Cyberscope

Audit Report

# Noracle

August 2024

|  |  |
|---|---|
| Network | BSC |
| Address | 0x2CAF93D2E8991BD4C51435f4CD9B56F1a89B56E9 |
| Audited by | © cyberscope |

## Analysis

● Critical ● Medium ● Minor / Informative ● Pass

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

## Diagnostics

● Critical ● Medium ● Minor / Informative

| Severity | Code | Description | Status |
|---|---|---|---|
| ● | AOD | Allocation Overwrite Discrepancy | Unresolved |
| ● | CR | Code Repetition | Unresolved |
| ● | CCR | Contract Centralization Risk | Unresolved |
| ● | MU | Modifiers Usage | Unresolved |
| ● | UAR | Unclaimed Allocation Retention | Unresolved |
| ● | L08 | Tautology or Contradiction | Unresolved |
| ● | L13 | Divide before Multiply Operation | Unresolved |
| ● | L19 | Stable Compiler Version | Unresolved |

# Table of Contents

## Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

| Severity | Likelihood / Impact of Exploitation |
|---|---|
| ● Critical | Highly Likely / High Impact |
| ● Medium | Less Likely / High Impact or Highly Likely/ Lower Impact |
| ● Minor / Informative | Unlikely / Low to no Impact |

## Review

| | |
|---|---|
| **Contract Name** | Noracle |
| **Compiler Version** | v0.8.20+commit.a1b79de6 |
| **Optimization** | 200 runs |
| **Explorer** | https://bscscan.com/address/0x2caf93d2e8991bd4c51435f4cd9b56f1a89b56e9 |
| **Address** | 0x2caf93d2e8991bd4c51435f4cd9b56f1a89b56e9 |
| **Network** | BSC |
| **Symbol** | NORA |
| **Decimals** | 18 |
| **Total Supply** | 70,000,000 |
| **Badge Eligibility** | Yes |

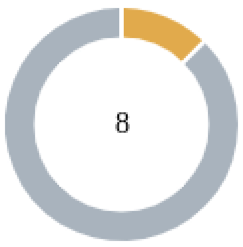
## Audit Updates

| | |
|---|---|
| **Initial Audit** | 01 Aug 2024 https://github.com/cyberscope-io/audits/blob/main/nora/v1/audit.pdf |
| **Corrected Phase 2** | 12 Aug 2024 |

## Source Files

| Filename | SHA256 |
|---|---|
| contracts/Noracle.sol | 11335b8bc08e08a00a5744fabe871c9810849379d857a12ff274a403f69c9ea9 |

## Findings Breakdown



|  |  |  |
|---|---|---|
| ● | Critical | 0 |
| ● | Medium | 1 |
| ● | Minor / Informative | 7 |

| Severity | | Unresolved | Acknowledged | Resolved | Other |
|---|---|---|---|---|---|
| ● | Critical | 0 | 0 | 0 | 0 |
| ● | Medium | 1 | 0 | 0 | 0 |
| ● | Minor / Informative | 7 | 0 | 0 | 0 |

## AOD - Allocation Overwrite Discrepancy

| | |
|---|---|
| **Criticality** | Medium |
| **Location** | contracts/Noracle.sol#L92,114,133 |
| **Status** | Unresolved |

### Description

The contract is designed to allocate tokens to specific addresses within different categories, such as investors, advisors, and airdrop beneficiaries. However, the functions responsible for setting these allocations directly assign the specified `balance` to variables `investorBalances` , `advisorBalances` and `airdropBeneficiaryBalances` , rather than incrementing the current balance. As a result, if the same function is called for an address that has already been assigned a balance, the new balance will overwrite the previous one without adjusting the total distributed tokens accordingly. This leads to discrepancies between the recorded total distributed tokens and the actual amounts allocated to each address, potentially causing misalignment in the contract's token distribution.

```
  function addInvestor(
        address investor,
        uint256 balance
    ) external onlyOwner {
        require(balance >= 0, "Noracle: Invalid balance");
        require(
            totalDistributedInvestorTokens + balance <= INVESTOR_LOCKED_SUPPLY,
            "Noracle: Exceeds investor max supply"
        );
        investors[investor] = true;
        investorBalances[investor] = balance;
        totalDistributedInvestorTokens += balance;
        investorRemainingWeeks[investor] = 24; //24 weeks;

        emit MemberAdded(investor, balance, "investor balance added");
    }

    function addAdvisor(address advisor, uint256 balance) external onlyOwner {
        require(balance >= 0, "Noracle: Invalid balance");
        require(
            totalDistributedAdvisorTokens + balance <= ADVISOR_LOCKED_SUPPLY,
            "Noracle: Exceeds advisor max supply"
        );
        advisors[advisor] = true;
        advisorBalances[advisor] = balance;
        totalDistributedAdvisorTokens += balance;
        advisorRemainingMonths[advisor] = 12; //12 months;

        emit MemberAdded(advisor, balance, "advisor balance added");
    }

    function addAirdropBeneficiary(
        address airdropBeneficiary,
        uint256 balance
    ) external onlyOwner {
        require(balance >= 0, "Noracle: Invalid balance");
        require(
            totalDistributedAirdropTokens + balance <= MAX_BALANCE_AIRDROP,
            "Noracle: Exceeds airdrop max supply"
        );
        airdropBeneficiaries[airdropBeneficiary] = true;
        airdropBeneficiaryBalances[airdropBeneficiary] = balance;
        totalDistributedAirdropTokens += balance;
        airdropBeneficiaryRemainingMonths[airdropBeneficiary] = 3; // 3x4
weeks;

        emit MemberAdded(airdropBeneficiary, balance, "airdrop beneficiary
balance added");
    }
```

## Recommendation

It is recommended to refactor the code so that, instead of setting each address's allocation to a balance variable, the allocation is incremented by the specified amount each time the function is executed. This approach will ensure that the total distributed tokens reflect the correct distribution and prevent any overwriting of previously assigned balances, maintaining the integrity of the token allocation process.

## CR - Code Repetition

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | contracts/Noracle.sol#L169,206,248 |
| **Status** | Unresolved |

### Description

The contract contains repetitive code segments. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

Specifically, the functions `claimInvestorReward` , `claimAdvisorReward` and `claimAirdropReward` share the same claim logic.

```
  function claimInvestorReward() external {
        ...
    }

    function claimAdvisorReward() external {
        ...
    }

    function claimAirdropReward() external {
        ...
    }
```

### Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help reduce the complexity and size of the contract. For instance, the contract could reuse the common code segments in an internal function in order to avoid repeating the same code in multiple places.

## CCR - Contract Centralization Risk

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | contracts/Nora.sol#L78,92,109,114,128,133,150,155,319,351 |
| **Status** | Unresolved |

### Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

Specifically, the owner has exclusive authority to set unlock times, manage investor and advisor roles, control budget allocations, and execute token transfers. This concentration of power can lead to potential misuse or arbitrary decisions without checks or balances in case where the owner's wallet is compromised, highlighting the need for governance mechanisms like multi-signature approvals or community voting to ensure a more decentralized and secure operation.

```
   function setUnlockTimes(uint256 newUnlockStartTime) external onlyOwner {
        require(
            newUnlockStartTime > block.timestamp,
            "Noracle: Invalid unlock time"
        );
        unlockStartTime = newUnlockStartTime;
        advisorUnlockStartTime = newUnlockStartTime + 365 days;
        airdropUnlockStartTime = newUnlockStartTime + 28 days;
        ecosystemUnlockStartTime = newUnlockStartTime + 180 days;
        ...
    }

    function addInvestor(
        address investor,
        uint256 balance
    ) external onlyOwner {
    ...
    }

    function removeInvestor(address investor) external onlyOwner {
        investors[investor] = false;
        emit MemberRemoved(investor, "investor removed");
    }

    function addAdvisor(address advisor, uint256 balance) external onlyOwner {
        ...
    }

    function removeAdvisor(address advisor) external onlyOwner {
        advisors[advisor] = false;
        emit MemberRemoved(advisor, "advisor removed");
    }

    function addAirdropBeneficiary(
        address airdropBeneficiary,
        uint256 balance
    ) external onlyOwner {
        ...
    }

    function removeAirdropBeneficiary(address airdropBeneficiary) external
onlyOwner {
        airdropBeneficiaries[airdropBeneficiary] = false;
        emit MemberRemoved(airdropBeneficiary, "airdrop beneficiary removed");
    }

    function setEcosystemReserve(address newEcosystemReserve) external onlyOwner
{
    ...
    }
    function setBudget(Category category, uint256 amount) external onlyOwner {
    ...
    }

    function transferNoraTokens(
        Category category,
        address recipient,
        uint256 amount
    ) external onlyOwner {
        ...
    }
}
```

## Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

## MU - Modifiers Usage

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | contracts/Noracle.sol#L96,115,137 |
| **Status** | Unresolved |

### Description

The contract is using repetitive statements on some methods to validate some preconditions. In Solidity, the form of preconditions is usually represented by the modifiers. Modifiers allow you to define a piece of code that can be reused across multiple functions within a contract. This can be particularly useful when you have several functions that require the same checks to be performed before executing the logic within the function.

```
require(balance >= 0, "Noracle: Invalid balance");
```

### Recommendation

The team is advised to use modifiers since it is a useful tool for reducing code duplication and improving the readability of smart contracts. By using modifiers to perform these checks, it reduces the amount of code that is needed to write, which can make the smart contract more efficient and easier to maintain.

## UAR - Unclaimed Allocation Retention

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | contracts/Noracle.sol#L92,109,114,128,133,150 |
| **Status** | Unresolved |

### Description

The contract is designed to manage and categorize addresses into various roles, such as investors, advisors, and airdrop beneficiaries, with corresponding token allocations. However, when an account is removed from its respective category (e.g., an investor, advisor, or airdrop beneficiary), the associated token allocation remains unadjusted. This oversight means that any unclaimed tokens allocated to the removed account are not redistributed or reallocated to other accounts or categories. As a result, these unclaimed tokens will remain locked and inaccessible, which could lead to an inefficient allocation of resources and potential issues in the overall token distribution strategy.

```
  function addInvestor(
        address investor,
        uint256 balance
    ) external onlyOwner {
        ...
        investors[investor] = true;
        investorBalances[investor] = balance;
        totalDistributedInvestorTokens += balance;
        ...
    }

    function removeInvestor(address investor) external onlyOwner {
        investors[investor] = false;
        emit MemberRemoved(investor, "investor removed");
    }

    function addAdvisor(address advisor, uint256 balance) external onlyOwner {
        ...
        advisors[advisor] = true;
        advisorBalances[advisor] = balance;
        totalDistributedAdvisorTokens += balance;
        ...
    }

    function removeAdvisor(address advisor) external onlyOwner {
        advisors[advisor] = false;
        emit MemberRemoved(advisor, "advisor removed");
    }

    function addAirdropBeneficiary(
        address airdropBeneficiary,
        uint256 balance
    ) external onlyOwner {
        ...
        airdropBeneficiaries[airdropBeneficiary] = true;
        airdropBeneficiaryBalances[airdropBeneficiary] = balance;
        totalDistributedAirdropTokens += balance;
       ...
    }

    function removeAirdropBeneficiary(address airdropBeneficiary) external
onlyOwner {
        airdropBeneficiaries[airdropBeneficiary] = false;
        emit MemberRemoved(airdropBeneficiary, "airdrop beneficiary removed");
    }
```

## Recommendation

It is recommended to reconsider the intended functionality of the contract. If an account is removed from a category, the contract should be updated to handle the redistribution of any unclaimed tokens. This could involve deducting the unclaimed amount from the removed account's allocation and appropriately redistributing it to other accounts or back to a general pool. Implementing this adjustment will ensure that the token distribution remains balanced and that no tokens are left unclaimed or locked unnecessarily.

## L08 - Tautology or Contradiction

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | contracts/Noracle.sol#L96,115,137,320 |
| **Status** | Unresolved |

### Description

A tautology is a logical statement that is always true, regardless of the values of its variables. A contradiction is a logical statement that is always false, regardless of the values of its variables.

Using tautologies or contradictions can lead to unintended behavior and can make the code harder to understand and maintain. It is generally considered good practice to avoid tautologies and contradictions in the code.

```
require(balance >= 0, "Noracle: Invalid balance")
require(amount >= 0, "Noracle: Invalid budget amount")
```

### Recommendation

The team is advised to carefully consider the logical conditions is using in the code and ensure that it is well-defined and make sense in the context of the smart contract.

## L13 - Divide before Multiply Operation

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | contracts/Noracle.sol#L181,186,192,218,227,234,260,269,276,297,304 |
| **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.

```
uint256 elapsedWeeks = (block.timestamp -
            investorLastClaimTime[msg.sender]) / 7 days
reward = elapsedWeeks * (investorBalances[msg.sender] / remainingWeeks)
elapsedWeeks = remainingWeeks < elapsedWeeks
            ? remainingWeeks
            : elapsedWeeks
```

### 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.

## L19 - Stable Compiler Version

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | contracts/Noracle.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 ^0.8.20;
```

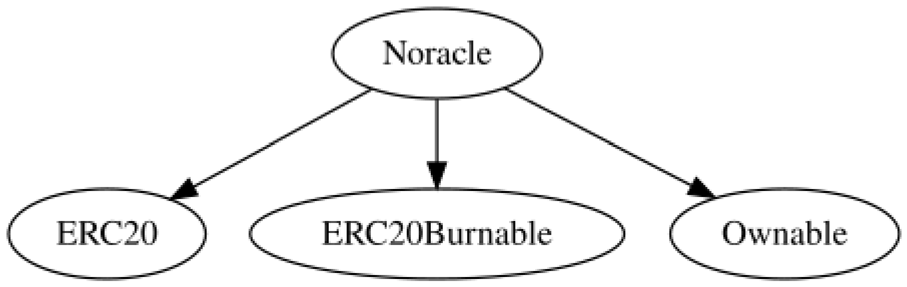
### Recommendation

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.

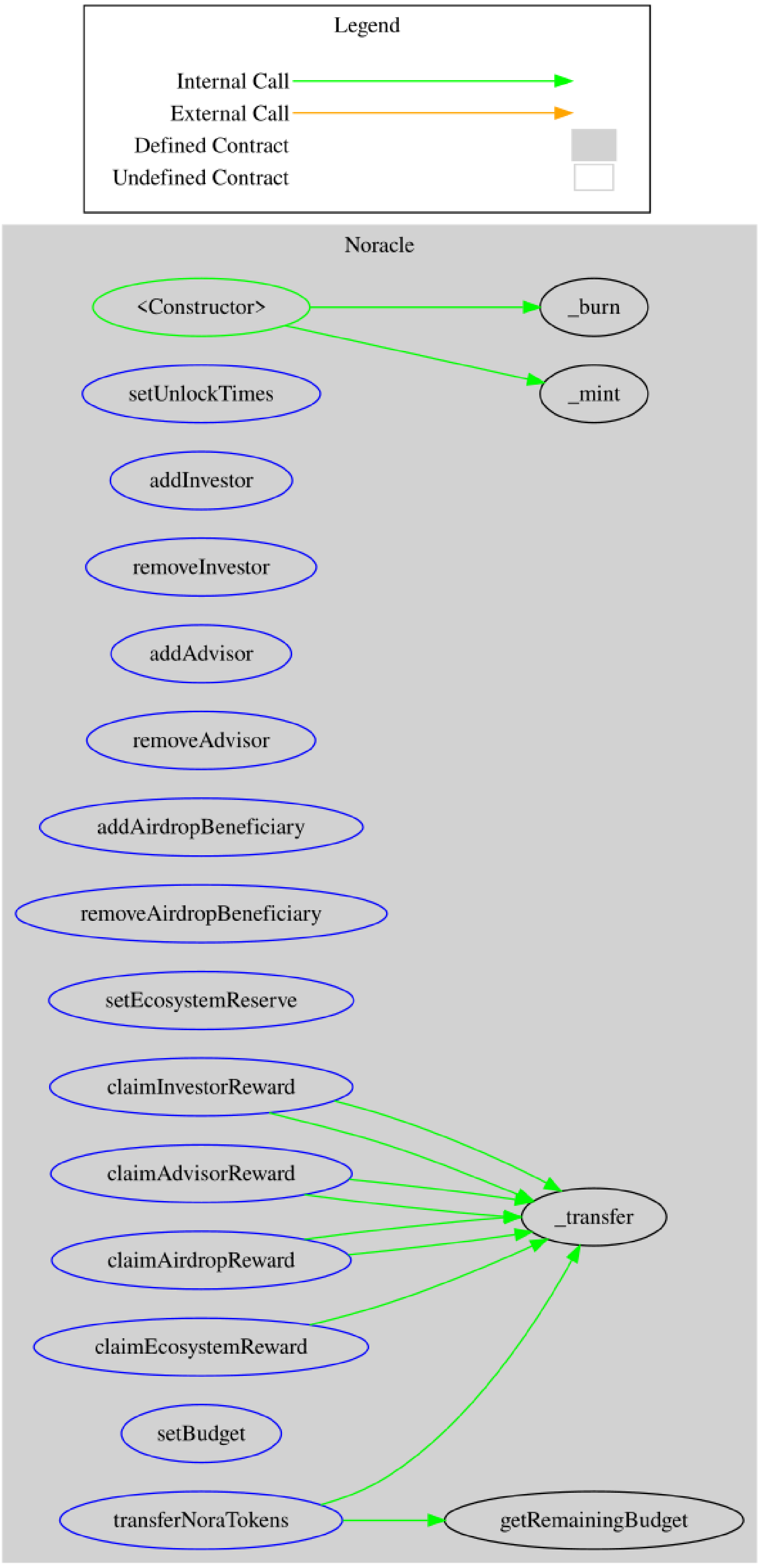
## Functions Analysis

| Contract | Type | Bases | | |
|---|---|---|---|---|
| | Function Name | Visibility | Mutability | Modifiers |
| | | | | |
| Noracle | Implementation | ERC20, ERC20Burnable, Ownable | | |
| | | Public | ✓ | ERC20 Ownable |
| | setUnlockTimes | External | ✓ | onlyOwner |
| | addInvestor | External | ✓ | onlyOwner |
| | removeInvestor | External | ✓ | onlyOwner |
| | addAdvisor | External | ✓ | onlyOwner |
| | removeAdvisor | External | ✓ | onlyOwner |
| | addAirdropBeneficiary | External | ✓ | onlyOwner |
| | removeAirdropBeneficiary | External | ✓ | onlyOwner |
| | setEcosystemReserve | External | ✓ | onlyOwner |
| | claimInvestorReward | External | ✓ | - |
| | claimAdvisorReward | External | ✓ | - |
| | claimAirdropReward | External | ✓ | - |
| | claimEcosystemReward | External | ✓ | - |
| | setBudget | External | ✓ | onlyOwner |
| | getRemainingBudget | Public | | - |
| | transferNoraTokens | External | ✓ | onlyOwner |

## Inheritance Graph



## Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Noracle
<Constructor>
_burn
_mint
setUnlockTimes
addInvestor
removeInvestor
addAdvisor
removeAdvisor
addAirdropBeneficiary
removeAirdropBeneficiary
setEcosystemReserve
claimInvestorReward
claimAdvisorReward
_transfer
claimAirdropReward
claimEcosystemReward
setBudget
transferNoraTokens
getRemainingBudget

## Summary

Noracle contract implements a token and vesting mechanism. This audit investigates security issues, business logic concerns and potential improvements. Noracle is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues. The contract Owner can access some admin functions that can not be used in a malicious way to disturb the users' transactions.

## 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**

cyberscope.io