Cyberscope

Audit Report

PAWDNAH

June 2023

SHA256 99a1f01dad16f2505c9c45e21fb9123e68fbd54c270f6926313e6a26441536be

Audited by © cyberscope

Table of Contents

Review

Testing Deploy	https://testnet.bscscan.com/address/0x327aa300f84ebb9640de614e7fb62913534116d3

Audit Updates

Initial Audit	06 Jun 2023 https://github.com/cyberscope-io/audits/blob/main/7-burn2/v1/audit.pdf
Corrected Phase 2	14 Jun 2023

Source Files

Filename	SHA256
@openzeppelin/contracts/access/AccessControl.sol	86908de632a9fbffc04a94fa27bd320c304a47072a85de02293e08f1724934fb
@openzeppelin/contracts/access/IAccessControl.sol	d03c1257f2094da6c86efa7aa09c1c07ebd33dd31046480c5097bc2542140e45
@openzeppelin/contracts/security/Pausable.sol	2072248d2f79e661c149fd6a6593a8a3f038466557c9b75e50e0b001bcb5cf97
@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/utils/Address.sol	8160a4242e8a7d487d940814e5279d934e81f0436689132a4e73394bab084a6d
@openzeppelin/contracts/utils/Context.sol	1458c260d010a08e4c20a4a517882259a23a4baa0b5bd9add9fb6d6a1549814a
@openzeppelin/contracts/utils/introspection/ERC165.sol	8806a632d7b656cadb8133ff8f2acae4405b3a64d8709d93b0fa6a216a8a6154
@openzeppelin/contracts/utils/introspection/IERC165.sol	701e025d13ec6be09ae892eb029cd83b3064325801d73654847a5fb11c58b1e5
@openzeppelin/contracts/utils/math/Math.sol	8059d642ec219d0b9b62fbc76912079529cf494cac988abe5e371f1168b29b0f

@openzeppelin/contracts/utils/Strings.sol	f81f11dca62dcd3e0895e680559676f4ba4f2e12a36bb0291d7ecbb6b983141f
contracts/SecureDeposit.sol	99a1f01dad16f2505c9c45e21fb9123e68fbd54c270f6926313e6a26441536be

Overview

The SecureDeposit contract implements a rewards mechanism based on deposits. The users can deposit a specific amount of tokens. The deposits are tracked from the contract using a FIFO (First in First out) structure. If the total number of depositors is more than three and the contract has three times the tokens of the first depositor, then the first depositor is applicable to withdraw three times the deposited amount and dequeued from the structure. The tokens are intended to be USDC. The deposited amount is defined by the contract owner. The depositors cannot withdraw their deposits unless they are applicable.

Roles

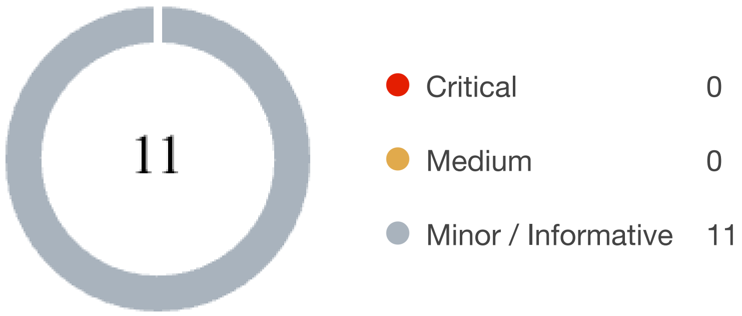
Users

- deposit
- withdraw

Admin

- pause
- unpause
- onlyRole
- addFundsToBackupWallet
- setDepositAmount
- revertState

Findings Breakdown



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

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Minor / Informative	RPC	Redundant Pause Condition	Unresolved
Minor / Informative	RC	Repetitive Calculations	Unresolved
Minor / Informative	MEE	Missing Events Emission	Unresolved
Minor / Informative	RPC	Redundant Precondition Check	Unresolved
Minor / Informative	RNRM	Redundant No Reentrant Modifier	Unresolved
Minor / Informative	PTAI	Potential Transfer Amount Inconsistency	Unresolved
Minor / Informative	RDS	Redundant Data Structure	Unresolved
Minor / Informative	IDI	Immutable Declaration Improvement	Unresolved
Minor / Informative	L04	Conformance to Solidity Naming Conventions	Unresolved
Minor / Informative	L16	Validate Variable Setters	Unresolved
Minor / Informative	L19	Stable Compiler Version	Unresolved

RPC - Redundant Pause Condition

Criticality	Minor / Informative
Location	contracts/testingDeploy/SecureDeposit.sol#L69
Status	Unresolved

Description

The method deposit contains a modifier `whenNotPaused()` which implies that the method should only execute when the contract is not in a paused state. However, the method also includes a condition check within its implementation, explicitly verifying if "!paused()" before proceeding with its logic.

This redundant condition check within the method creates unnecessary complexity and increases unnecessarily gas consumption. Since the `whenNotPaused()` modifier is already responsible for checking if the contract is not paused, the additional condition check is redundant and does not contribute to the intended functionality of the method.

```
function deposit(uint256 amount) public whenNotPaused {
...
if (
    depositQueue.length >= 3 &&
    !paused() &&
    usdc.balanceOf(address(this)) >= depositQueue[0].amount * 3
) {
```

Recommendation

To address this finding and improve code simplicity and readability, it is recommended to remove the redundant condition check within the method. By relying solely on the "whenNotPaused" modifier, the codebase becomes more concise, easier to understand, and less prone to logical errors or inconsistencies.

RC - Repetitive Calculations

Criticality	Minor / Informative
Location	contracts/testingDeploy/SecureDeposit.sol#L70
Status	Unresolved

Description

The contract contains multiple occurrences of the same calculation being performed. The calculation is repeated without utilizing a variable to store its result, which leads to redundant code, hinders code readability, and increases gas consumption. Each repetition of the calculation requires computational resources and can impact the performance of the contract, especially if the calculation is resource-intensive.

The expression `depositQueue[0].amount * 3` that is the same as `firstDeposit.amount * 3)` is getting called 4 times in the same method.

```
if (
    depositQueue.length >= 3 &&
    !paused() &&
    usdc.balanceOf(address(this)) >= depositQueue[0].amount * 3
) {
    // Send funds to withdrawal wallet
    Deposit memory firstDeposit = depositQueue[0];
    usdc.safeTransfer(withdrawalWallet, firstDeposit.amount * 3);
    // Update the balances and remove the paid-out deposit from the
queue
    withdrawalWalletBalance += (firstDeposit.amount * 3);
    eligibleWithdrawals[firstDeposit.depositor] +=
(firstDeposit.amount * 3);
    removeFirstDepositorFromQueue();
}
```

Recommendation

To address this finding and enhance the efficiency and maintainability of the contract, it is recommended to refactor the code by assigning the calculation result to a variable once and then utilizing that variable throughout the method. By storing the calculation result in a variable, the contract eliminates the need for redundant calculations and optimizes code execution.

Refactoring the code to assign the calculation result to a variable has several benefits. It improves code readability by making the purpose and intent of the calculation explicit. It also reduces code redundancy, making the method more concise, easier to maintain, and gas effective. Additionally, by performing the calculation once and reusing the variable, the contract improves performance by avoiding unnecessary computations

MEE - Missing Events Emission

Criticality	Minor / Informative
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.

When the transfer is triggered from the method `deposit()` or the `revertState()` it would be helpful to emit an event as well.

```
usdc.safeTransfer(withdrawalWallet, firstDeposit.amount * 3);
...
revertState()
```

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.

RPC - Redundant Precondition Check

Criticality	Minor / Informative
Location	contracts/testingDeploy/SecureDeposit.sol#L99
Status	Unresolved

Description

The contract contains a private method `removeFirstDepositorFromQueue()` that is invoked by `deposit()` that already checks the length of the `depositQueue` array before calling this method. The caller ensures that the array length is more than three, implying that the array is not empty.

```
if (
    depositQueue.length >= 3 &&
    !paused() &&
    usdc.balanceOf(address(this)) >= depositQueue[0].amount * 3
) {
...
require(depositQueue.length > 0, "Deposit queue is empty.");
```

Recommendation

To address this finding and improve code simplicity and clarity, it is recommended to remove the redundant empty array check within the `removeFirstDepositorFromQueue()` method. By relying on the precondition established by the caller, the codebase becomes more concise, easier to understand, and less prone to redundant or conflicting conditions.

RNRM - Redundant No Reentrant Modifier

Criticality	Minor / Informative
Location	contracts/testingDeploy/SecureDeposit.sol#L79
Status	Unresolved

Description

The contract uses the `nonReentrant` modifier to the `withdraw()` method, which suggests an intention to prevent potential reentrancy attacks. However, the `withdraw()` method exclusively deals with the `usdc` token, which is considered a trusted source within the contract.

Given that the `usdc` token is a trusted entity and no external address can be executed through the `withdraw()` method, the risk of reentrancy vulnerabilities is effectively mitigated. Consequently, the usage of the `nonReentrant` modifier becomes redundant, adding unnecessary complexity to the codebase.

```
function withdraw(uint256 amount) public whenNotPaused nonReentrant
{
    require(eligibleWithdrawals[msg.sender] >= amount,
"Insufficient eligible withdrawal balance.");
    // Transfer the requested amount to the depositor
    usdc.safeTransfer(msg.sender, amount);
    // Update the withdrawal eligibility amount
    eligibleWithdrawals[msg.sender] -= amount;
}
```

Recommendation

To address this finding and enhance code simplicity and clarity, it is recommended to remove the unnecessary "nonReentrant" modifier from the "withdraw()" method. By removing the modifier, the code becomes more streamlined and easier to comprehend, reducing the gas consuption.

PTAI - Potential Transfer Amount Inconsistency

Criticality	Minor / Informative
Location	contracts/testingDeploy/SecureDeposit.sol#L62
Status	Unresolved

Description

The `safeTransfer()` and `safeTransferFrom()` functions are used to transfer a specified amount of tokens to an address. The fee or tax is an amount that is charged to the sender of an ERC20 token when tokens are transferred to another address. According to the specification, the transferred amount could potentially be less than the expected amount. This may produce inconsistency between the expected and the actual behavior.

The following example depicts the diversion between the expected and actual amount.

Tax	Amount	Expected	Actual
No Tax	100	100	100
10% Tax	100	100	90

```
usdc.safeTransferFrom(msg.sender, address(this), netAmount);
// Update deposits mapping and depositQueue array
deposits[msg.sender] = deposits[msg.sender] + netAmount;
depositQueue.push(Deposit(msg.sender, netAmount, block.timestamp));
```

Recommendation

The team is advised to take into consideration the actual amount that has been transferred instead of the expected.

It is important to note that an ERC20 transfer tax is not a standard feature of the ERC20 specification, and it is not universally implemented by all ERC20 contracts. Therefore, the contract could produce the actual amount by calculating the difference between the transfer call.

```
Actual Transferred Amount = Balance After Transfer - Balance Before Transfer
```

RDS - Redundant Data Structure

Criticality	Minor / Informative
Location	contracts/testingDeploy/SecureDeposit.sol#L19
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 `deposits` mapping is not utilized in the contact's implementation. Hence, it is redundant.

```
mapping(address => uint256) public deposits;
```

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. It is recommended to remove redundant data structures.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	contracts/SecureDeposit.sol#L42,44,45,46,47,48
Status	Unresolved

Description

The contract declares state variables that their value is initialized once in the constructor and is not modified afterward. The `immutable` is a special declaration for this kind of state variable that saves gas when it is defined.

```
usdc
withdrawalWallet
backupWallet
maintenanceWallet
withdrawalWalletBalance
totalDeposits
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/SecureDeposit.sol#L121
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.

```
uint256 _newAmount
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	contracts/SecureDeposit.sol#L44,45,46
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.

```
withdrawalWallet = _withdrawalWallet
backupWallet = _backupWallet
maintenanceWallet = _maintenanceWallet
```

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/SecureDeposit.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 ^0.8.19;
```

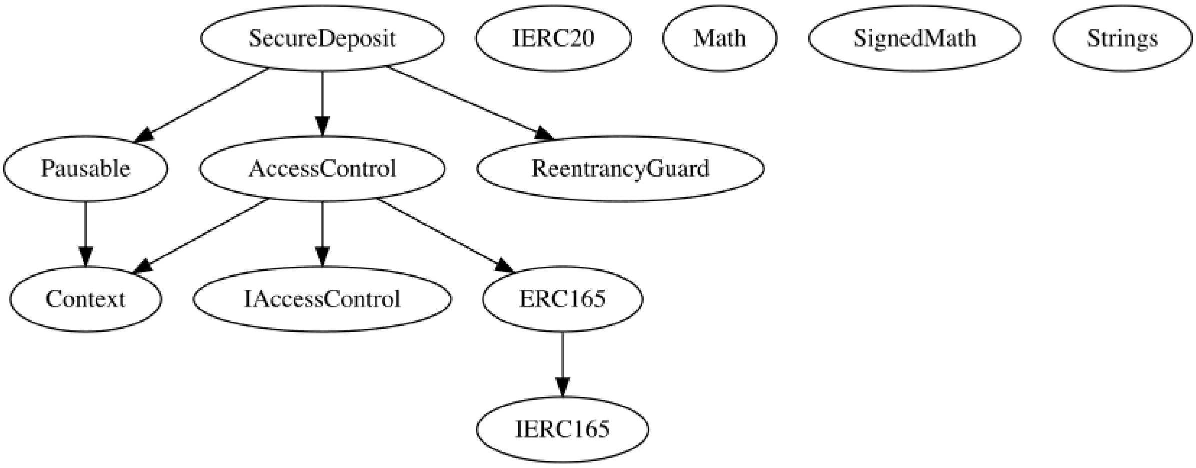
Recommendation

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.

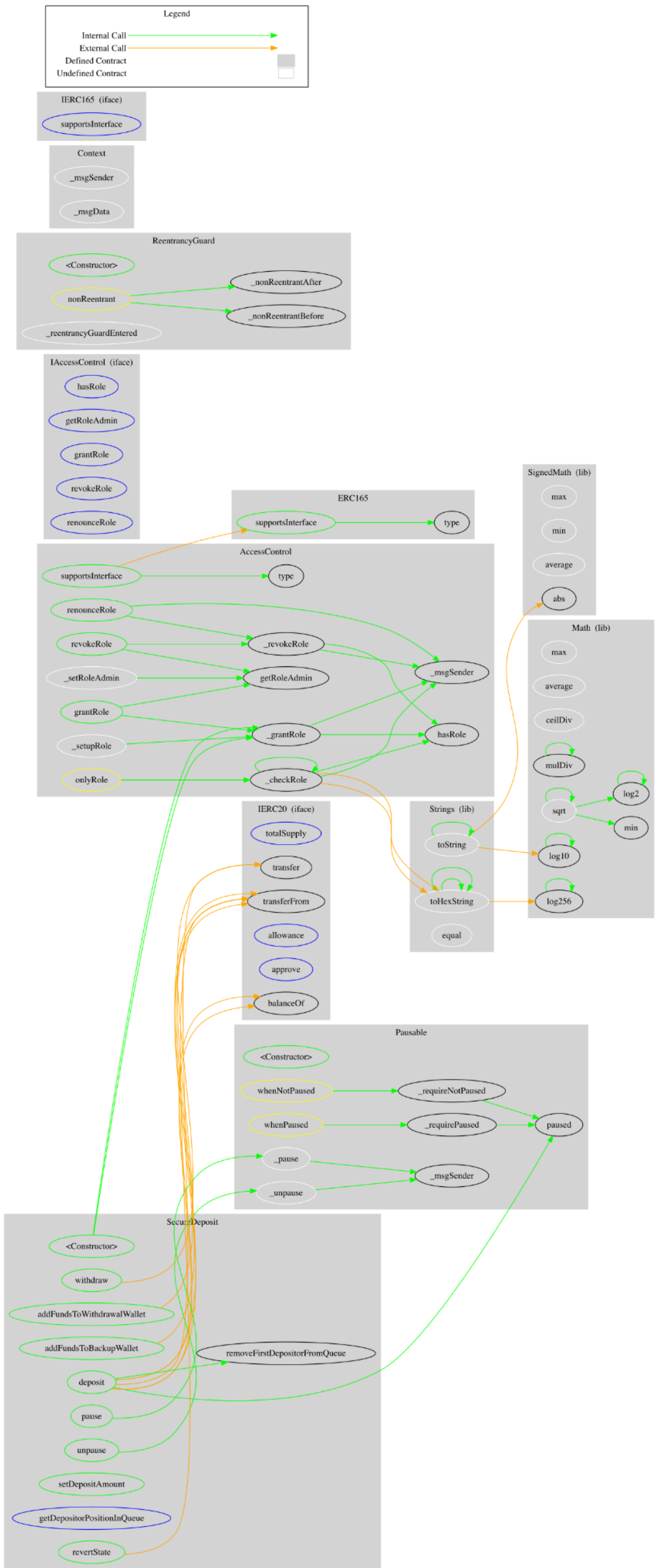
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
SecureDeposit	Implementation	AccessControl, ReentrancyGuard, Pausable		
		Public	✓	-
	deposit	Public	✓	whenNotPaused
	withdraw	Public	✓	whenNotPaused nonReentrant
	removeFirstDepositorFromQueue	Internal	✓	
	pause	Public	✓	onlyRole
	unpause	Public	✓	onlyRole
	setDepositAmount	Public	✓	onlyRole
	getDepositorPositionInQueue	External		-
	revertState	Public	✓	onlyRole

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IERC165 (iface)
supportsInterface
Context
_msgSender
_msgData
ReentrancyGuard
<Constructor>
nonReentrant
_nonReentrantAfter
_nonReentrantBefore
_reentrancyGuardEntered
IAccessControl (iface)
hasRole
getRoleAdmin
grantRole
revokeRole
renounceRole
ERC165
supportsInterface
type
SignedMath (lib)
max
min
average
abs
AccessControl
supportsInterface
type
renounceRole
_revokeRole
revokeRole
_msgSender
_setRoleAdmin
getRoleAdmin
grantRole
_grantRole
hasRole
_setupRole
onlyRole
_checkRole
Math (lib)
max
average
ceilDiv
mulDiv
log2
sqrt
min
log10
log256
IERC20 (iface)
totalSupply
transfer
transferFrom
allowance
approve
balanceOf
Strings (lib)
toString
toHexString
equal
Pausable
<Constructor>
whenNotPaused
_requireNotPaused
whenPaused
_requirePaused
paused
_pause
_msgSender
_unpause
SecureDeposit
<Constructor>
withdraw
addFundsToWithdrawalWallet
addFundsToBackupWallet
removeFirstDepositorFromQueue
deposit
pause
unpause
setDepositAmount
getDepositorPositionInQueue
revertState

Summary

Marsmello Burn contract implements a rewards mechanism based on deposits. 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