Cyberscope

Audit Report

PymeDAO

August 2023

Network	ETH
Address	0xdd9eB6b26BeD3be56198649b0c0fe8302e7Fb9E8
Audited by	© cyberscope

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Acknowledged
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	MT	Mints Tokens	Acknowledged
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Acknowledged

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	MEE	Missing Events Emission	Unresolved
	RSW	Redundant Storage Writes	Unresolved
	RSK	Redundant Storage Keyword	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L09	Dead Code Elimination	Unresolved
	L13	Divide before Multiply Operation	Unresolved
	L14	Uninitialized Variables in Local Scope	Unresolved
	L15	Local Scope Variable Shadowing	Unresolved
	L17	Usage of Solidity Assembly	Unresolved
	L18	Multiple Pragma Directives	Unresolved
	L19	Stable Compiler Version	Unresolved

Table of Contents

Analysis **1**
Diagnostics **2**
Table of Contents **3**
Review **5**
Audit Updates 5
Source Files 5
Findings Breakdown **6**
ST - Stops Transactions 7
Description 7
Recommendation 7
Team Update 8
MT - Mints Tokens 9
Description 9
Recommendation 9
Team Update 9
BC - Blacklists Addresses 10
Description 10
Recommendation 10
Team Update 11
MEE - Missing Events Emission 12
Description 12
Recommendation 12
RSW - Redundant Storage Writes 13
Description 13
Recommendation 13
RSK - Redundant Storage Keyword 14
Description 14
Recommendation 14
L04 - Conformance to Solidity Naming Conventions 15
Description 15
Recommendation 16
L09 - Dead Code Elimination 17
Description 17
Recommendation 18
L13 - Divide before Multiply Operation 19
Description 19
Recommendation 19
L14 - Uninitialized Variables in Local Scope 20
Description 20

Recommendation 20
L15 - Local Scope Variable Shadowing 21
Description 21
Recommendation 21
L17 - Usage of Solidity Assembly 22
Description 22
Recommendation 22
L18 - Multiple Pragma Directives 23
Description 23
Recommendation 23
L19 - Stable Compiler Version 24
Description 24
Recommendation 24
Functions Analysis **25**
Inheritance Graph **27**
Flow Graph **28**
Summary **29**
Disclaimer **30**
About Cyberscope **31**

Review

Contract Name	PymeToken
Compiler Version	v0.8.18+commit.87f61d96
Optimization	200 runs
Explorer	https://etherscan.io/address/0xdd9eb6b26bed3be56198649b0c0fe8302e7fb9e8
Address	0xdd9eb6b26bed3be56198649b0c0fe8302e7fb9e8
Network	ETH
Decimals	18

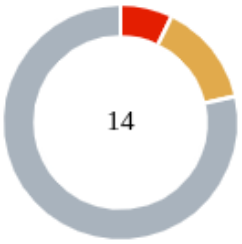
Audit Updates

Initial Audit	16 Aug 2023

Source Files

Filename	SHA256
PymeToken.sol	035416d20b9272af6989c8eb69cfd9a3861ecf783c6282a48a072792c5f2a48c

Findings Breakdown



Critical	1
Medium	2
Minor / Informative	11

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	1	0	0
Medium	0	2	0	0
Minor / Informative	11	0	0	0

ST - Stops Transactions

Criticality	Medium
Location	PymeToken.sol#L4880,4906
Status	Acknowledged

Description

The contract owner has the authority to stop the sales for all users excluding the `whitelist` address. The owner may whitelist their own addresses or those of their associates, effectively allowing only a select few to transact. As a result, the contract may operate as a honeypot.

```
    function pause() public onlyOwner {
        _pause();
    }

    function unpause() public onlyOwner {
        _unpause();
    }

    function _beforeTokenTransfer(address from, address to, uint256
amount)
        internal
        override(ERC20Upgradeable, ERC20SnapshotUpgradeable)
    {
        require(!blacklist[from] && !blacklist[to], "ERC20Pausable:
token transfer to/from blacklisted address");
        if (!whitelist[from] && !whitelist[to]) {
            require(!paused(), "ERC20Pausable: token transfer while
paused");
        }
        super._beforeTokenTransfer(from, to, amount);
    }
```

Recommendation

It is recommended that the project could consider removing the `pause` functionality from the contract entirely. 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. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

Team Update

The team has acknowledged that this is not a security issue and states:

"*Ability added-in post voting the DAO to stop trading activity until listing of the token.*"

MT - Mints Tokens

Criticality	Critical
Location	PymeToken.sol#L4896
Status	Acknowledged

Description

The contract owner has the authority to mint tokens. The owner may take advantage of it by calling the `mint` function. As a result, the contract tokens will be highly inflated.

```
    function mint(address to, uint256 amount) public onlyOwner {
        _mint(to, amount);
    }
```

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. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

Team Update

The team has acknowledged that this is not a security issue and states:

"*Contract has a hardcap of 1B tokens and only 560,000,000 tokens in circulation for the moment. This means we need the mint feature for the DAO to mint additional tokens as and when voted and needed.*"

BC - Blacklists Addresses

Criticality	Medium
Location	PymeToken.sol#L4892,4904
Status	Acknowledged

Description

The contract owner has the authority to stop addresses from transactions. The owner may take advantage of it by calling the `setBlacklist` function.

```
    function setBlacklist(address _address, bool _state) public onlyOwner
{
        blacklist[_address] = _state;
    }

  function _beforeTokenTransfer(address from, address to, uint256
amount)
        internal
        override(ERC20Upgradeable, ERC20SnapshotUpgradeable)
    {
        require(!blacklist[from] && !blacklist[to], "ERC20Pausable:
token transfer to/from blacklisted address");
        ...
    }
```

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. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

Team Update

The team has acknowledged that this is not a security issue and states:

"*Has been added on as a security fallback incase needed for compliance against malicious actors. This again is part of the DAO governance.*"

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	PymeToken.sol#L4888,4892
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
    function setWhitelist(address _address, bool _state) public
onlyOwner {
        whitelist[_address] = _state;
    }

    function setBlacklist(address _address, bool _state) public
onlyOwner {
        blacklist[_address] = _state;
    }
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	PymeToken.sol#L4888,4892
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The contract updates the `whitelist` and `blacklist` status of an account even if its current state is the same as the one passed as an argument. As a result, the contract performs redundant storage writes.

```
   function setWhitelist(address _address, bool _state) public
onlyOwner {
        whitelist[_address] = _state;
    }

    function setBlacklist(address _address, bool _state) public
onlyOwner {
        blacklist[_address] = _state;
    }
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

RSK - Redundant Storage Keyword

Criticality	Minor / Informative
Location	PymeToken.sol#L1545,2279,2289,2299,2309,2319,2329,2329,2339,2349,2349,2381,2414,2414,2432,2432,2450,2450,4374,4542,4542,4714,4757
Status	Unresolved

Description

The contract uses the `storage` keyword in a view function. The `storage` keyword is used to persist data on the contract's storage. View functions are functions that do not modify the state of the contract and do not perform any actions that cost gas (such as sending a transaction). As a result, the use of the `storage` keyword in view functions is redundant.

```
Counter storage counter
AddressSlot storage r
BooleanSlot storage r
Bytes32Slot storage r
Uint256Slot storage r
StringSlot storage r
string storage store
BytesSlot storage r
bytes storage store
uint256[] storage array
address[] storage arr
StorageSlotUpgradeable.AddressSlot storage
bytes32[] storage arr
StorageSlotUpgradeable.Bytes32Slot storage

...
```

Recommendation

It is generally considered good practice to avoid using the `storage` keyword in view functions because it is unnecessary and can make the code less readable.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	PymeToken.sol#L1309,1519,2899,2902,3048,3074,3077,3080,3162,3224,3228,3308,3318,3327,3348,3369,3391,3394,3409,3446,3450,3528,3560,3564,3625,3794,3798,4115,4127,4181,4188,4192,4229,4249,4281,4284,4309,4554,4611,4614,4770,4789,4792,4824,4882,4886
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.

```
function CLOCK_MODE() external view returns (string memory);
function DOMAIN_SEPARATOR() external view returns (bytes32);
function __ERC1967Upgrade_init() internal onlyInitializing {}
function __ERC1967Upgrade_init_unchained() internal onlyInitializing {}
uint256[50] private __gap;
...
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	PymeToken.sol#L189,206,223,257,274,291,308,325,342,359,376,393,410,427,444,461,478,495,512,529,546,563,580,597,631,665,682,699,713,731,749,767,785,803,821,839,857,875,893,911,929,947,965,983,1001,1019,1037,1055,1073,1091,1109,1127,1145,1163,1181,1199,1217,1235,1253,1271,1285,1427,1434,1442,1451,1555,1563,1588,1614,1624,1708,1757,1810,1821,1859,1872,1902,1929,1954,1961,1970,1985,1992,2052,2085,2098,2109,2162,2181,2211,2299,2319,2329,2339,2349,2414,2432,2529,2554,2564,2583,2593,2610,2620,2684,2867,2874,2899,2902,2982,2989,2999,3013,3020,3035,3077,3224,3391,3394,4192,4284,4447,4454,4531,4535,4614,4792
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function toUint248(uint256 value) internal pure returns (uint248) {
        require(value <= type(uint248).max, "SafeCast: value doesn't fit
in 248 bits");
        return uint248(value);
    }

function toUint240(uint256 value) internal pure returns (uint240) {
        require(value <= type(uint240).max, "SafeCast: value doesn't fit
in 240 bits");
        return uint240(value);
    }

function toUint232(uint256 value) internal pure returns (uint232) {
        require(value <= type(uint232).max, "SafeCast: value doesn't fit
in 232 bits");
        return uint232(value);
    }

...
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	PymeToken.sol#L1670,1673,1685,1689,1690,1691,1692,1693,1694,1700
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.

```
denominator := div(denominator, twos)
inverse *= 2 - denominator * inverse
```

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.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	PymeToken.sol#L2963,4504
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
try IERC1822ProxiableUpgradeable(newImplementation).proxiableUUID()
returns (bytes32 slot) {
(uint256 oldWeight, uint256 newWeight) =
_writeCheckpoint(_checkpoints[dst], _add, amount);
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	PymeToken.sol#L4188
Status	Unresolved

Description

Local scope variable shadowing occurs when a local variable with the same name as a variable in an outer scope is declared within a function or code block. When this happens, the local variable "shadows" the outer variable, meaning that it takes precedence over the outer variable within the scope in which it is declared.

```
    function __ERC20Permit_init(string memory name) internal
onlyInitializing {
        __EIP712_init_unchained(name, "1");
    }
```

Recommendation

It's important to be aware of shadowing when working with local variables, as it can lead to confusion and unintended consequences if not used correctly. It's generally a good idea to choose unique names for local variables to avoid shadowing outer variables and causing confusion.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	PymeToken.sol#L1631,1935,2060,2166,2196,2281,2291,2301,2311,2321,2331,2341,2351,2420,2438,2456,2701,4543
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly {
    let mm := mulmod(x, y, not(0))
    prod0 := mul(x, y)
    prod1 := sub(sub(mm, prod0), lt(mm, prod0))
    }

assembly {
    mstore(0, ckpts.slot)
    result.slot := add(keccak256(0, 0x20), pos)
    }

...
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	PymeToken.sol#L6,87,116,135,159,1297,1316,1376,1387,1418,1464,1527,1573,1915,2002,2222,2362,2469,2716,2884,3056,3170,3377,3417,3536,3633,3714,3744,4121,4146,4257,4562,4572,4778,4831
Status	Unresolved

Description

If the contract includes multiple conflicting pragma directives, it may produce unexpected errors. To avoid this, it's important to include the correct pragma directive at the top of the contract and to ensure that it is the only pragma directive included in the contract.

```
pragma solidity ^0.8.0;
...
pragma solidity ^0.8.0;
...
pragma solidity ^0.8.0;
...
pragma solidity ^0.8.9;
```

Recommendation

It is important to include only one pragma directive at the top of the contract and to ensure that it accurately reflects the version of Solidity that the contract is written in.

By including all required compiler options and flags in a single pragma directive, the potential conflicts could be avoided and ensure that the contract can be compiled correctly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	PymeToken.sol#L6,87,116,135,159,1297,1316,1376,1387,1418,1464,1527,1573,1915,2002,2222,2362,2469,2716,2884,3056,3170,3377,3417,3536,3633,3714,3744,4121,4146,4257,4562,4572,4778,4831
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 ^0.8.0;
...
pragma solidity ^0.8.0;
...
pragma solidity ^0.8.0;
...
pragma solidity ^0.8.1;
...
```

Recommendation

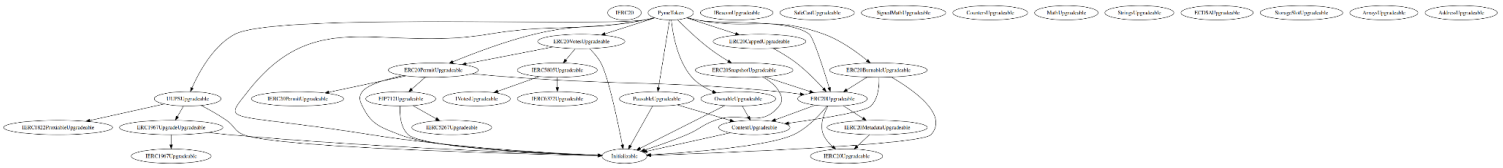
The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest 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

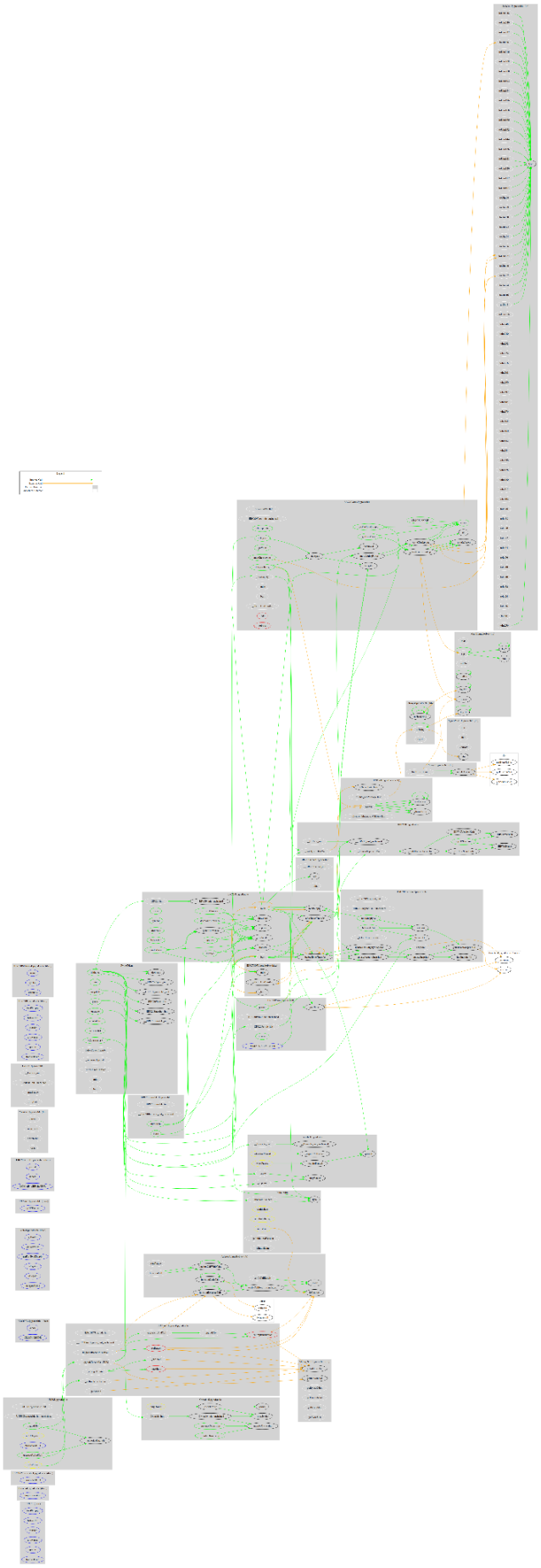
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
PymeToken	Implementation	Initializable, ERC20Upgradeable, ERC20BurnableUpgradeable, ERC20SnapshotUpgradeable, OwnableUpgradeable, PausableUpgradeable, ERC20PermitUpgradeable, ERC20VotesUpgradeable, UUPSUpgradeable, ERC20CappedUpgradeable		
		Public	✓	-
	initialize	Public	✓	initializer
	snapshot	Public	✓	onlyOwner
	pause	Public	✓	onlyOwner
	unpause	Public	✓	onlyOwner
	setWhitelist	Public	✓	onlyOwner
	setBlacklist	Public	✓	onlyOwner
	mint	Public	✓	onlyOwner
	_beforeTokenTransfer	Internal	✓	
	_authorizeUpgrade	Internal	✓	onlyOwner
	_afterTokenTransfer	Internal	✓	

	_mint	Internal	✓	
	_burn	Internal	✓	

Inheritance Graph



Flow Graph



Summary

PymeToken contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. There are some functions that can be abused by the owner like stop transactions, mint tokens and blacklist addresses. If the contract owner abuses the mint functionality, then the contract will be highly inflated. The team has acknowledged the issues. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats.

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