Cyberscope

A TAC Security Company

Audit Report

UnCensored Waves

June 2025

Files controller.sol, devTeamVesting.sol ,naun.sol, nau.sol, nauy.sol, stakingRewards.sol

Audited by © cyberscope

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

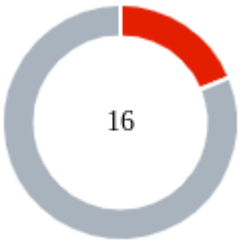
Audit Updates

Initial Audit	06 Jun 2025

Source Files

Filename	SHA256
stakingRewards.sol	f52c64ca15a3fe755fed5062216a7be93a6f5d69e7ece6fdeaeb51cafbe86e03
nauy.sol	34690b65917e6a95539f0d748842cef53210ff67231b872289a4c2ae98cdfb42
naun.sol	300c37df240c4b7777b0900b5519347b59a444948e6dc0ace1d5f7fd8c97d56e
nau.sol	4bab9b29bcb4a301d09a7e4f0465d17e5a99c04e2c4268c2a7c3a71d886765ac
devTeamVesting.sol	5ee381d2f0e274a935854918b218175a7d5e7ab2c562390e8c13ce6759af9533
controller.sol	8a7c85430e4cc6dde29fcb719614a0d208ff491cbf8d8d2b02cf812651659165

Findings Breakdown



Critical	3
Medium	0
Minor / Informative	13

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	3	0	0	0
Medium	0	0	0	0
Minor / Informative	13	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	BT	Burns Tokens	Unresolved
●	MT	Mints Tokens	Unresolved
●	ST	Stops Transactions	Unresolved
●	CO	Code Optimization	Unresolved
●	ERT	Early Release Timestamp	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	PAMAR	Pair Address Max Amount Restriction	Unresolved
●	POSD	Potential Oracle Stale Data	Unresolved
●	PPM	Potential Price Manipulation	Unresolved
●	PTAI	Potential Transfer Amount Inconsistency	Unresolved
●	RAR	Redundant Admin Role	Unresolved
●	TSI	Tokens Sufficiency Insurance	Unresolved

●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved

BT - Burns Tokens

Criticality	Critical
Location	nau.sol#L111
Status	Unresolved

Description

The `DEFAULT_ADMIN_ROLE` authority has the priviledges to burn tokens from a specific address. The owner may take advantage of it by calling the `controllerBurn` function having assigned the `CONTROLLER_ROLE` to an own address. As a result, the targeted addresses will lose the corresponding tokens.

```
function controllerBurn(address account, uint256 amount) external
onlyRole(CONTROLLER_ROLE) {
_burn(account, amount);
}
```

Recommendation

The team should carefully manage the private keys of the DEFAULT_ADMIN_ROLE account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the authority, which will eliminate the threats but it is non-reversible.

MT - Mints Tokens

Criticality	Critical
Location	naun.sol#L45 nauy.sol#L45
Status	Unresolved

Description

The `DEFAULT_ADMIN_ROLE` authority has the priviledge to mint tokens. The admin may take advantage of it by calling the `controllerMint` function, having granted the `MINTER_ROLE` to an owned address. As a result, the contract tokens will be highly inflated.

```
function controllerMint(address to, uint256 amount) external
onlyRole(MINTER_ROLE) {
_mint(to, amount);
}
```

Recommendation

The team should carefully manage the private keys of the `DEFAULT_ADMIN_ROLE` account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the authority, which will eliminate the threats but it is non-reversible.

ST - Stops Transactions

Criticality	Critical
Location	nau.sol#L82 naun.sol#L39 nauy.sol#L39
Status	Unresolved

Description

The `DEFAULT_ADMIN_ROLE` authority has the privilege to stop the sales for all users. The owner may take advantage of it by appending a false flag for the `isExcludedFromMaxWallet` mapping of the pair. As a result, the contract may operate as a honeypot. In addition, the contract implements a cooldown period of 60 seconds for all transfers.

```
if(from != owner() && to != owner()) {
  require(amount <= _maxTxAmount, "Transfer amount exceeds the
maxTxAmount.");
}
```

```
if (
from != address(0) && from != controller && !hasRole(DEFAULT_ADMIN_ROLE, from)
&& !isExcludedFromCooldown[from]
) {
require(block.timestamp - lastTxTimestamp[from] >= COOLDOWN_TIME, "Cooldown in
effect");
lastTxTimestamp[from] = block.timestamp;
}
```

Recommendation

The contract could embody a check for not allowing revoking the `isExcludedFromMaxWallet` flag from the pair. The team should carefully manage the private keys of the `DEFAULT_ADMIN_ROLE` account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the authority, which will eliminate the threats but it is non-reversible.

CO - Code Optimization

Criticality	Minor / Informative
Location	stakingRewards.sol#L68
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. In particular, the contract performs a verification for the `_stakingTokenInstance` to be included in the array of `_allowedStakingTokens`. However, both `_stakingTokenInstance` and `_allowedStakingTokens` are user provided variables, deeming this check redundant. Furthermore, the `_allowedStakingTokens` array is not utilized in any other point of the codebase.

```
bool allowed = false;
for (uint256 i = 0; i < _allowedStakingTokens.length; i++) {
if (_stakingTokenInstance == _allowedStakingTokens[i]) {
allowed = true;
break;
}
}
require(allowed, "SR: Staking token not in allowed list");
```

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.

ERT - Early Release Timestamp

Criticality	Minor / Informative
Location	stakingRewards.sol#L133
Status	Unresolved

Description

The staking mechanism implements a vesting period for tokens staked through the contract. The locup period is initiated when a user makes their first stake, but is not updated for subsequent stakes. Consequently tokens staked in later timestamps may be released earlier than intended.

```
if (balanceOf[msg.sender] == 0) {
lockupReleaseTime[msg.sender] = block.timestamp + LOCKUP_PERIOD;
}
```

Recommendation

The implementation should be revised to ensure it aligns with the intended release mechanism. Tokens should be released at a fixed interval from the moment of staking. This will ensure consistency and prevent users from manipulating the system in an attempt to extract value.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	nau.sol#L44,45,46 controller.sol#L46
Status	Unresolved

Description

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

```
reserveWallet
developerWallet
stakingOpsWallet
```

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.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	controller.sol#L48,57 naun.sol#L30 nau.sol#L58,64,70,76 nauy.sol#L30
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 setQuoteToken(address _quoteToken) external
onlyRole(DEFAULT_ADMIN_ROLE) {
require(_quoteToken != address(0), "Controller: Invalid quote token address");
quoteTokenAddress = _quoteToken;
}
```

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.

PAMAR - Pair Address Max Amount Restriction

Criticality	Minor / Informative
Location	nau.sol#L103
Status	Unresolved

Description

The contract is configured to enforce a maximum token accumulation limit through checks. This mechanism aims to prevent excessive token concentration by reverting transactions that overcome the specified cap. However, this functionality encounters issues when transactions default to the pair address during sales. If the pair address is not listed in the exceptions, then the sale transactions are inadvertently stopped, effectively disrupting operations and making the contract susceptible to unintended behaviors akin to a honeypot.

```
if (to != address(0) && !isExcludedFromMaxWallet[to]) {
require(
balanceOf(to) <= (INITIAL_SUPPLY * MAX_WALLET_PERCENT) /
BASIS_POINTS_DIVISOR,
"Recipient exceeds max wallet limit"
);
}
```

Recommendation

It is advised to modify the contract to ensure uninterrupted operations by either permitting the pair address to exceed the established token accumulation limit or by safeguarding its status in the exception list. By recognizing and allowing these essential addresses the flexibility to hold more tokens than typical limits, the contract can maintain seamless transaction flows and uphold the liquidity and stability of the ecosystem. This modification is vital for avoiding disruptions that could impact the functionality and security of the contract.

POSD - Potential Oracle Stale Data

Criticality	Minor / Informative
Location	controller.sol#L78
Status	Unresolved

Description

The contract relies on retrieving price data from an oracle. However, it lacks proper checks to ensure the data is not stale. The absence of these checks can result in outdated price data being trusted, potentially leading to significant financial inaccuracies.

```
(int56[] memory tickCumulatives,) = pool.observe(secondsAgos);
```

Recommendation

To mitigate the risk of using stale data, it is recommended to implement checks on the round and period values returned by the oracle's data retrieval function. The value indicating the most recent round or version of the data should confirm that the data is current. Additionally, the time at which the data was last updated should be checked against the current interval to ensure the data is fresh. For example, consider defining a threshold value, where if the difference between the current period and the data's last update period exceeds this threshold, the data should be considered stale and discarded, raising an appropriate error.

For contracts deployed on Layer-2 solutions, an additional check should be added to verify the sequencer's uptime. This involves integrating a boolean check to confirm the sequencer is operational before utilizing oracle data. This ensures that during sequencer downtimes, any transactions relying on oracle data are reverted, preventing the use of outdated and potentially harmful data.

By incorporating these checks, the smart contract can ensure the reliability and accuracy of the price data it uses, safeguarding against potential financial discrepancies and enhancing overall security.

PPM - Potential Price Manipulation

Criticality	Minor / Informative
Location	controller.sol#L161,162
Status	Unresolved

Description

The contract implements mint and burn operations involving decentralized pools. This design potentially enables price manipulation across different trading pairs to extract value. Such mechanisms are often prone to exploitation, as the token supply depends on the assets in the pool, which in turn are cyclically dependent on the token's price.

```
uint256 amountNAUY = FullMath.mulDiv(valueNAUYInQuote, 1e18, priceNAUY);
uint256 amountNAUN = FullMath.mulDiv(valueNAUNInQuote, 1e18, priceNAUN);
```

Recommendation

The team is advised to refrain from such designs. Instead, it is recommended to leverage the functionalities of the decentralized exchange to perform operations that affect the token supply.

PTAI - Potential Transfer Amount Inconsistency

Criticality	Minor / Informative
Location	stakingRewards.sol#L140
Status	Unresolved

Description

The `transfer()` and `transferFrom()` 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

```
totalSupply += _amount;
balanceOf[msg.sender] += _amount;
stakingToken.safeTransferFrom(msg.sender, address(this), _amount);
```

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
```

RAR - Redundant Admin Role

Criticality	Minor / Informative
Location	devTeamVesting.sol#L47
Status	Unresolved

Description

The contract implements access control through the `DEFAULT_ADMIN_ROLE`. However, this declaration may be redundant, as there are no functions in the contract that require administrator control. Eliminating redundancies will reduce code size and optimize overall readability and maintainability of the contract.

```
_grantRole(DEFAULT_ADMIN_ROLE, _admin);
```

Recommendation

The team is advised to eliminate redundancies to improve overall code size and consistency.

TSI - Tokens Sufficiency Insurance

Criticality	Minor / Informative
Location	devTeamVesting.sol stakingRewards.sol
Status	Unresolved

Description

The tokens are not held within the contract itself. Instead, the contract is designed to provide the tokens from an external administrator. While external administration can provide flexibility, it introduces a dependency on the administrator's actions, which can lead to various issues and centralization risks.

```
xToken.safeTransfer(devBeneficiary, _amount);
```

```
function fundRewards(uint256 _amount) external onlyRole(FUNDER_ROLE) {
require(_amount > 0, "SR: Cannot fund 0");
// Assumes the FUNDER (msg.sender) has been approved by the source wallet
// or the FUNDER *is* the source wallet and approved this contract.
// Standard: Pull from msg.sender, requires caller to have
funds/allowance.
rewardToken.safeTransferFrom(msg.sender, address(this), _amount);
emit RewardsFunded(_amount);
}
```

Recommendation

It is recommended to consider implementing a more decentralized and automated approach for handling the contract tokens. One possible solution is to hold the tokens within the contract itself. If the contract guarantees the process it can enhance its reliability, security, and participant trust, ultimately leading to a more successful and efficient process.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	controller.sol#L30
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.

```
uint32 public twapInterval = 60
```

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	stakingRewards.sol#L120,133,147,196,205,215 nau.sol#L58 devTeamVesting.sol#L81 controller.sol#L57
Status	Unresolved

Description

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

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

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

```
address _account
uint256 _amount
uint256 _rewardRate
address _to
address _controller
address _quoteToken
```

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.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	controller.sol#L101,102
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.

```
uint256 baseRatio_18
int256 adjustment_exponent
```

Recommendation

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

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
StakingRewards	Implementation	AccessControl, ReentrancyGuard		
		Public	✓	-
	_lastTimeRewardApplicable	Internal		
	rewardPerToken	Public		-
	earned	Public		-
	stake	External	✓	nonReentrant updateReward
	unstake	Public	✓	nonReentrant updateReward
	claimRewards	Public	✓	nonReentrant updateReward
	setRewardRate	External	✓	onlyRole
	fundRewards	External	✓	onlyRole
	recoverExcessRewardTokens	External	✓	onlyRole
NAUY	Implementation	ERC20, ERC20Burnable, AccessControl		
		Public	✓	ERC20
	setIsExcludedFromCooldown	External	✓	onlyRole
	_update	Internal	✓	

	controllerMint	External	✓	onlyRole
NAUN	Implementation	ERC20, ERC20Burnable, AccessControl		
		Public	✓	ERC20
	setIsExcludedFromCooldown	External	✓	onlyRole
	_update	Internal	✓	
	controllerMint	External	✓	onlyRole
NAU	Implementation	ERC20, ERC20Burnable, AccessControl		
		Public	✓	ERC20
	setController	External	✓	onlyRole
	setIsExcludedFromMaxWallet	External	✓	onlyRole
	setIsExcludedFromCooldown	External	✓	onlyRole
	setIsExcludedFromMaxTx	External	✓	onlyRole
	_update	Internal	✓	
	controllerBurn	External	✓	onlyRole
DevTeamVesting	Implementation	AccessControl		
		Public	✓	-
	vestedAmount	Public		-
	claimVestedTokens	External	✓	-

	claimableAmount	Public		-
INAU	Interface	IERC20		
	burn	External	✓	-
	burnFrom	External	✓	-
INAUXMintable	Interface	IERC20		
	controllerMint	External	✓	-
Controller	Implementation	AccessControl		
		Public	✓	-
	setQuoteToken	External	✓	onlyRole
	setPool	External	✓	onlyRole
	getTwapPrice	Public		-
	transformX	External	✓	-

Summary

UnCensored Waves contracts implement a token, staking and vesting mechanism. This audit investigates security issues, business logic concerns and potential improvements.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope’s prior written consent. This report is not nor should be considered an “endorsement” or “disapproval” of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope’s position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

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

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

A **TAC Security** Company

The Cyberscope team

cyberscope.io