Cyberscope

Audit Report

Myntflo Staking

February 2023

Network	MATIC
Address	0xe6a82adf644763ca2c3b47c5850c833542c3a098
Audited by	© cyberscope

Table of Contents

Table of Contents 1
Review 3
Audit Updates 3
Source Files 4
Introduction 6
Roles 6
Rewards Formula 7
Diagnostics 10
PSU - Potential Subtraction Underflow 11
Description 11
OCTD - Transfers Contract's Tokens 12
Description 12
Recommendation 12
PDT - Performant Data Type 13
Description 13
Recommendation 13
DPI - Decimals Precision Inconsistency 14
Description 14
Recommendation 14
L04 - Conformance to Solidity Naming Conventions 16
Description 16
Recommendation 17
L06 - Missing Events Access Control 18
Description 18
Recommendation 18
L13 - Divide before Multiply Operation 19
Description 19
Recommendation 19
L16 - Validate Variable Setters 20
Description 20
Recommendation 20
L19 - Stable Compiler Version 21

Description 21
Recommendation 21
L20 - Succeeded Transfer Check 22
Description 22
Recommendation 22
Functions Analysis 23
Inheritance Graph 28
Flow Graph 29
Summary 30
Disclaimer 31
About Cyberscope 32

Review

Contract Name	MyntfloStaking
Compiler Version	v0.8.17+commit.8df45f5f
Optimization	200 runs
Explorer	https://polygonscan.com/address/0xe6a82adf644763ca2c3b47c5850c833542c3a098
Address	0xe6a82adf644763ca2c3b47c5850c833542c3a098
Network	MATIC

Audit Updates

Initial Audit	06 Jan 2023
Corrected Phase 2	21 Feb 2023

Source Files

Filename	SHA256
@openzeppelin/contracts/metatx/ERC2771Context.sol	350e132f5ebc838e000770ceee044e4541a598b05bf998e96285c859eea5d8ef
@openzeppelin/contracts/metatx/MinimalForwarder.sol	699c5b0f8ba44ddcaef0af6bb5ddac7492a2d4078172aa172ac0ecf0239981e0
@openzeppelin/contracts/security/ReentrancyGuard.sol	3b30604df38d0f9b2b281a3e6661eb1b9cd577579e66225c674df21ca5b89b2c
@openzeppelin/contracts/token/ERC20/extensions/draft-IERC20Permit.sol	3e7aa0e0f69eec8f097ad664d525e7b3f0a3fda8dcdd97de5433ddb131db86ef
@openzeppelin/contracts/token/ERC20/IERC20.sol	94f23e4af51a18c2269b355b8c7cf4db8003d075c9c541019eb8dcf4122864d5
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol	0c8a43f12ac2081c6194d54da96f02ebc457760d6514f6b940689719fcef8c0a
@openzeppelin/contracts/token/ERC721/IERC721.sol	c7703068bac02fe1cdf109e38faf10399c66eb411e4c9ae0d70c009eca4bf5ef
@openzeppelin/contracts/utils/Address.sol	8160a4242e8a7d487d940814e5279d934e81f0436689132a4e73394bab084a6d
@openzeppelin/contracts/utils/Context.sol	1458c260d010a08e4c20a4a517882259a23a4baa0b5bd9add9fb6d6a1549814a
@openzeppelin/contracts/utils/cryptography/ECDSA.sol	d18195404f37ee86b44cfb01858b76ac0d4d17b77328fa82895ee893718cb0c2
@openzeppelin/contracts/utils/cryptography/EIP712.sol	8e8907de613172eb24cb7c8c6ae34381bfe5aa38d9998e27d3065e3a711390c0
@openzeppelin/contracts/utils/introspection/IERC165.sol	701e025d13ec6be09ae892eb029cd83b3064325801d73654847a5fb11c58b1e5

@openzeppelin/contracts/utils/math/Math.sol	8059d642ec219d0b9b62fbc76912079529cf494cac988abe5e371f1168b29b0f
@openzeppelin/contracts/utils/Strings.sol	f81f11dca62dcd3e0895e680559676f4ba4f2e12a36bb0291d7ecbb6b983141f
contracts/MyntfloStaking.sol	f5a105194310a0b859dda43291ec0dfddc4d122492d608457cd5217ab188950e
hardhat/console.sol	47a72fddde001a2977f460b759ce035f9788daa34c186ce1f9a10066236b3f75

Introduction

MyntfloStaking implements an NFT staking mechanism where users have the ability to stake NFTs in order to receive rewards. The eligible NFT collections are defined by the contract owner. Additionally, the contract owner has the authority to change the rewarded token address. The reward amount is calculated by a formula proportionally to the time period that has elapsed. The reward amount is redeemed by an ERC20 token. The users have the ability to claim their rewards and unstack their deposits at any time.

The contract does not implement any mechanism that guarantees the reward reserves. This is a common methodology in many staking contracts since the staking period does not have an expiration date and the reward token does not implement a public mint method. The contract owner is responsible for keeping the reward reserves in a healthy amount.

Roles

Public Roles

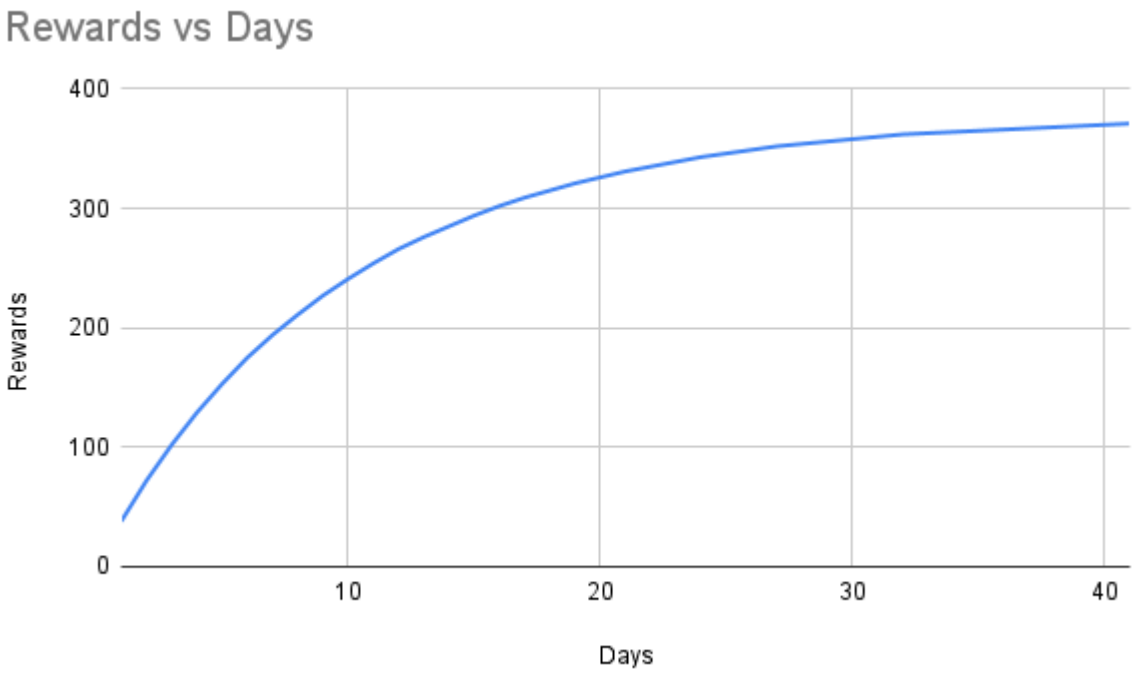
- stake()
- unstake()
- claimRewards()

Admin Roles

- setRewardsToken()
- setElegibleCollection()
- setElegibleCollections()

Rewards Formula

As an integral part of the audit report, the rewards algorithm was tested by providing applicable values. The following chart depicts the rewards (Y-Axis), normalized by 10^18, that will be provided when a specific time period has elapsed (X-Axis). For instance, if the time that has elapsed since the last claim is 5 days, then the rewarded amount will be 153. If more than 41 days have elapsed then the reward amount is stabilized at 371.



The following table depicts the rewarded amount across the period that has elapsed.

Days	Reward * 10^18
1	38
2	72
3	102
4	129
5	153
6	175
7	194
8	211
9	227
10	241
11	254
12	266
13	276
14	285
15	294
16	302
17	309
18	315
19	321
20	326
21	331
22	335
23	339
24	343
25	346
26	349
27	352
28	354
29	356
30	358
31	360
32	362

33	363
34	364
35	365
36	366
37	367
38	368
39	369
40	370
41	371

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	PSU	Potential Subtraction Underflow	Unresolved
●	OCTD	Transfers Contract's Tokens	Unresolved
●	PDT	Performant Data Type	Unresolved
●	DPI	Decimals Precision Inconsistency	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L06	Missing Events Access Control	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

PSU - Potential Subtraction Underflow

Criticality	Minor / Informative
Location	contracts/MyntfloStaking.sol#L180
Status	Unresolved

Description

The contract owner has the authority to change the `timeStacked` and `timeOfLastUpdate` of a user's staking record.

```
function setTimeOfLastUpdate(address staker, uint256 index, uint256
_timeStaked, uint256 _timeUpdated) external onlyOwner {
stakers[staker].stakedTokens[index].timeStaked = _timeStaked;
stakers[staker].stakedTokens[index].timeOfLastUpdate = _timeUpdated;
}
```

If the owner sets a value greater than the current timestamp, then the `unstake()` method will underflow.

```
uint256 secondsPassed = block.timestamp -
stakers[_msgSender()].stakedTokens[index].timeOfLastUpdate;
```

OCTD - Transfers Contract's Tokens

Criticality	Minor / Informative
Location	contracts/MyntfloStaking.sol#L189
Status	Unresolved

Description

The contract owner has the authority to claim all the balance of reward tokens from the contract. The owner may take advantage of it by calling the `withdrawRewardsToken` function.

```
function withdrawRewardsToken() external onlyOwner {
    rewardsToken.transfer(owner, rewardsToken.balanceOf(address(this)));
}
```

Recommendation

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. That risk can be prevented by temporarily locking the contract or renouncing ownership.

PDT - Performant Data Type

Criticality	Minor / Informative
Location	contracts/testingDeploy/MyntfloStaking.sol#L26
Status	Unresolved

Description

The `StakedToken` struct stores staker's address. The `StakedToken` struct is part of the `Staker` struct. The `Staker` struct is pointed by the staker's mapping. The contract does not contain any other structure that is pointing directly to the `StakedToken` struct. The `StakedToken.staker` variable is used by the contract to determine if the stack record is active. This requirement could also be archived by exploiting a boolean data type.

```
struct StakedToken {
    address staker;
    uint256 tokenId;
    uint256 timeStaked;
    uint256 timeOfLastUpdate;
    address contractAddress;
}
```

Recommendation

Since the staker's address is solely used as a boolean indicator. The team is advised to use a boolean indicator rather than an address since the boolean data type will reduce the storage size and improve the gas cost.

DPI - Decimals Precision Inconsistency

Criticality	Minor / Informative
Status	Unresolved

Description

The decimals field of a contract's ERC20 token can be used to specify the number of decimal places that the token uses. For example, if decimals are set to `8`, it means that the smallest unit of the token is `0.00000001`, and if decimals are set to `18`, it means that the smallest unit of the token is `0.000000000000000001`.

However, there is an inconsistency in the way that the decimals field is handled in some ERC20 contracts. The ERC20 specification does not specify how the decimals field should be implemented, and as a result, some contracts use different precision numbers.

This inconsistency can cause problems when interacting with these contracts, as it is not always clear how the decimals field should be interpreted. For example, if a contract expects the decimals field to be 18 digits, but the contract being interacted with uses 8 digits, the result of the interaction may not be what was expected.

The contract uses the `decay` method to calculate the rewarded amount. The `decay` is not taking into consideration the rewarded token decimals. As a result, the reward amount may vary according to the size of the token decimal. As a result, unexpected reward amounts may be produced.

```
uint256 rewards = decay(secondsPassed);
rewardsToken.safeTransfer(_msgSender(), rewards);
```

Recommendation

To avoid these issues, it is important to carefully review the implementation of the `decay` method. The team is advised to take into consideration the rewarded token decimals inside the `decay` algorithm.

The following example depicts the reward result of 4 tokens with different decimals precision. We assume that the `decay` will return the value `10`.

ERC20	Decimals	Reward
Token 1	6	3
Token 2	9	5
Token 3	18	10
Token 4	24	13

All the decimals could be normalized to 18 since it represents the ERC20 token with the greatest digits.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/MyntfloStaking.sol#L64,100,145,149,153,180,198,204,251
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
address _tokenContract
uint256 _tokenId
IERC20 _rewardsToken
address _collection
bool _elegible
bool[] memory _elegible
address[] memory _collections
uint256 _timeStaked
uint256 _timeUpdated
address _staker
address _user
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.

Find more information on the Solidity documentation https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L06 - Missing Events Access Control

Criticality	Minor / Informative
Location	contracts/MyntfloStaking.sol#L186
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task. There are functions that have no event emitted, so it is difficult to track off-chain changes.

```
owner = newOwner
```

Recommendation

To avoid this issue, it's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	contracts/MyntfloStaking.sol#L242
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
cumulativeRewards += (A / ((B + i) ** X)) * 10 ** 18
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	contracts/MyntfloStaking.sol#L186
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
owner = newOwner
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/MyntfloStaking.sol#L2
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.4;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	contracts/MyntfloStaking.sol#L190
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
rewardsToken.transfer(owner, rewardsToken.balanceOf(address(this)))
```

Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
ERC2771Context	Implementation	Context		
		Public	✓	-
	isTrustedForwarder	Public		-
	_msgSender	Internal		
	_msgData	Internal		
MinimalForwarder	Implementation	EIP712		
		Public	✓	EIP712
	getNonce	Public		-
	verify	Public		-
	execute	Public	Payable	-
ReentrancyGuard	Implementation			
		Public	✓	-
	_nonReentrantBefore	Private	✓	
	_nonReentrantAfter	Private	✓	
IERC20Permit	Interface			
	permit	External	✓	-
	nonces	External		-
	DOMAIN_SEPARATOR	External		-
IERC20	Interface			

	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
SafeERC20	Library			
	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	safeApprove	Internal	✓	
	safeIncreaseAllowance	Internal	✓	
	safeDecreaseAllowance	Internal	✓	
	safePermit	Internal	✓	
	_callOptionalReturn	Private	✓	
IERC721	Interface	IERC165		
	balanceOf	External		-
	ownerOf	External		-
	safeTransferFrom	External	✓	-
	safeTransferFrom	External	✓	-
	transferFrom	External	✓	-
	approve	External	✓	-
	setApprovalForAll	External	✓	-
	getApproved	External		-
	isApprovedForAll	External		-
Address	Library			
	isContract	Internal		

	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	verifyCallResultFromTarget	Internal		
	verifyCallResult	Internal		
	_revert	Private		
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
ECDSA	Library			
	_throwError	Private		
	tryRecover	Internal		
	recover	Internal		
	tryRecover	Internal		
	recover	Internal		
	tryRecover	Internal		
	recover	Internal		
	toEthSignedMessageHash	Internal		
	toEthSignedMessageHash	Internal		
	toTypedDataHash	Internal		

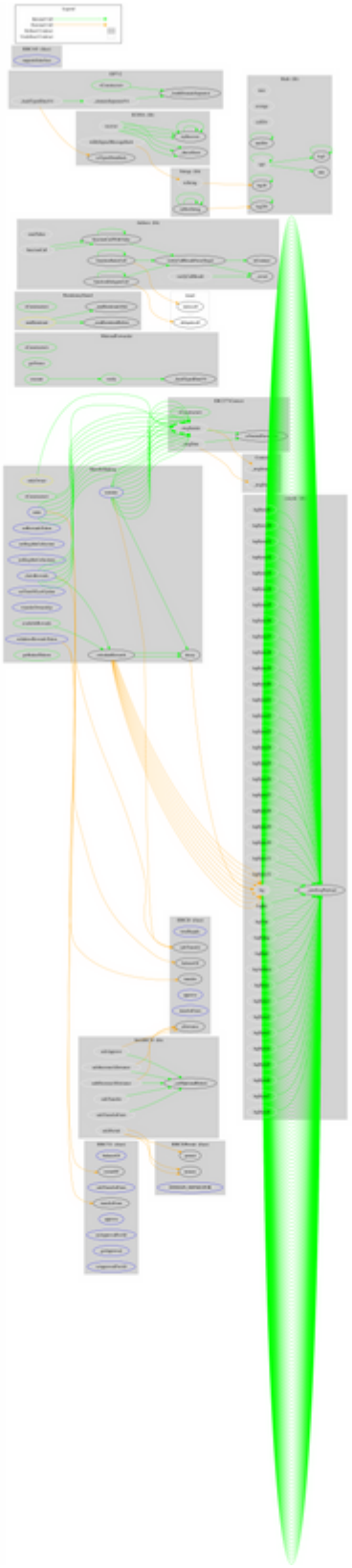
EIP712	Implementation			
		Public	✓	-
	_domainSeparatorV4	Internal		
	_buildDomainSeparator	Private		
	_hashTypedDataV4	Internal		
IERC165	Interface			
	supportsInterface	External		-
Math	Library			
	max	Internal		
	min	Internal		
	average	Internal		
	ceilDiv	Internal		
	mulDiv	Internal		
	mulDiv	Internal		
	sqrt	Internal		
	sqrt	Internal		
	log2	Internal		
	log2	Internal		
	log10	Internal		
	log10	Internal		
	log256	Internal		
	log256	Internal		
Strings	Library			
	toString	Internal		
	toHexString	Internal		
	toHexString	Internal		

	toHexString	Internal		
MyntfloStaking	Implementation	Reentrancy Guard, ERC2771Context		
		Public	✓	ERC2771Context
	stake	External	✓	nonReentrant
	unstake	External	✓	nonReentrant
	setRewardsToken	External	✓	onlyOwner
	setElegibleCollection	External	✓	onlyOwner
	setElegibleCollections	External	✓	onlyOwner
	claimRewards	External	✓	-
	setTimeOfLastUpdate	External	✓	onlyOwner
	transferOwnership	External	✓	onlyOwner
	withdrawRewardsToken	External	✓	onlyOwner
	availableRewards	Public		-
	getStakedTokens	Public		-
	decay	Internal		
	calculateRewards	Internal		

Inheritance Graph



Flow Graph



Summary

Myntflo contract implements an NFT staking mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope’s prior written consent. This report is not nor should be considered an “endorsement” or “disapproval” of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment 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 Cyberscope’s position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.



The Cyberscope team

https://www.cyberscope.io

Cyberscope

Audit Report

Myntflo Marketplace

January 2023

Github	https://github.com/oxalexa/myntflo-contracts
Commit	3abe6b1e5bc1534f1aa7c181d8adb574477e3fa2
Audited by	© cyberscope

Table of Contents

Table of Contents 1
Review 2
Audit Updates 2
Source Files 3
Introduction 5
Roles 5
Diagnostics 6
PTC - Potential TokenId Conflict 7
Description 7
Recommendation 7
L04 - Conformance to Solidity Naming Conventions 8
Description 8
Recommendation 8
L19 - Stable Compiler Version 9
Description 9
Recommendation 9
Functions Analysis 10
Inheritance Graph 15
Flow Graph 16
Summary 17
Disclaimer 18
About Cyberscope 19

Review

Contract Name	MyntfloMarketplace
Repository	https://github.com/oxalexa/myntflo-contracts
Commit	3abe6b1e5bc1534f1aa7c181d8adb574477e3fa2

Audit Updates

Initial Audit	12 Jan 2023

Source Files

Filename	SHA256
@openzeppelin/contracts/metatx/ERC2771Context.sol	350e132f5ebc838e000770ceee044e4541a598b05bf998e96285c859eea5d8ef
@openzeppelin/contracts/metatx/MinimalForwarder.sol	95a2f6b10918f410d143f27581a0a1c7603c9dd774c31899bb4cc20cc1619515
@openzeppelin/contracts/token/ERC1155/IERC1155.sol	fd6a1801f1f2f8af0a3ece0b254da06ec24568aec02cfe94827061379aebc6f3
@openzeppelin/contracts/token/ERC1155/IERC1155Receiver.sol	578834a1bcdac6a22de5e07ae63bbbd4d41615f35950afc6e6c068d92619b334
@openzeppelin/contracts/token/ERC1155/utils/ERC1155Holder.sol	a7ad38fa0a06fe6e24f81fee4f1fc3870767db96d1ba37df7be1199f7a3ace7f
@openzeppelin/contracts/token/ERC1155/utils/ERC1155Receiver.sol	cf407886a0ce7e2af7efe7867e2d2864903426f63eeaa68eecf33d57f7d910c2
@openzeppelin/contracts/token/ERC20/IERC20.sol	94f23e4af51a18c2269b355b8c7cf4db8003d075c9c541019eb8dcf4122864d5
@openzeppelin/contracts/token/ERC721/IERC721.sol	fde830ac73ef320f7e3ce977b8cf567173f1e479ba86d584498f8362a67a5dc0
@openzeppelin/contracts/token/ERC721/IERC721Receiver.sol	77f0f7340c2da6bb9edbc90ab6e7d3eb8e2ae18194791b827a3e8c0b11a09b43
@openzeppelin/contracts/token/ERC721/utils/ERC721Holder.sol	2cfe4ed66b63283ca12b0360b1c8c1a3c298a510e2e29c60d9ccedb634b738a6
@openzeppelin/contracts/utils/Context.sol	1458c260d010a08e4c20a4a517882259a23a4baa0b5bd9add9fb6d6a1549814a
@openzeppelin/contracts/utils/cryptography/draft-EIP712.sol	fc0e6c5d7184bd03b8deae6ca9a48a1eaaecf9f5e4703611aabfb63401e6d43f

@openzeppelin/contracts/utils/cryptography/ECDSA.sol	4e45d53327d561848fbcf381262ec5c0ac91b2f1f06432210bf76db55279d945
@openzeppelin/contracts/utils/introspection/ERC165.sol	8806a632d7b656cadb8133ff8f2acae4405b3a64d8709d93b0fa6a216a8a6154
@openzeppelin/contracts/utils/introspection/IERC165.sol	701e025d13ec6be09ae892eb029cd83b3064325801d73654847a5fb11c58b1e5
@openzeppelin/contracts/utils/math/SafeMath.sol	0dc33698a1661b22981abad8e5c6f5ebca0dfe5ec14916369a2935d888ff257a
@openzeppelin/contracts/utils/Strings.sol	34127ad0054df5963b0fd694c1b313d17e9114a2f426b85526d6d976210298ab
contracts/testingDeploy/MyntfloMarketplace.sol	47b32ed062bfa99280830de0c88d360ee7fa0bcf3146e96b4d1b7dbe2918caa1

Introduction

MyntfloMarketplace implements marketplace/exchange mechanism. The contract owner deposits ERC1155 and ERC721 tokens to the contract and sets a specific price per token. Then the users have the ability to buy each of those tokens by providing the "paymentToken" as currency. The "paymentToken" can be configured by the contract owner.

Roles

Owner

- setPaymentToken()
- addERC1155()
- removeERC1155()
- addERC721()
- removeERC721()

Users

buyToken()

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	PTC	Potential TokenId Conflict	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L19	Stable Compiler Version	Unresolved

PTC - Potential TokenId Conflict

Criticality	Minor / Informative
Status	Unresolved

Description

The contract combines two different token protocols in one collection. The protocols are ERC721 and ERC1155. Both protocols are indexing the corresponding tokens with a tokenId. Since these two protocols consist of different instances, then the same tokenId value could index a token from both collections. The contract does not deterministically guarantee the uniqueness of the tokenId. The token relies on the fact that the contract owner will provide the proper `contractAddress` that will distinguish the potential conflict.

```
function findListingByTokenId(uint256 tokenId, address contractAddress) public
view returns (bool, uint256) {
    for (uint256 i = 0; i < listings.length; i++) {
        if (listings[i].tokenId == tokenId && listings[i].contractAddress ==
contractAddress) {
            return (true, i);
        }
    }
    return (false, 0);
}
```

Recommendation

The team is advised to add an additional boolean indicator (`isERC1155`) in the `findListingByTokenId()` method. That way the contract will guarantee that even if the owner misuses the method parameters, there will not be a conflict between the two protocols.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/testingDeploy/MyntfloMarketplace.sol#L68
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
IERC20 _paymentToken
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.

Find more information on the Solidity documentation https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/testingDeploy/MyntfloMarketplace.sol#L2
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.13;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
ERC2771Context	Implementation	Context		
		Public	✓	-
	isTrustedForwarder	Public		-
	_msgSender	Internal		
	_msgData	Internal		
MinimalForwarder	Implementation	EIP712		
		Public	✓	EIP712
	getNonce	Public		-
	verify	Public		-
	execute	Public	Payable	-
IERC1155	Interface	IERC165		
	balanceOf	External		-
	balanceOfBatch	External		-
	setApprovalForAll	External	✓	-
	isApprovedForAll	External		-
	safeTransferFrom	External	✓	-
	safeBatchTransferFrom	External	✓	-
IERC1155Receiver	Interface	IERC165		
	onERC1155Received	External	✓	-
	onERC1155BatchReceived	External	✓	-

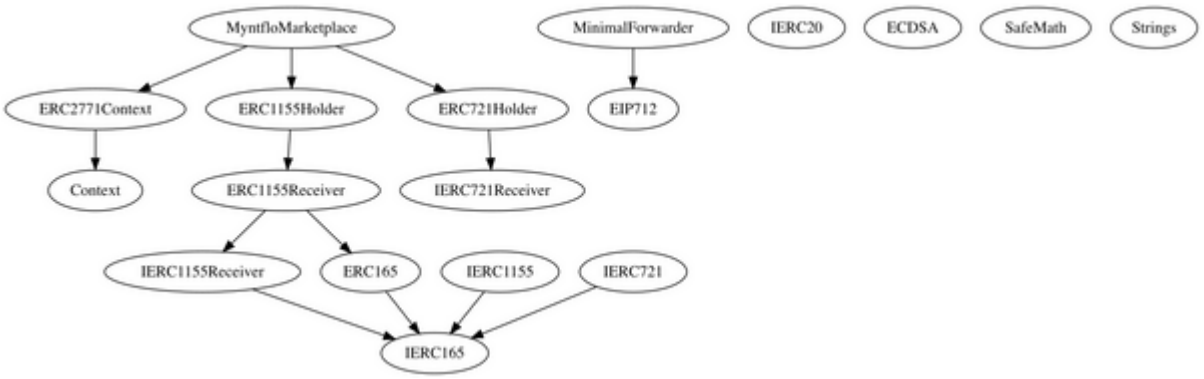
ERC1155Holder	Implementation	ERC1155Receiver		
	onERC1155Received	Public	✓	-
	onERC1155BatchReceived	Public	✓	-
ERC1155Receiver	Implementation	ERC165, IERC1155Receiver		
	supportsInterface	Public		-
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC721	Interface	IERC165		
	balanceOf	External		-
	ownerOf	External		-
	safeTransferFrom	External	✓	-
	safeTransferFrom	External	✓	-
	transferFrom	External	✓	-
	approve	External	✓	-
	setApprovalForAll	External	✓	-
	getApproved	External		-
	isApprovedForAll	External		-
IERC721Receiver	Interface			

	onERC721Received	External	✓	-
ERC721Holder	Implementation	IERC721Receiver		
	onERC721Received	Public	✓	-
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
EIP712	Implementation			
		Public	✓	-
	_domainSeparatorV4	Internal		
	_buildDomainSeparator	Private		
	_hashTypedDataV4	Internal		
ECDSA	Library			
	_throwError	Private		
	tryRecover	Internal		
	recover	Internal		
	tryRecover	Internal		
	recover	Internal		
	tryRecover	Internal		
	recover	Internal		
	toEthSignedMessageHash	Internal		
	toEthSignedMessageHash	Internal		
	toTypedDataHash	Internal		
ERC165	Implementation	IERC165		
	supportsInterface	Public		-

IERC165	Interface			
	supportsInterface	External		-
SafeMath	Library			
	tryAdd	Internal		
	trySub	Internal		
	tryMul	Internal		
	tryDiv	Internal		
	tryMod	Internal		
	add	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	mod	Internal		
	sub	Internal		
	div	Internal		
	mod	Internal		
Strings	Library			
	toString	Internal		
	toHexString	Internal		
	toHexString	Internal		
	toHexString	Internal		
MyntfloMarketplace	Implementation	ERC2771Context, ERC1155Holder, ERC721Holder		
		Public	✓	ERC2771Context

	setPaymentToken	Public	✓	onlyOwner
	getListings	Public		-
	findListingByTokenId	Public		-
	addERC1155	Public	✓	onlyOwner
	removeERC1155	Public	✓	onlyOwner
	addERC721	Public	✓	onlyOwner
	removeERC721	Public	✓	onlyOwner
	buyToken	Public	✓	-

Inheritance Graph



Flow Graph



Summary

MyntfloMarketplace contract implements a marketplace mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope’s prior written consent. This report is not nor should be considered an “endorsement” or “disapproval” of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment 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 Cyberscope’s position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.



The Cyberscope team

https://www.cyberscope.io

Cyberscope

Audit Report

Myntflo Staking

January 2023

Github	https://github.com/oxalexa/myntflo-contracts
Commit	3abe6b1e5bc1534f1aa7c181d8adb574477e3fa2
Audited by	© cyberscope

Table of Contents

Table of Contents 1
Review 2
Audit Updates 2
Source Files 3
Introduction 5
Roles 5
Rewards Formula 6
Diagnostics 9
DPI - Decimals Precision Inconsistency 10
Description 10
Recommendation 10
PDT - Performant Data Type 12
Description 12
Recommendation 12
L04 - Conformance to Solidity Naming Conventions 13
Description 13
Recommendation 14
L19 - Stable Compiler Version 15
Description 15
Recommendation 15
Functions Analysis 16
Inheritance Graph 22
Flow Graph 23
Summary 24
Disclaimer 25
About Cyberscope 26

Review

Contract Name	MyntfloStaking
Repository	https://github.com/oxalexa/myntflo-contracts
Commit	3abe6b1e5bc1534f1aa7c181d8adb574477e3fa2
Audit Scope	MyntfloStaking.sol

Audit Updates

Initial Audit	06 Jan 2023

Source Files

Filename	SHA256
@openzeppelin/contracts/metatx/ERC2771Context.sol	350e132f5ebc838e000770ceee044e4541a598b05bf998e96285c859eea5d8ef
@openzeppelin/contracts/metatx/MinimalForwarder.sol	95a2f6b10918f410d143f27581a0a1c7603c9dd774c31899bb4cc20cc1619515
@openzeppelin/contracts/security/ReentrancyGuard.sol	aa73590d5265031c5bb64b5c0e7f84c44cf5f8539e6d8606b763adac784e8b2e
@openzeppelin/contracts/token/ERC20/extensions/draft-IERC20Permit.sol	3e7aa0e0f69eec8f097ad664d525e7b3f0a3fda8dcdd97de5433ddb131db86ef
@openzeppelin/contracts/token/ERC20/IERC20.sol	94f23e4af51a18c2269b355b8c7cf4db8003d075c9c541019eb8dcf4122864d5
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol	fa36a21bd954262006d806b988e4495562e7b50420775e2aa0deecb596fd1902
@openzeppelin/contracts/token/ERC721/IERC721.sol	fde830ac73ef320f7e3ce977b8cf567173f1e479ba86d584498f8362a67a5dc0
@openzeppelin/contracts/utils/Address.sol	1e0922f6c0bf6b1b8b4d480dcabb691b1359195a297bde6dc5172e79f3a1f826
@openzeppelin/contracts/utils/Context.sol	1458c260d010a08e4c20a4a517882259a23a4baa0b5bd9add9fb6d6a1549814a
@openzeppelin/contracts/utils/cryptography/draft-EIP712.sol	fc0e6c5d7184bd03b8deae6ca9a48a1eaaecf9f5e4703611aabfb63401e6d43f
@openzeppelin/contracts/utils/cryptography/ECDSA.sol	4e45d53327d561848fbcf381262ec5c0ac91b2f1f06432210bf76db55279d945
@openzeppelin/contracts/utils/introspection/IERC165.sol	701e025d13ec6be09ae892eb029cd83b3064325801d73654847a5fb11c58b1e5

@openzeppelin/contracts/utils/Strings.sol	34127ad0054df5963b0fd694c1b313d17e9114a2f426b85526d6d976210298ab
contracts/testingDeploy/MyntfloStaking.sol	be9b8061d21a225b04d1d07dec1a78483c19e0efaa9a5b06e830f2c0820625da
hardhat/console.sol	47a72fddde001a2977f460b759ce035f9788daa34c186ce1f9a10066236b3f75

Introduction

MyntfloStaking implements an NFT staking mechanism where users have the ability to stake NFTs in order to receive rewards. The eligible NFT collections are defined by the contract owner. Additionally, the contract owner has the authority to change the rewarded token address. The reward amount is calculated by a formula proportionally to the time period that has elapsed. The reward amount is redeemed by an ERC20 token. The users have the ability to claim their rewards and unstack their deposits at any time.

The contract does not implement any mechanism that guarantees the reward reserves. This is a common methodology in many staking contracts since the staking period does not have an expiration date and the reward token does not implement a public mint method. The contract owner is responsible for keeping the reward reserves in a healthy amount.

Roles

Public Roles

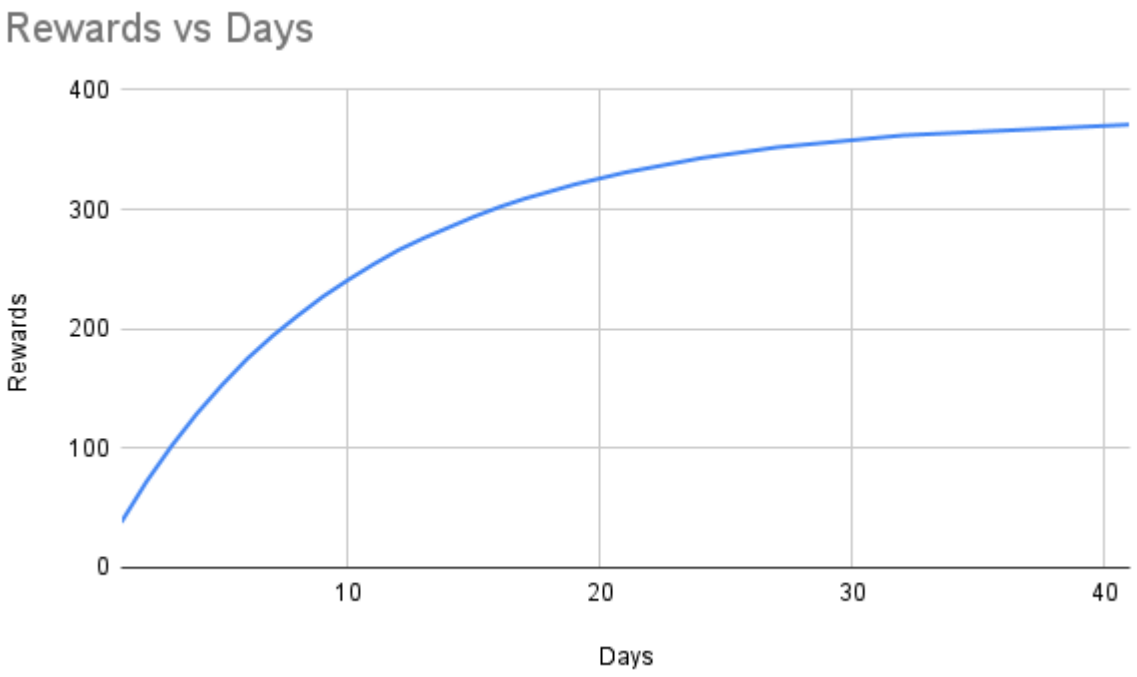
- stake()
- unstake()
- claimRewards()

Admin Roles

- setRewardsToken()
- setElegibleCollection()
- setElegibleCollections()

Rewards Formula

As an integral part of the audit report, the rewards algorithm was tested by providing applicable values. The following chart depicts the rewards (Y-Axis) that will be provided when a specific time period has elapsed (X-Axis). For instance, if the time that has elapsed since the last claim is 5 days, then the rewarded amount will be 153. If more than 41 days have elapsed then the reward amount is stabilized at 371.



The following table depicts the rewarded amount across the period that has elapsed.

Days	Reward
1	38
2	72
3	102
4	129
5	153
6	175
7	194
8	211
9	227
10	241
11	254
12	266
13	276
14	285
15	294
16	302
17	309
18	315
19	321
20	326
21	331
22	335
23	339
24	343
25	346
26	349
27	352
28	354
29	356
30	358
31	360
32	362

33	363
34	364
35	365
36	366
37	367
38	368
39	369
40	370
41	371

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	DPI	Decimals Precision Inconsistency	Unresolved
	PDT	Performant Data Type	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L19	Stable Compiler Version	Unresolved

DPI - Decimals Precision Inconsistency

Criticality	Minor / Informative
Status	Unresolved

Description

The decimals field of a contract's ERC20 token can be used to specify the number of decimal places that the token uses. For example, if decimals is set to `8`, it means that the smallest unit of the token is `0.00000001`, and if decimals are set to `18`, it means that the smallest unit of the token is `0.000000000000000001`.

However, there is an inconsistency in the way that the decimals field is handled in some ERC20 contracts. The ERC20 specification does not specify how the decimals field should be implemented, and as a result, some contracts use different precision numbers.

This inconsistency can cause problems when interacting with these contracts, as it is not always clear how the decimals field should be interpreted. For example, if a contract expects the decimals field to be 18 digits, but the contract being interacted with uses 8 digits, the result of the interaction may not be what was expected.

The contract uses the `decay` method to calculate the rewarded amount. The `decay` is not taking into consideration the rewarded token decimals. As a result, the reward amount may vary according to the size of the token decimal. As a result, unexpected reward amounts may be produced.

```
uint256 rewards = decay(secondsPassed);
rewardsToken.safeTransfer(_msgSender(), rewards);
```

Recommendation

To avoid these issues, it is important to carefully review the implementation of the `decay` method. The team is advised to take into consideration the rewarded token decimals inside the `decay` algorithm.

The following example depicts the reward result of 4 tokens with different decimals precision. We assume that the `decay` will return the value `10`.

ERC20	Decimals	Reward
Token 1	6	3
Token 2	9	5
Token 3	18	10
Token 4	24	13

All the decimals could be normalized to 18 since it represents the ERC20 token with the greatest digits.

PDT - Performant Data Type

Criticality	Minor / Informative
Location	contracts/testingDeploy/MyntfloStaking.sol#L26
Status	Unresolved

Description

The `StakedToken` struct stores staker's address. The `StakedToken` struct is part of the `Staker` struct. The `Staker` struct is pointed by the staker's mapping. The contract does not contain any other structure that is pointing directly to the `StakedToken` struct. The `StakedToken.staker` variable is used by the contract to determine if the stack record is active. This requirement could also be archived by exploiting a boolean data type.

```
struct StakedToken {
    address staker;
    uint256 tokenId;
    uint256 timeStaked;
    uint256 timeOfLastUpdate;
    address contractAddress;
}
```

Recommendation

Since the staker's address is solely used as a boolean indicator. The team is advised to use a boolean indicator rather than an address since the boolean data type will reduce the storage size and improve the gas cost.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/testingDeploy/MyntfloStaking.sol#L61,89,131,135,139,172,178,225
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
address _tokenContract
uint256 _tokenId
IERC20 _rewardsToken
address _collection
bool _elegible
bool[] memory _elegible
address[] memory _collections
address _staker
address _user
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.

Find more information on the Solidity documentation https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/testingDeploy/MyntfloStaking.sol#L2
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.4;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
ERC2771Context	Implementation	Context		
		Public	✓	-
	isTrustedForwarder	Public		-
	_msgSender	Internal		
	_msgData	Internal		
MinimalForwarder	Implementation	EIP712		
		Public	✓	EIP712
	getNonce	Public		-
	verify	Public		-
	execute	Public	Payable	-
ReentrancyGuard	Implementation			
		Public	✓	-
IERC20Permit	Interface			
	permit	External	✓	-
	nonces	External		-
	DOMAIN_SEPARATOR	External		-
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-

	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
SafeERC20	Library			
	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	safeApprove	Internal	✓	
	safeIncreaseAllowance	Internal	✓	
	safeDecreaseAllowance	Internal	✓	
	safePermit	Internal	✓	
	_callOptionalReturn	Private	✓	
IERC721	Interface	IERC165		
	balanceOf	External		-
	ownerOf	External		-
	safeTransferFrom	External	✓	-
	safeTransferFrom	External	✓	-
	transferFrom	External	✓	-
	approve	External	✓	-
	setApprovalForAll	External	✓	-
	getApproved	External		-
	isApprovedForAll	External		-
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	

	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	verifyCallResult	Internal		
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
EIP712	Implementation			
		Public	✓	-
	_domainSeparatorV4	Internal		
	_buildDomainSeparator	Private		
	_hashTypedDataV4	Internal		
ECDSA	Library			
	_throwError	Private		
	tryRecover	Internal		
	recover	Internal		
	tryRecover	Internal		
	recover	Internal		
	tryRecover	Internal		
	recover	Internal		
	toEthSignedMessageHash	Internal		
	toEthSignedMessageHash	Internal		

	toTypedDataHash	Internal		
IERC165	Interface			
	supportsInterface	External		-
Strings	Library			
	toString	Internal		
	toHexString	Internal		
	toHexString	Internal		
	toHexString	Internal		
MyntfloStaking	Implementation	ReentrancyGuard, ERC2771Context		
		Public	✓	ERC2771Context
	stake	External	✓	nonReentrant
	unstake	External	✓	nonReentrant
	setRewardsToken	External	✓	onlyOwner
	setElegibleCollection	External	✓	onlyOwner
	setElegibleCollections	External	✓	onlyOwner
	claimRewards	External	✓	-
	availableRewards	Public		-
	getStakedTokens	Public		-
	decay	Internal		
	calculateRewards	Internal		
console	Library			
	_sendLogPayload	Private		
	log	Internal		
	logInt	Internal		

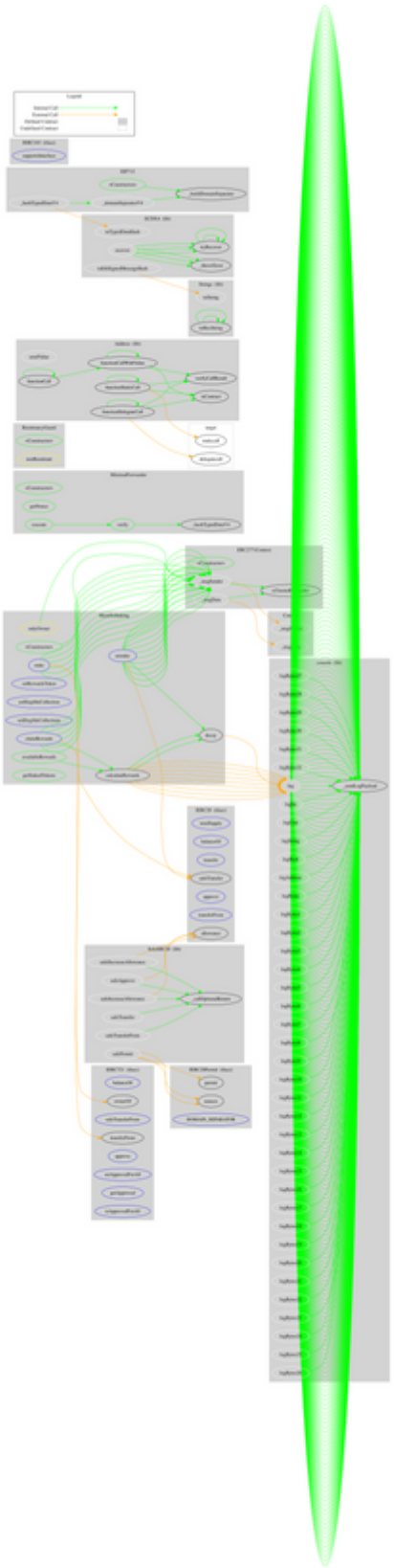
	logUint	Internal		
	logString	Internal		
	logBool	Internal		
	logAddress	Internal		
	logBytes	Internal		
	logBytes1	Internal		
	logBytes2	Internal		
	logBytes3	Internal		
	logBytes4	Internal		
	logBytes5	Internal		
	logBytes6	Internal		
	logBytes7	Internal		
	logBytes8	Internal		
	logBytes9	Internal		
	logBytes10	Internal		
	logBytes11	Internal		
	logBytes12	Internal		
	logBytes13	Internal		
	logBytes14	Internal		
	logBytes15	Internal		
	logBytes16	Internal		
	logBytes17	Internal		
	logBytes18	Internal		
	logBytes19	Internal		
	logBytes20	Internal		
	logBytes21	Internal		
	logBytes22	Internal		
	logBytes23	Internal		
	logBytes24	Internal		

	logBytes25	Internal		
	logBytes26	Internal		
	logBytes27	Internal		
	logBytes28	Internal		
	logBytes29	Internal		
	logBytes30	Internal		
	logBytes31	Internal		
	logBytes32	Internal		
	log	Internal		

Inheritance Graph



Flow Graph



Summary

Myntflo contract implements an NFT staking mechanism. This audit investigates security issues, business logic concerns and potential improvements.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope’s prior written consent. This report is not nor should be considered an “endorsement” or “disapproval” of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment 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 Cyberscope’s position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.



The Cyberscope team

https://www.cyberscope.io