
NftManager.sol	Low	L3	ACK
Resource.sol	Low	L3	ACK
Treasury.sol	Low	L3	ACK
Masterchef.sol	Low	L3	ACK

Description – Adding the constant latest version of solidity is recommended, as this prevents the unintentional deployment of a contract with an outdated compiler that contains unresolved bugs.

#11 | Owner can set any arbitrary value in the mint price.

File	Severity	Location	Status
NftManager.sol	Low	L415-418	Fixed

Description – The mint amount can be set to any arbitrary value apart from zero. As this can result in the locking of minting if the amount is set so high no minting of tokens will be possible.

Remediation – Add a 'require' check that the price must lie between certain threshold and cannot be extended through that range.

#12 | Owner can set any arbitrary value in the processing delay time.

File	Severity	Location	Status
NftManager.sol	Low	L344-347	Fixed

Description – The processing delay time can be set to any arbitrary value apart from zero. As this can stop the minting of tokens if the value is set so high then no minting of tokens will be possible.

Remediation – It is recommended that the value must lie between a certain threshold to avoid these circumstances where the functionality of the tokens can be locked.

#13 | Remove unused code.

File	Severity	Location	Status
Resource.sol	Low	L28-35	Fixed

Description – The only treasury modifier is implemented but not used anywhere in the contract.

Remediation – It is recommended to remove the unused code to avoid high gas fees.

#14 | Missing 'isContract' check.

File	Severity	Location	Status
Resource.sol	Low	L93-100	Fixed
Treasury.sol	Low	L62-81	Fixed

Description – The owner can add any address excluding zero as a treasury address. Add a 'require' check that the address must be a contract address otherwise the functionality will fail. Also, add a 'require' check on the treasury contract while initializing the parameters in the constructor make sure that the NFT manager address must be a contract address.

#15 | Missing zero or dead address check.

File	Severity	Location	Status
Masterchef.sol	Low	L747-751	ACK
Treasury.sol	Low	L243-247	ACK

Description – Add a 'require' check that the address must not be zero or dead.

#16 | Missing Threshold.

File	Severity	Location	Status
NftManager.sol	Low	L383-391, L393-402	ACK

Description – There must be a certain threshold where these values lie between them. In this contract, these values can be set to any arbitrary



value which is not recommended. For Example, the processing fees should not be more than 25% but it can be set to more than 100%.

Remediation – Add a ‘require’ condition for each function where the values will lie between these ranges only.

#17 | NatSpec Documentation missing.

File	Severity	Location	Status
NftManager.sol	Informational	--	ACK
Masterchef.sol	Informational	--	ACK
Treasury.sol	Informational	--	ACK
Resource.sol	Informational	--	ACK

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



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