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
Medium	ITD	Incomplete Token Distribution	Unresolved
Minor	CCR	Contract Centralization Risk	Unresolved
Minor	MVN	Misleading Variable Naming	Unresolved
Minor	MC	Missing Check	Unresolved
Minor	MEE	Missing Events Emission	Unresolved
Minor	MTC	Missing Time Check	Unresolved
Minor	RCL	Redundant Conditional Logic	Unresolved
Minor	UAS	Unused Advisor Supply	Unresolved
Minor	L02	State Variables could be Declared Constant	Unresolved
Minor	L04	Conformance to Solidity Naming Conventions	Unresolved
Minor	L08	Tautology or Contradiction	Unresolved
Minor	L13	Divide before Multiply Operation	Unresolved
Minor	L16	Validate Variable Setters	Unresolved
Minor	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

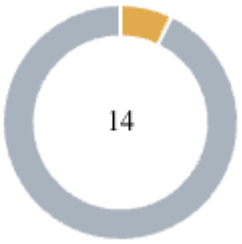
Audit Updates

Initial Audit	01 Aug 2024 https://github.com/cyberscope-io/audits/blob/main/nora/v1/audit.pdf
Corrected Phase 2	10 Aug 2024

Source Files

Filename	SHA256
hardhat/console.sol	1feb20538481d9635acd84e3395e14f7a69897531fbb7e73fe5d6f7e21405eb0
contracts/Noracle.sol	11335b8bc08e08a00a5744fabe871c9810849379d857a12ff274a403f69c9ea9
@openzeppelin/contracts/utils/Context.sol	847fda5460fee70f56f4200f59b82ae622bb03c79c77e67af010e31b7e2cc5b6
@openzeppelin/contracts/token/ERC20/IERC20.sol	6f2faae462e286e24e091d7718575179644dc60e79936ef0c92e2d1ab3ca3cee
@openzeppelin/contracts/token/ERC20/ERC20.sol	2d874da1c1478ed22a2d30dcf1a6ec0d09a13f897ca680d55fb49fbcc0e0c5b1
@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol	1d079c20a192a135308e99fa5515c27acfbb071e6cdb0913b13634e630865939
@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol	2e6108a11184dd0caab3f3ef31bd15fed1bc7e4c781a55bc867ccedd8474565c
@openzeppelin/contracts/interfaces/draft-IERC6093.sol	4aea87243e6de38804bf8737bf86f750443d3b5e63dd0fd0b7ad92f77cdbd3e3
@openzeppelin/contracts/access/Ownable.sol	38578bd71c0a909840e67202db527cc6b4e6b437e0f39f0c909da32c1e30cb81

Findings Breakdown



Critical	0
Medium	1
Minor / Informative	13

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	1	0	0	0
Minor / Informative	13	0	0	0

ITD - Incomplete Token Distribution

Criticality	Medium
Location	contracts/Nora.sol#L10,56,65,78
Status	Unresolved

Description

The contract is designed to distribute the total `MAX_SUPPLY` of 120 million tokens but fails to account for the full amount in its current distribution logic. The contract specifies:

- 50 million tokens to be burned (`INITIALLY_BURNED_SUPPLY`).
- 15 million tokens allocated for IDO (`MAX_BALANCE_IDO`).
- 10 million tokens allocated for liquidity (`MAX_BALANCE_LIQUIDITY`).
- 8 million tokens allocated for marketing purposes. (`MAX_BALANCE_MARKETING`)
- 16 million tokens allocated for ecosystem rewards. (`INITIAL_RESERVE_LOCKED`)

After these allocations, 21 million tokens remain from the original 120 million for advisor and investor allocations. However, the contract owner has the unrestricted ability to set any balance for advisors or investors, without a hard limit, which could result in the vesting of more tokens than available. This discrepancy could lead to an oversupply of tokens, adversely affecting the tokenomics and other contract functions requiring specific token amounts.

```
   uint256 public constant MAX_SUPPLY = 120_000_000 * 1e18;
    uint256 public constant INITIALLY_BURNED_SUPPLY =
50_000_000 * 1e18;
    uint256 public constant ADVISOR_LOCKED_SUPPLY = 6_000_000 *
1e18;
    uint256 public MAX_BALANCE_IDO = 15_000_000 * 1e18;
    uint256 public MAX_BALANCE_LIQUIDITY = 10_000_000 * 1e18;

    constructor(
        …
        _mint(address(this), MAX_SUPPLY);
        …
        _burn(address(this), INITIALLY_BURNED_SUPPLY);
    }

    function addInvestor(
        address _investor,
        uint256 _balance
    ) external onlyOwner {
        investors[_investor] = true;
        investorBalances[_investor] = _balance;
        investorRemainingWeeks[_investor] = 24; //24 weeks;
    }

    function addAdvisor(address _advisor, uint256 _balance)
external onlyOwner {
        advisors[_advisor] = true;
        advisorBalances[_advisor] = _balance;
        advisorRemainingMonths[_advisor] = 12; //12 months;
    }
```

Recommendation

It is recommended to implement a mechanism to track and cap the total number of tokens allocated to advisors and investors. The contract should ensure that the sum of tokens distributed does not exceed the `MAX_SUPPLY` of 120 million tokens. This will help maintain the integrity of the token distribution and prevent any potential oversupply, which could disrupt the contract's intended tokenomics.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	contracts/Nora.sol#L59,65,74,78,84,211,231
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 _unlockStartTime) external
onlyOwner {
        unlockStartTime = _unlockStartTime;
        advisorUnlockStartTime = _unlockStartTime + 365 days;
        ecosystemUnlockStartTime = _unlockStartTime + 180 days;
    }

    function addInvestor(
        address _investor,
        uint256 _balance
    ) external onlyOwner {
        investors[_investor] = true;
        investorBalances[_investor] = _balance;
        investorRemainingWeeks[_investor] = 24; //24 weeks;
    }

    function removeInvestor(address _investor) external
onlyOwner {
        investors[_investor] = false;
    }

    function addAdvisor(address _advisor, uint256 _balance)
external onlyOwner {
        advisors[_advisor] = true;
        advisorBalances[_advisor] = _balance;
        advisorRemainingMonths[_advisor] = 12; //12 months;
    }

    function removeAdvisor(address _advisor) external onlyOwner
{
        advisors[_advisor] = false;
    }

    function setBudget(Category category, uint256 amount)
external onlyOwner {
        if(category == Category.IDO) {
            require(amount <= MAX_BALANCE_IDO, "Nora: IDO
budget exceeds limit");
        }
        if(category == Category.Liquidity) {
            require(amount <= MAX_BALANCE_LIQUIDITY, "Nora:
Liquidity budget exceeds limit");
        }
        if(category == Category.Marketing) {
            require(amount <= MAX_BALANCE_MARKETING, "Nora:
Marketing budget exceeds limit");
        }

        budgets[category] = amount;
    }
```

```
    function transferNoraTokens(
        Category category,
        address recipient,
        uint256 amount
    ) external onlyOwner {
        require(
            amount <= getRemainingBudget(category),
            "Nora: Transfer amount exceeds remaining budget"
        );
        spentAmounts[category] += amount;
        _transfer(address(this), recipient, amount);
    }
```

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.

MVN - Misleading Variable Naming

Criticality	Minor / Informative
Location	contracts/Nora.sol#L179
Status	Unresolved

Description

The contract is using the variable `INITIAL_RESERVE_UNLOCKED` to track the amount of tokens that have been claimed over time. However, the variable is declared using capital letters, typically indicating a constant that should not change. This naming convention is misleading because `INITIAL_RESERVE_UNLOCKED` is being updated to reflect the total amount of tokens unlocked as claims are made. The use of a capitalized name for a mutable variable can cause confusion and misinterpretation, suggesting that the value should remain constant when it actually changes.

```
    function claimEcosystemReward() external {
        require(msg.sender == ecosystemReserve, "Nora: Not
authorized");
        require(
            block.timestamp >= ecosystemUnlockStartTime,
            "Nora: Ecosystem tokens locked"
        );

        uint256 elapsedMonths = (block.timestamp -
ecosystemUnlockStartTime) /
            30 days;
        uint256 totalUnlocked = 0;

        if (elapsedMonths >= 13) {
            totalUnlocked = INITIAL_RESERVE_LOCKED;
        } else if (elapsedMonths > 0) {
            totalUnlocked = (1_300_000 * 1e18 * elapsedMonths);
            if (elapsedMonths == 13) {
                totalUnlocked += 400_000 * 1e18;
            }
        }

        require(
            totalUnlocked > INITIAL_RESERVE_UNLOCKED,
            "Nora: No new tokens to unlock"
        );

        uint256 reward = totalUnlocked -
INITIAL_RESERVE_UNLOCKED;
        INITIAL_RESERVE_UNLOCKED = totalUnlocked;
        _transfer(address(this), ecosystemReserve, reward);

        emit ClaimedReward(msg.sender, reward, "e");
    }
```

Recommendation

It is recommended to rename the `INITIAL_RESERVE_UNLOCKED` variable to better reflect its mutable nature. A more appropriate name, would accurately indicate that this variable is updated dynamically. This change will help maintain clarity and prevent misunderstandings regarding the variable's purpose and usage in the contract.

MC - Missing Check

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

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues.

Specifically, the contract is missing a check to verify that the `amount` is greater than zero before invoke the `_transfer` function.

```
    function transferNoraTokens(
        Category category,
        address recipient,
        uint256 amount
    ) external onlyOwner {
        require(
            amount <= getRemainingBudget(category),
            "Nora: Transfer amount exceeds remaining budget"
        );
        spentAmounts[category] += amount;
        _transfer(address(this), recipient, amount);
    }
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	contracts/Nora.sol#L59,211
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 setUnlockTimes(uint256 _unlockStartTime) external
onlyOwner {
        unlockStartTime = _unlockStartTime;
        advisorUnlockStartTime = _unlockStartTime + 365 days;
        ecosystemUnlockStartTime = _unlockStartTime + 180 days;
    }

    function addInvestor(
        address _investor,
        uint256 _balance
    ) external onlyOwner {
        investors[_investor] = true;
        investorBalances[_investor] = _balance;
        investorRemainingWeeks[_investor] = 24; //24 weeks;
    }

    function removeInvestor(address _investor) external
onlyOwner {
        investors[_investor] = false;
    }

    function addAdvisor(address _advisor, uint256 _balance)
external onlyOwner {
        advisors[_advisor] = true;
        advisorBalances[_advisor] = _balance;
        advisorRemainingMonths[_advisor] = 12; //12 months;
    }

    function removeAdvisor(address _advisor) external onlyOwner
{
        advisors[_advisor] = false;
    }

    function setEcosystemReserve(address _ecosystemReserve)
external onlyOwner {
        require(
            ecosystemReserve == address(0),
            "Nora: Ecosystem reserve already set"
        );
        ecosystemReserve = _ecosystemReserve;
    }

    function setBudget(Category category, uint256 amount)
external onlyOwner {
        ...
        budgets[category] = amount;
    }
```

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.

MTC - Missing Time Check

Criticality	Minor / Informative
Location	contracts/Nora.sol#L60
Status	Unresolved

Description

The contract is missing a check within the `setUnlockTimes` function to verify that the `_unlockStartTime` is greater than the current block timestamp (`block.timestamp`). While the contract correctly implements this check during the construction phase, ensuring that the initial `unlockStartTime` is valid, the `setUnlockTimes` function does not include a similar verification. This omission could lead to setting an unlock time in the past, which might inadvertently allow immediate access to locked tokens, undermining the intended token distribution schedule and security of the contract.

```
    constructor(
        address _owner,
        uint256 _unlockStartTime
    ) ERC20("Nora", "NORA") Ownable(_owner) {
        require(_unlockStartTime > block.timestamp, "Nora:
Invalid unlock time");
        _mint(address(this), MAX_SUPPLY);
        unlockStartTime = _unlockStartTime;
        advisorUnlockStartTime = _unlockStartTime + 365 days;
        ecosystemUnlockStartTime = _unlockStartTime + 180 days;
        _burn(address(this), INITIALLY_BURNED_SUPPLY);
    }

    function setUnlockTimes(uint256 _unlockStartTime) external
onlyOwner {
        unlockStartTime = _unlockStartTime;
        advisorUnlockStartTime = _unlockStartTime + 365 days;
        ecosystemUnlockStartTime = _unlockStartTime + 180 days;
    }
```

Recommendation

It is recommended to include a verification check within the `setUnlockTimes` function to ensure that the provided `_unlockStartTime` is greater than the current `block.timestamp`. This will enforce the intended restrictions on token unlock timings and prevent the setting of past or invalid unlock times, maintaining the integrity of the contract's token distribution mechanism.

RCL - Redundant Conditional Logic

Criticality	Minor / Informative
Status	Unresolved

Description

The contract is designed to calculate and unlock tokens based on elapsed months using a conditional structure. However there is a logic error in how conditions are handled for the case when `elapsedMonths` equals 13. The initial condition checks if `elapsedMonths` is greater than or equal to 13, setting `totalUnlocked` to `INITIAL_RESERVE_LOCKED`. Consequently, the nested if statement within the else-if block, intended to add an additional amount when `elapsedMonths` is exactly 13, becomes redundant and unreachable. This setup means that the additional logic specified for exactly 13 months will never be executed due to the earlier broader condition capturing this scenario.

```
if (elapsedMonths >= 13) {
    totalUnlocked = INITIAL_RESERVE_LOCKED;
} else if (elapsedMonths > 0) {
    totalUnlocked = (1_300_000 * 1e18 * elapsedMonths);
    if (elapsedMonths == 13) {
        totalUnlocked += 400_000 * 1e18;
    }
}
```

Recommendation

It is recommended to reconsider and restructure the logic to ensure that all conditions are reachable and function as intended. Specifically, remove the unreachable if statement within the else-if block that checks again for `elapsedMonths` being 13. This will simplify the code, eliminate redundancy, and make the contract's behavior easier to understand and maintain. Adjusting these conditions will ensure that each specific scenario regarding token unlocks is appropriately and uniquely handled.

UAS - Unused Advisor Supply

Criticality	Minor / Informative
Location	contracts/Nora.sol#L12
Status	Unresolved

Description

The contract is declaring a constant variable `ADVISOR_LOCKED_SUPPLY` with a value of 6,000,000 tokens. However, this variable is not utilized anywhere within the contract's logic or functions. This setup is redundant as the declared advisor locked supply does not influence any operational or functional aspects of the contract. Such a configuration may lead to confusion or misinterpretation by stakeholders, as it suggests an allocation that is not practically enforced or implemented.

```
uint256 public constant ADVISOR_LOCKED_SUPPLY = 6_000_000 *
1e18;
```

Recommendation

It is recommended to either utilize the `ADVISOR_LOCKED_SUPPLY` variable in the contract's logic, such as incorporating it into functions or mechanisms related to advisor allocations or token distribution restrictions, or to remove the variable entirely if it serves no purpose. This will help in maintaining a clean and efficient codebase, avoiding potential confusion and ensuring that all declared elements have a clear and meaningful role in the contract's operation.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	contracts/Nora.sol#L13,14,15,20
Status	Unresolved

Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
uint256 public MAX_BALANCE_IDO = 15_000_000 * 1e18
uint256 public MAX_BALANCE_LIQUIDITY = 10_000_000 * 1e18
uint256 public MAX_BALANCE_MARKETING = 8_000_000 * 1e18
uint256 public INITIAL_RESERVE_LOCKED = 16_000_000 * 1e18
```

Recommendation

Constant state variables can be useful when the contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behavior, such as the contract's address or the maximum number of times a certain function can be called. The team is advised to add the constant keyword to state variables that never change.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/Nora.sol#L13,14,15,20,21,59,66,67,74,78,84,88
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 public MAX_BALANCE_IDO = 15_000_000 * 1e18
uint256 public MAX_BALANCE_LIQUIDITY = 10_000_000 * 1e18
uint256 public MAX_BALANCE_MARKETING = 8_000_000 * 1e18
uint256 public INITIAL_RESERVE_LOCKED = 16_000_000 * 1e18
uint256 public INITIAL_RESERVE_UNLOCKED = 0
uint256 _unlockStartTime
address _investor
uint256 _balance
address _advisor
address _ecosystemReserve
```

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/stable/style-guide.html#naming-conventions.

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/Nora.sol#L108,113,119,147,156,163,186,193
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 =
                (investorBalances[msg.sender] / remainingWeeks) *
                elapsedWeeks
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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	contracts/Nora.sol#L93
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.

```
ecosystemReserve = _ecosystemReserve
```

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/Nora.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 ^0.8.0;
```

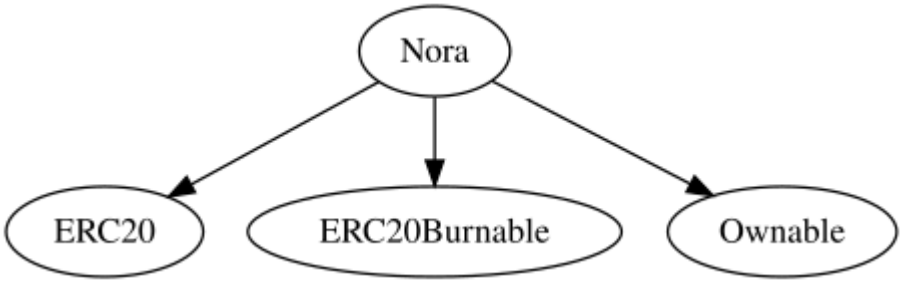
Recommendation

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.

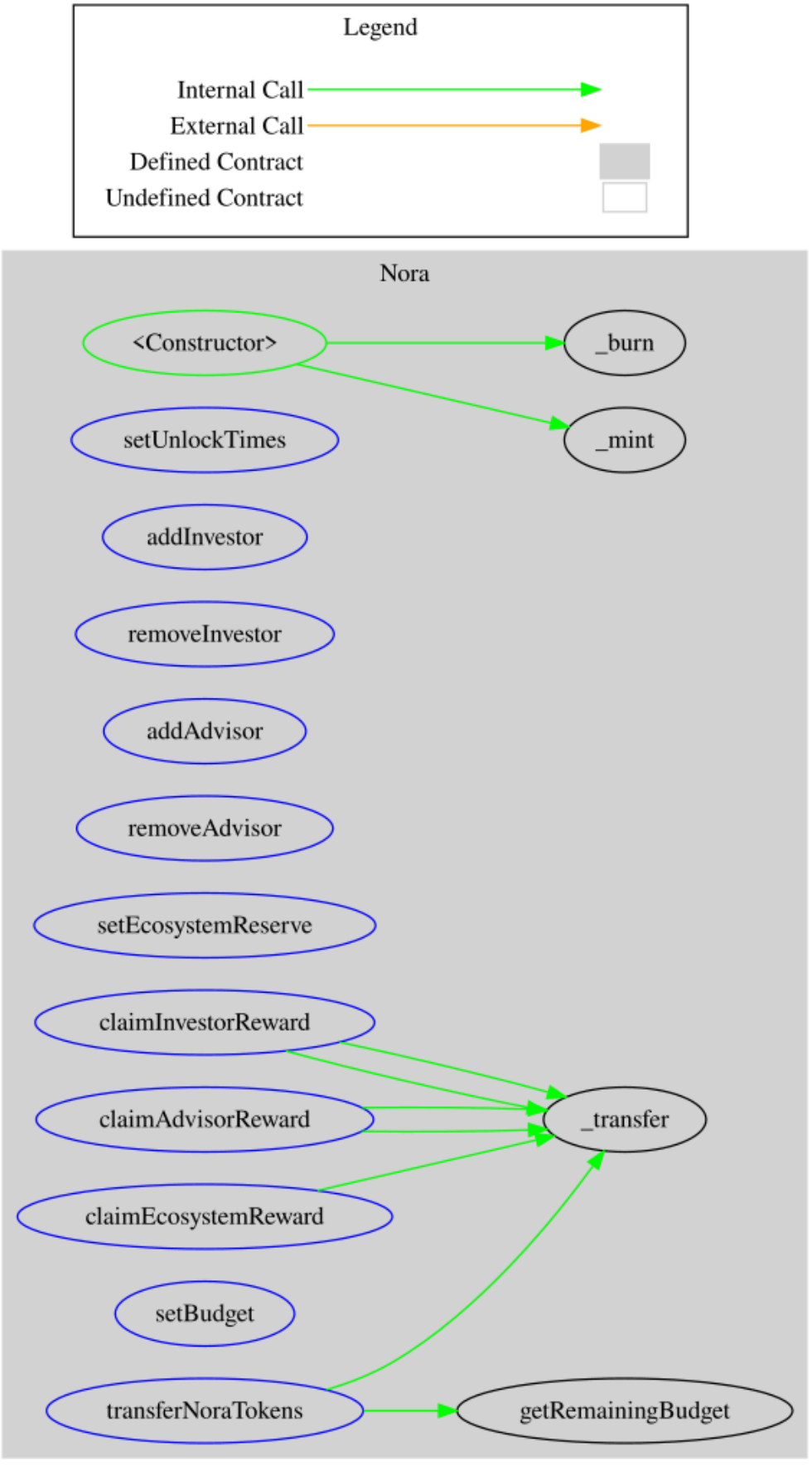
Functions Analysis

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

Inheritance Graph



Flow Graph


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

Summary

Noracle contract implements a token 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 errors 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