Cyberscope

Audit Report

NOCAP

May 2025

Network BASE

Address 0x3Ce2cf27aC0d499898822DC92c4eBDa952B68545

Address 0x94a808F903bEeFA4fEE8f4c09e52d7407d771068

Source Files

SHA256 d0708f420506258aab2d5b72dae4878db6ad8f224e220b9141a184946011783d

Audited by © cyberscope

Analysis

Critical Medium Minor / Informative Pass

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

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Medium	IBC	Ineffective Batch Claims	Unresolved
Medium	MUI	Missing User Incentive	Unresolved
Medium	UFCD	Unfair First Claim Drain	Unresolved
Minor / Informative	CCR	Contract Centralization Risk	Unresolved
Minor / Informative	DDP	Decimal Division Precision	Unresolved
Minor / Informative	DRP	Dual Reward Payouts	Acknowledged
Minor / Informative	MEM	Missing Error Messages	Unresolved
Minor / Informative	MEE	Missing Events Emission	Unresolved
Minor / Informative	MMC	Missing Minimum Check	Unresolved
Minor / Informative	MU	Modifiers Usage	Unresolved
Minor / Informative	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
Minor / Informative	PTRP	Potential Transfer Revert Propagation	Unresolved
Minor / Informative	RASI	Redundant Amount Swap Inclusion	Unresolved
Minor / Informative	RDO	Redundant Deployer Ownership	Unresolved

●	RSML	Redundant SafeMath Library	Unresolved
●	STPB	Short Tier Price Bypass	Unresolved
●	UBV	Unoptimized Batch Validation	Unresolved
●	ZAB	Zero Amount Burn	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	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

NoCap Contract

Contract Name	NoCap
Compiler Version	v0.8.19+commit.7dd6d404
Optimization	200 runs
Explorer	https://basescan.org/address/0x3ce2cf27ac0d499898822dc92c4ebda952b68545
Address	0x3ce2cf27ac0d499898822dc92c4ebda952b68545
Network	BASE
Symbol	NOCAP
Decimals	18
Total Supply	100,000,000
Badge Eligibility	Must Fix Criticals

NOCAPPERS Contract

Contract Name	NOCAPPERS
Explorer	https://basescan.org/address/0xdef7d4fc0cffa91fd0c37b816cbced8a1f9c3de8
Address	0xDEF7d4fC0CFfa91fd0C37B816cbCed8a1F9C3de8
Network	BASE

NoCapTreasury Contract

Contract Name	NoCapTreasury
SHA256	d0708f420506258aab2d5b72dae4878db6ad8f224e220b9141a184946011783d
Test Deploy	https://testnet.bscscan.com/address/0x7aa2c95cbce0a9e1fd949567895e517c2d1b4824
Test Address	0x7AA2c95CBcE0A9E1FD949567895E517C2d1b4824
Testnet	BSC TESTNET

Audit Updates

Initial Audit	13 May 2025
Corrected Phase 2	21 May 2025

Source Files

Filename	SHA256
NOCAP.sol	e458aac95924a3ad7451ee690ec679d3d966fd59416b7e6771a0a84a37f4be96
NOCAPPERS.sol	2a1b8527b8e2a1de4e0738eec5a2084dbc0d2e24cc1a24fb26d2b749d6976b0b
Treasury.sol	d0708f420506258aab2d5b72dae4878db6ad8f224e220b9141a184946011783d

Overview

NoCap contract

The `NoCap` contract is a standard ERC-20 token designed for deployment on the Base network, incorporating basic fee-on-transfer mechanics and UniswapV2 integration. It applies configurable buy and sell fees which are allocated to liquidity, team, treasury, staking, and NFT reward mechanisms. The contract supports automated swapping of collected fees into ETH, liquidity provision, and forwarding of funds to relevant contracts and wallets. It includes standard features such as trading activation control, exclusion lists for fees and transaction limits, early trading protections, and administrative functions for updating parameters, retrieving stuck tokens, and burning tokens.

NoCapTreasury contract

The `NoCapTreasury` contract serves as a treasury vault for the NOCAP token, allowing users to redeem their tokens for a share of backing assets after a 180-day lock period. Upon redemption, the contract burns the user's NOCAP tokens and transfers back a proportional share (90%) of its WETH and NOCAP/ETH LP token holdings based on total supply. The contract also supports wrapping received ETH into WETH and includes an owner-only function to recover non-core tokens accidentally sent to the treasury.

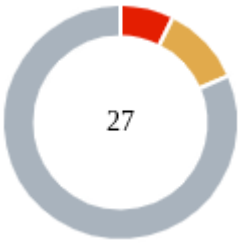
NOCAPPERS contract

The `NOCAPPERS` contract is an ERC721 NFT implementation that introduces a capped collection of 5,000 NFTs with tiered mint pricing and built-in reward distribution mechanics. Minting is subject to supply and per-transaction limits, and the proceeds are split evenly between a team wallet and a treasury wallet. The pricing structure adapts based on total supply, gradually increasing across four tiers as more NFTs are minted. Minting is also pausable by the contract owner, providing administrative flexibility.

A core feature of this contract is the founders' earnings system, which allows NFT holders to claim rewards in the form of the NoCap ERC20 token. Rewards are distributed proportionally based on a global `founderTokenBag` index, which tracks accumulated earnings. Each NFT’s withdrawn rewards are individually tracked, and users can claim rewards either one-by-one or in batch. The contract also allows the designated token contract or external users

to deposit tokens, increasing the reward pool. Additional administrative functions include metadata configuration, pausing minting, and wallet updates. The contract also inherits voting functionality via `ERC721Votes`, enabling governance extensions if required.

Findings Breakdown



Critical	2
Medium	3
Minor / Informative	22

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	2	0	0	0
Medium	3	0	0	0
Minor / Informative	21	1	0	0

ST - Stops Transactions

Criticality	Critical
Location	NOCAP.sol#L239
Status	Unresolved

Description

The transactions are initially disabled for all users excluding the authorized addresses. The owner can enable the transactions for all users. Once the transactions are enable the owner will not be able to disable them again.

```
if (_limits) {
    if (from != owner() && to != owner() && to != address(0) && to !=
address(0xdead) && !swapping) {
        if (!tradingActive) {
            require(_isExcludedFromFees[from] || _isExcludedFromFees[to],
"Trading is not active.");
        }
```

Additionally, the contract owner has the authority to stop transactions, as detailed in the `PTRP` finding. As a result, the contract might operate as a honeypot.

Recommendation

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. Some suggestions are:

- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Additionally, the team is advised to follow the recommendations outlined in the `PTRP` finding and implement the necessary steps to mitigate the identified risks, ensuring that the contract does not operate as a honeypot. Renouncing ownership will effectively eliminate the threats, but it is non-reversible.

BC - Blacklists Addresses

Criticality	Critical
Location	NOCAP.sol#L204,246,546
Status	Unresolved

Description

The contract is designed to restrict large buy transactions during the initial trading phase using a combination of maximum wallet checks and address-based exclusions. However, the contract owner retains full authority to arbitrarily assign or remove any address as an Automated Market Maker (AMM) pair via the `setAutomatedMarketMakerPair` function, and to include or exclude addresses from maximum transaction limits using `excludeFromMaxTransaction`. By manipulating these settings, the owner could intentionally cause transfers to or from specific addresses (e.g., users or the liquidity pair) to revert during buys. This is possible by adding a legitimate address to the AMM pair list and removing its exclusion, triggering restrictive conditions that lead to reverts when the wallet balance or transaction amount exceeds the defined limits. Such control introduces a centralisation risk and can be abused to freeze or block trading for targeted users or contracts.

```
   function setAutomatedMarketMakerPair(address pair, bool value) public
onlyOwner {
        require(pair != uniswapV2Pair, "The pair cannot be removed from
automatedMarketMakerPairs");

        _setAutomatedMarketMakerPair(pair, value);
    }

  if (automatedMarketMakerPairs[from] &&
!_isExcludedMaxTransactionAmount[to]) {
            require(amount <= _maxWallet, "Buy transfer amount exceeds
the max amount.");
            require(amount + balanceOf(to) <= _maxWallet, "Max wallet
exceeded");
        }

   function excludeFromMaxTransaction(address updAds, bool isEx) public
onlyOwner {
        _isExcludedMaxTransactionAmount[updAds] = isEx;}
```

Recommendation

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

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

IBC - Ineffective Batch Claims

Criticality	Medium
Location	NOCAPPERS.sol#L158
Status	Unresolved

Description

The contract's `batchCollectFoundersEarnings` function allows users to pass multiple token IDs to claim rewards in a single call. However, due to the way `founderTokenBag` is globally tracked and not split per token, only the first token ID processed in the loop will yield the full claimable amount. Subsequent token IDs within the same transaction will observe no increase in `founderTokenBag` and thus have no additional rewards to claim. As a result, passing multiple token IDs is redundant and misleading, only the first effectively contributes to the payout, while the rest return zero rewards.

```
    function batchCollectFoundersEarnings(uint256[] calldata _tokenIds)
public nonReentrant {
        uint256 totalClaimEarnings = 0;

        for (uint256 i = 0; i < _tokenIds.length; i++) {
            uint256 tokenId = _tokenIds[i];
            require(this.ownerOf(tokenId) == msg.sender, "Not owner");
            require(founderTokenBag > founderWithdraws[tokenId], "Nothing
to claim");

            uint256 claimEarnings =
founderTokenBag.sub(founderWithdraws[tokenId]);
            founderWithdraws[tokenId] =
founderWithdraws[tokenId].add(claimEarnings);
            totalClaimEarnings = totalClaimEarnings.add(claimEarnings);
        }

        NoCapToken.transfer(msg.sender, totalClaimEarnings);
        emit ClaimedRewardsEv(msg.sender, totalClaimEarnings);
    }
```

Recommendation

It is recommended to revise the reward logic to distribute rewards proportionally across all NFTs based on their share. Alternatively, document that rewards are globally pooled and only claimable once per `founderTokenBag` update, to avoid confusion and unnecessary gas usage. If batch claiming remains supported, ensure that each token ID contributes meaningfully to the total claim, or restrict batch calls to enforce a one-claim-per-distribution rule.

MUI - Missing User Incentive

Criticality	Medium
Location	NOCAPPERS.sol#L129
Status	Unresolved

Description

The contract's `updateFoundersEarnings` function lacks any incentive for external users to call it. When called by non-token-contract addresses, users must transfer tokens to the contract, but only a fraction of their deposit, specifically, one unit per 5,000 tokens, is added to the global reward pool (`founderTokenBag`). The rest of the deposited amount remains locked in the contract with no direct benefit to the user. This one-sided mechanism discourages participation, as users lose tokens while they do not receive a proportionate contribution to the reward system.

```
   function updateFoundersEarnings(uint256 _depositAmount)
external {
        uint256 incrementBagIndex = _depositAmount.div(5000);

        if (msg.sender == NoCapTokenAddress) {
            founderTokenBag =
founderTokenBag.add(incrementBagIndex);
        } else {
            NoCapToken.safeTransferFrom(
                msg.sender,
                address(this),
                _depositAmount
            );
            founderTokenBag =
founderTokenBag.add(incrementBagIndex);
        }
    }
```

Recommendation

It is recommended to redesign the function to offer fairer incentives to users who deposit tokens. This could involve attributing a proportional share of deposited tokens to the user's NFT or introducing a mechanism to refund or reward users more directly based on their contribution. Clearly defining the purpose and benefit of calling `updateFoundersEarnings` would encourage proper use and improve system participation.

UFCD - Unfair First Claim Drain

Criticality	Medium
Location	NOCAPPERS.sol#L144,158
Status	Unresolved

Description

The contract is vulnerable to an unfair reward distribution mechanism in the `collectFoundersEarnings` and `batchCollectFoundersEarnings` functions. The current logic allows the first NFT ID that claims rewards to withdraw the entire `founderTokenBag` balance, regardless of the number of NFTs in circulation. Since rewards are not split proportionally across all NFTs, the first caller can drain the full amount available in the contract, leaving no rewards for other holders. This creates a race condition and undermines the fairness of the reward model.

```
    function collectFoundersEarnings(uint256 _tokenId) public nonReentrant
{
        require(this.ownerOf(_tokenId) == msg.sender, "Not owner");
        ...
        uint256 claimEarnings =
founderTokenBag.sub(founderWithdraws[_tokenId]);
        ...
        NoCapToken.transfer(msg.sender, claimEarnings);
        emit ClaimedRewardsEv(msg.sender, claimEarnings);
    }

    function batchCollectFoundersEarnings(uint256[] calldata _tokenIds)
public nonReentrant {
        ...
    }
```

Recommendation

It is recommended to modify the reward distribution logic so that each NFT can only claim a portion of the `founderTokenBag` balance, proportional to its share among all minted NFTs. This could be achieved by distributing rewards based on the total supply of NFTs at the time of distribution and tracking per-token entitlements accordingly. This change would ensure fair and predictable distribution for all eligible NFT holders.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	NOCAPPERS.sol#L56,186 NOCAP.sol#L472
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.

```
    function setTreasury(address _treasury) external payable onlyOwner {
        ...
    }

    function setStakingContracts(address _NOCAPStaking, address
_lpStaking, address _NoCappersNFTs) external onlyOwner {
        ...
    }

    function excludeFromFees(address account, bool excluded) public
onlyOwner {
        ...
    }

    function excludeFromMaxTransaction(address updAds, bool isEx) public
onlyOwner {
        _isExcludedMaxTransactionAmount[updAds] = isEx;
    }

    function updateTeamWallet(address newWallet) external onlyOwner {
        ...
    }

    function withdrawStuckEth(address toAddr) external onlyOwner {
        ...
    }
```

```
  function setNoCapToken(address _NoCapToken) external onlyOwner {
        NoCapToken = IERC20(_NoCapToken);
        NoCapTokenAddress = _NoCapToken;
    }

    function setTeamWallet(address _teamWalletAddress) external
onlyOwner {
        teamWallet = _teamWalletAddress;
    }

    function setTreasury(address _treasuryAddress) external onlyOwner
{
        treasury = _treasuryAddress;
    }

    function setBaseURI(string memory _newBaseURI) public onlyOwner {
        baseURI = _newBaseURI;
    }

    function setBaseExtension(
        string memory _newBaseExtension
    ) public onlyOwner {
        baseExtension = _newBaseExtension;
    }

    function pause(bool _state) public onlyOwner {
        paused = _state;
    }
```

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.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	NOCAP.sol#L283
Status	Unresolved

Description

Division of decimal (fixed point) numbers can result in rounding errors due to the way that division is implemented in Solidity. Thus, it may produce issues with precise calculations with decimal numbers.

Solidity represents decimal numbers as integers, with the decimal point implied by the number of decimal places specified in the type (e.g. decimal with 18 decimal places). When a division is performed with decimal numbers, the result is also represented as an integer, with the decimal point implied by the number of decimal places in the type. This can lead to rounding errors, as the result may not be able to be accurately represented as an integer with the specified number of decimal places.

Hence, the splitted shares will not have the exact precision and some funds may not be calculated as expected.

```
if (automatedMarketMakerPairs[to] && sellTotalFees > 0) {
      fees = (amount * sellTotalFees) / 10000;
      if (_limits) fees = fees * _taxMultiplier;
      tokensForLiquidity += (fees * sellLiquidityFee) /
sellTotalFees;
      tokensForTeam += (fees * sellTeamFee) / sellTotalFees;
      tokensForBacking += (fees * sellBackingFee) / sellTotalFees;
      tokensForRev += (fees * sellRevFee) / sellTotalFees;
      tokensForLPStaking += (fees * sellLPStakingFee) /
sellTotalFees;
      tokensForNFTStaking += (fees * sellNFTStakingFee) /
sellTotalFees;
  }
  // on buy
  else if (automatedMarketMakerPairs[from] && buyTotalFees > 0) {
      fees = (amount * buyTotalFees) / 10000;
      if (_limits) fees = fees * _taxMultiplier;
      tokensForLiquidity += (fees * buyLiquidityFee) /
buyTotalFees;
      tokensForTeam += (fees * buyTeamFee) / buyTotalFees;
      tokensForBacking += (fees * buyBackingFee) / buyTotalFees;
      tokensForRev += (fees * buyRevFee) / buyTotalFees;
      tokensForLPStaking += (fees * buyLPStakingFee) /
buyTotalFees;
      tokensForNFTStaking += (fees * buyNFTStakingFee) /
buyTotalFees;
  }
```

Recommendation

The team is advised to take into consideration the rounding results that are produced from the solidity calculations. The contract could calculate the subtraction of the divided funds in the last calculation in order to avoid the division rounding issue.

DRP - Dual Reward Payouts

Criticality	Minor / Informative
Location	Treasury.sol#L65
Status	Acknowledged

Description

The contract is susceptible to a double earning mechanism via the `depositForRedemption` function. This function allows users to burn their tokens in exchange for two separate assets, WETH and LP tokens. However, the design permits users to receive WETH directly from the contract and simultaneously obtain LP tokens, which can themselves be redeemed for additional WETH and the project's tokens. This effectively results in users extracting value twice from a single burn operation, once from the direct WETH payout and once from the underlying assets of the LP tokens. As a result, the protocol enables dual yield extraction, which could destabilize the tokenomics and disproportionately reward redemptions, especially when combined with the price mechanism that does not account for this dual outflow of value.

```
function depositForRedemption(uint256 _amount) external nonReentrant {
        ...
        uint256 supply = IERC20(NOCAP).totalSupply();
        uint256 wethBal = IERC20(WETH).balanceOf(address(this));
        uint256 lpBal = IERC20(uniswapV2Pair).balanceOf(address(this));

        uint256 wethOut = (_amount * wethBal * 9) / (supply * 10);
        uint256 lpOut = (_amount * lpBal * 9) / (supply * 10);

        require(wethOut > 0 || lpOut > 0, "Redemption too small");

        // record pending amounts and deadline
        p.wethAmount = wethOut;
        p.lpAmount = lpOut;
        p.deadline = block.timestamp + CLAIM_DELAY;

        // burn user's NOCAP
        INOCAP(NOCAP).burnFrom(msg.sender, _amount);

        emit RedemptionInitiated(msg.sender, wethOut, lpOut,
p.deadline);
    }
```

Recommendation

It is recommended to evaluate whether this dual reward mechanism aligns with the intended economic model of the protocol. If not explicitly designed to operate this way, the team should consider limiting rewards to a single asset type per redemption to avoid double-dipping and ensure sustainable token utility and liquidity.

MEM - Missing Error Messages

Criticality	Minor / Informative
Location	NOCAP.sol#L568
Status	Unresolved

Description

The contract is missing error messages. Specifically, there are no error messages to accurately reflect the problem, making it difficult to identify and fix the issue. As a result, the users will not be able to find the root cause of the error.

```
require(success)
```

Recommendation

The team is suggested to provide a descriptive message to the errors. This message can be used to provide additional context about the error that occurred or to explain why the contract execution was halted. This can be useful for debugging and for providing more information to users that interact with the contract.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	NOCAP.sol#L461 NOCAPPERS.sol#L186
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 enableTrading() external {
        ...
    }

    function setTreasury(address _treasury) external payable onlyOwner {
        ...
    }

    function setStakingContracts(address _NOCAPStaking, address
_lpStaking, address _NoCappersNFTs) external onlyOwner {
        ...
    }

    function updateSwapTokensAtPercent(uint256 newPercent) external
onlyOwner returns (bool) {
        ...
    }

    function updateSwapEnabled(bool enabled) external onlyOwner {
        swapEnabled = enabled;
    }
    function updateBuyFees(
        uint256 _backingFee,
        uint256 _liquidityFee,
        uint256 _teamFee,
        uint256 _revFee,
        uint256 _lpStakingFee,
        uint256 _nftStakingFee
    ) external onlyOwner {
        ...
    }

    function updateSellFees(
        uint256 _backingFee,
        uint256 _liquidityFee,
        uint256 _teamFee,
        uint256 _revFee,
        uint256 _lpStakingFee,
        uint256 _nftStakingFee
    ) external onlyOwner {
        ...
    }
```

```
function setBaseURI(string memory _newBaseURI) public onlyOwner {
        baseURI = _newBaseURI;
    }

    function setBaseExtension(
        string memory _newBaseExtension
    ) public onlyOwner {
        baseExtension = _newBaseExtension;
    }

    function pause(bool _state) public onlyOwner {
        paused = _state;
    }
```

Recommendation

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

MMC - Missing Minimum Check

Criticality	Minor / Informative
Location	NOCAP.sol#L129
Status	Unresolved

Description

The contract is missing a validation check in the `updateFoundersEarnings` function to ensure that the `_depositAmount` is greater than or equal to 5,000 units. Without this check, calls with a value less than 5,000 will result in an `incrementBagIndex` of zero due to integer division, causing no actual update to the `founderTokenBag` while still allowing the token transfer to proceed. This may confuse users or waste gas and tokens without yielding any effective outcome. In such cases, the `safeTransferFrom` call may emit a zero transfer event without any actual token transfer occurring, which could lead to misleading transaction logs and unnecessary gas consumption.

```
  function updateFoundersEarnings(uint256 _depositAmount)
external {
        uint256 incrementBagIndex = _depositAmount.div(5000);

        if (msg.sender == NoCapTokenAddress) {
            founderTokenBag =
founderTokenBag.add(incrementBagIndex);
        } else {
            NoCapToken.safeTransferFrom(
                msg.sender,
                address(this),
                _depositAmount
            );
            founderTokenBag =
founderTokenBag.add(incrementBagIndex);
        }
    }
```

Recommendation

It is recommended to add an explicit `require` statement ensuring that `_depositAmount` is at least 5,000 units. This will prevent ineffective transactions, reduce unnecessary token transfers, and ensure that each call to `updateFoundersEarnings` meaningfully contributes to the reward system.

MU - Modifiers Usage

Criticality	Minor / Informative
Location	NOCAPPERS.sol#L145,163
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(this.ownerOf(_tokenId) == msg.sender, "Not owner");
```

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.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	NOCAP.sol#L319
Status	Unresolved

Description

The contract operates under the assumption that liquidity is consistently provided to the pair between the contract's token and the native currency. However, there is a possibility that liquidity is provided to a different pair. This inadequacy in liquidity provision in the main pair could expose the contract to risks. Specifically, during eligible transactions, where the contract attempts to swap tokens with the main pair, a failure may occur if liquidity has been added to a pair other than the primary one. Consequently, transactions triggering the swap functionality will result in a revert.

```
  function swapTokensForEth(uint256 tokenAmount) internal {
        // generate the uniswap pair path of token -> weth
        address[] memory path = new address[](2);
        path[0] = address(this);
        path[1] = uniswapV2Router.WETH();

        if (tokenAmount > 0) {
            // make the swap

uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
                tokenAmount,
                0, // accept any amount of ETH
                path,
                address(this),
                block.timestamp
            );
        }
    }
```

Recommendation

The team is advised to implement a runtime mechanism to check if the pair has adequate liquidity provisions. This feature allows the contract to omit token swaps if the pair does not have adequate liquidity provisions, significantly minimizing the risk of potential failures.

Furthermore, the team could ensure the contract has the capability to switch its active pair in case liquidity is added to another pair.

Additionally, the contract could be designed to tolerate potential reverts from the swap functionality, especially when it is a part of the main transfer flow. This can be achieved by executing the contract's token swaps in a non-reversible manner, thereby ensuring a more resilient and predictable operation.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	NOCAP.sol#L346,400
Status	Unresolved

Description

The contract sends funds to a `teamWallet` , `NOCAPStaking` and `treasury` addresses as part of the transfer flow. These addresses can either be a wallet address or a contract. If the address belongs to a contract then it may revert from incoming payment. As a result, the error will propagate to the token's contract and revert the transfer.

```
(success,) = address(teamWallet).call{value: ethForTeam}("");
(success,) = address(NOCAPStaking).call{value: ethForRev}("");
(success,) = address(treasury).call{value: address(this).balance}("");
...
NoCappersNFTs.updateFoundersEarnings(tokensForNFTStaking);
...
    function addLiquidity(uint256 tokenAmount, uint256 ethAmount)
internal {
        // add the liquidity
        uniswapV2Router.addLiquidityETH{value: ethAmount}(
            address(this),
            tokenAmount,
            0, // slippage is unavoidable
            0, // slippage is unavoidable
            treasury,
            block.timestamp
        );
    }
```

Recommendation

The contract should tolerate the potential revert from the underlying contracts when the interaction is part of the main transfer flow. This could be achieved by not allowing set contract addresses or by sending the funds in a non-revertable way.

RASI - Redundant Amount Swap Inclusion

Criticality	Minor / Informative
Location	NOCAP.sol#L366
Status	Unresolved

Description

The contract is incorrectly including `tokensForNFTStaking` in the `totalTokensToSwap` calculation even though these tokens are transferred and reset to zero prior to the computation. This leads to a misleading total that suggests the `tokensForNFTStaking` tokens are part of the swap operation, when in fact they are not. Although the variable has been set to zero before the calculation, and thus the impact is currently neutral, this inclusion introduces unnecessary complexity and may cause confusion for future developers or auditors. Moreover, if the sequence of operations is modified in future updates, this discrepancy could lead to faulty logic or incorrect value distributions.

```
    function swapBack() internal {
        ...
        if (tokensForNFTStaking > 0) {
            super._transfer(address(this), NoCappersNFTsAddress,
tokensForNFTStaking);
            NoCappersNFTs.updateFoundersEarnings(tokensForNFTStaking);
            tokensForNFTStaking = 0;
        }

        uint256 contractBalance = balanceOf(address(this));
        uint256 totalTokensToSwap = tokensForLiquidity + tokensForBacking
+ tokensForTeam + tokensForRev + tokensForNFTStaking;
```

Recommendation

It is recommended to remove `tokensForNFTStaking` from the `totalTokensToSwap` calculation to improve code clarity, prevent potential misinterpretation of the swap logic, and reduce the risk of future functional bugs arising from changes in execution order.

RDO - Redundant Deployer Ownership

Criticality	Minor / Informative
Location	NOCAP.sol#L150
Status	Unresolved

Description

The contract is assigning the `deployer` variable to the `msg.sender` and also setting the `owner` and `teamWallet` to the same address during deployment. This creates a redundancy, as both `deployer` and `owner` reference the same entity but are used independently in the codebase. Maintaining multiple variables for the same authority increases the risk of inconsistent access control logic, unclear role separation, and unnecessary complexity when auditing or extending the contract.

```
    constructor() ERC20("No Cap", "NOCAP") {
        ...

        deployer = msg.sender;
        teamWallet = owner(); // set as team wallet
```

`

Recommendation

It is recommended to rely solely on the `Ownable` pattern's `owner()` for authority control and remove the redundant `deployer` variable unless it serves a distinct purpose with separate permissions. If differentiation between roles is intended, it should be clearly defined and enforced with separate logic and documentation.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	NOCAPPERS.sol
Status	Unresolved

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert to underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases gas consumption unnecessarily in cases where the explanatory error message is not used.

```
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library in cases where the revert error message is not used. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on https://docs.soliditylang.org/en/stable/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

STPB - Short Tier Price Bypass

Criticality	Minor / Informative
Location	NOCAPPERS.sol#L114
Status	Unresolved

Description

The contract is using a dynamic pricing mechanism in the `getNFTPrice` function that calculates NFT price based on the current `totalSupply` without accounting for the `mintAmount` requested in the transaction. As a result, users can mint multiple NFTs at a lower tier price even when their purchase causes the `totalSupply` to cross into a higher pricing tier. For example, if the current supply is 99 and a user mints 10 NFTs, all 10 will be priced using tier 0 rates, despite the supply crossing tier boundaries (e.g., tier 1 at 100). This allows users to bypass intended pricing progression for up to 10 NFTs, potentially reducing project revenue and creating unfair pricing discrepancies among minters.

```
    function getNFTPrice(uint256 _mintAmount) public view returns
(uint256) {
        uint256 currentSupply = totalSupply();
        uint256 tierIndex = 0;

        if (currentSupply >= 2500) {
            tierIndex = 3;
        } else if (currentSupply >= 500) {
            tierIndex = 2;
        } else if (currentSupply >= 100) {
            tierIndex = 1;
        }

        return _mintAmount * tiers[tierIndex];
    }
```

Recommendation

It is recommended to update the pricing logic to dynamically adjust the price tier for each NFT in the batch based on its position in the overall supply. Alternatively, restrict batch minting to avoid tier overlaps or recalculate the average cost of all NFTs in the mint batch using progressive pricing. This will ensure that each NFT is priced fairly according to the intended tier structure.

UBV - Unoptimized Batch Validation

Criticality	Minor / Informative
Location	NOCAPPERS.sol#L158
Status	Unresolved

Description

The contract's `batchCollectFoundersEarnings` function uses `require` statements to validate ownership and claim eligibility for each token ID in the loop. If any single token ID in the input array fails a check, either not owned by the caller or with no rewards to claim, the entire transaction is reverted, and no rewards are processed at all. This all-or-nothing approach can lead to user frustration, wasted gas, and poor UX, especially in cases where only one invalid or ineligible ID prevents valid claims from being processed.

```
    function batchCollectFoundersEarnings(uint256[] calldata _tokenIds)
public nonReentrant {
        uint256 totalClaimEarnings = 0;

        for (uint256 i = 0; i < _tokenIds.length; i++) {
            uint256 tokenId = _tokenIds[i];
            require(this.ownerOf(tokenId) == msg.sender, "Not owner");
            require(founderTokenBag > founderWithdraws[tokenId], "Nothing
to claim");

            ...
        }

        NoCapToken.transfer(msg.sender, totalClaimEarnings);
        emit ClaimedRewardsEv(msg.sender, totalClaimEarnings);
    }
```

Recommendation

It is recommended to replace the `require` statements with `if` conditions, allowing the function to gracefully skip invalid or ineligible token IDs and continue processing the rest. This ensures that users can still claim available rewards for NFTs they own and that satisfy the reward conditions, even if other NFTs in the same batch are not eligible.

ZAB - Zero Amount Burn

Criticality	Minor / Informative
Location	NOCAP.sol#L220,449
Status	Unresolved

Description

The contract is implementing a `burnFrom` function that allows authorized addresses to burn tokens on behalf of others. However, this function does not include a validation to ensure that the burn amount is greater than zero. As a result, users can invoke the `burnFrom` function with an amount of zero, which will pass and emit associated events unnecessarily. This behaviour introduces the risk of spamming the blockchain with redundant transactions and events, potentially leading to inflated logs, confusion in off-chain systems, and unnecessary gas consumption without meaningful state changes.

```
    function _burnFrom(address account, uint256 amount) internal {
        uint256 decreasedAllowance_ = allowance(account, msg.sender) -
amount;

        _approve(account, msg.sender, decreasedAllowance_);
        _burn(account, amount);
    }

    function burnFrom(address account, uint256 amount) external {
        _burnFrom(account, amount);
    }
```

Recommendation

It is recommended to include an explicit check to ensure the burn amount is greater than zero before proceeding with the burn logic. This will help prevent zero-value transactions, reduce unnecessary event emissions, and align the function with expected behaviour for token burning.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	NOCAPPERS.sol#L26
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 maxMintAmount = 10
```

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	Treasury.sol#L23,65,129,130 NOCAPPERS.sol#L40,41,56,61,65,73,114,129,144,158,176,186,191,196 NOCAP.sol#L17,55,64,68,70,133,144,472,480,503,504,505,506,507,508, 522,523,524,525,526,527,559
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 public immutable NOCAP
uint256 _amount
address _token
IERC20 public NoCapToken
address public NoCapTokenAddress
address _NoCapToken
address _teamWalletAddress
address _treasuryAddress
uint256 _mintAmount
uint256 _depositAmount
uint256 _tokenId
uint256[] calldata _tokenIds
address _owner
string memory _newBaseURI

...
```

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	NOCAP.sol#L284,285,286,287,288,289,290,291,295,296,297,298,299,300,301,302
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.

```
fees = (amount * sellTotalFees) / 10000
tokensForBacking += (fees * sellBackingFee) / sellTotalFees
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	NOCAP.sol#L279
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 fees
```

Recommendation

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

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	Treasury.sol#L55 NOCAPPERS.sol#L53,58,62,66 NOCAP.sol#L474,481,482,484,553,567
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.

```
NOCAP = _NOCAP
NoCapTokenAddress = _NoCapToken
teamWallet = _teamWalletAddress
treasury = _treasuryAddress
treasury = _treasury
NOCAPStaking = _NOCAPStaking
lpStaking = _lpStaking
NoCappersNFTsAddress = _NoCappersNFTs
teamWallet = newWallet
(bool success,) = toAddr.call{value: address(this).balance}("")
```

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	Treasury.sol#L2 NOCAPPERS.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 ^0.8.13;
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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	Treasury.sol#L134 NOCAPPERS.sol#L154,171 NOCAP.sol#L562
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
IERC20(WETH).transfer(msg.sender, _amountETHToSend)
IERC20(uniswapV2Pair).transfer(msg.sender, _amountLPToSend)
IERC20(_token).transfer(msg.sender, _amount)
NoCapToken.transfer(msg.sender, claimEarnings)
NoCapToken.transfer(msg.sender, totalClaimEarnings)
IERC20(_token).transfer(_to, _contractBalance)
```

Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

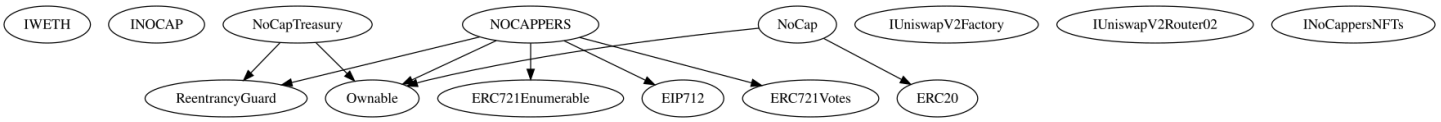
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
NoCapTreasury	Implementation	Ownable, ReentrancyGuard		
		Public	✓	-
		External	Payable	-
	depositForRedemption	External	✓	nonReentrant
	claimRedemption	External	✓	nonReentrant
	withdrawStuckToken	External	✓	onlyOwner
NOCAPPERS	Implementation	ERC721Enumerable, Ownable, EIP712, ERC721Votes, ReentrancyGuard		
		Public	✓	ERC721 EIP712
	setNoCapToken	External	✓	onlyOwner
	setTeamWallet	External	✓	onlyOwner
	setTreasury	External	✓	onlyOwner
	_baseURI	Internal		
	safeMint	Public	Payable	nonReentrant
	_refundExcess	Internal	✓	
	getNFTPrice	Public		-

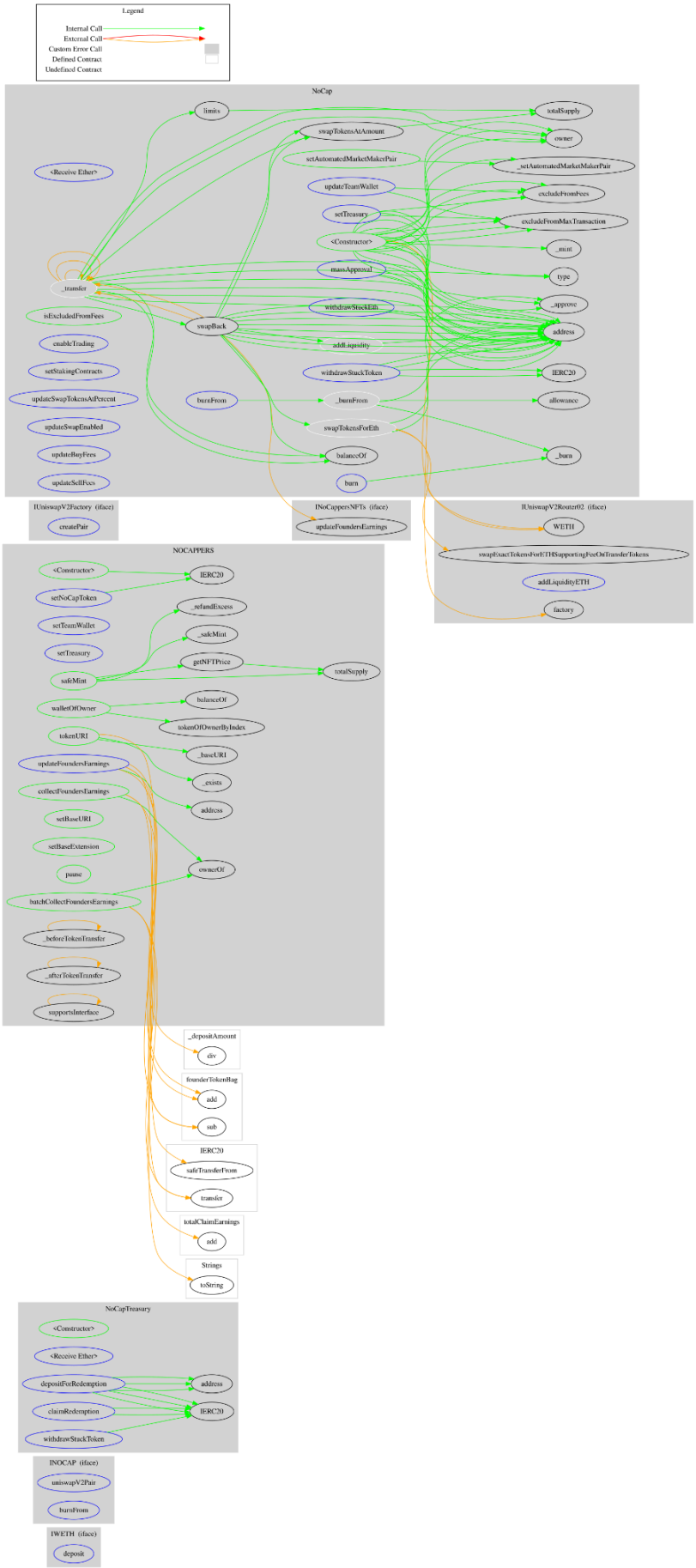
	updateFoundersEarnings	External	✓	-
	collectFoundersEarnings	Public	✓	nonReentrant
	batchCollectFoundersEarnings	Public	✓	nonReentrant
	walletOfOwner	Public		-
	setBaseURI	Public	✓	onlyOwner
	setBaseExtension	Public	✓	onlyOwner
	pause	Public	✓	onlyOwner
	tokenURI	Public		-
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
	supportsInterface	Public		-
NoCap	Implementation	ERC20, Ownable		
		Public	✓	ERC20
		External	Payable	-
	setAutomatedMarketMakerPair	Public	✓	onlyOwner
	_setAutomatedMarketMakerPair	Private	✓	
	_burnFrom	Internal	✓	
	_transfer	Internal	✓	
	swapTokensForEth	Internal	✓	
	addLiquidity	Internal	✓	
	swapBack	Internal	✓	
	isExcludedFromFees	Public		-

	swapTokensAtAmount	Public		-
	limits	Public		-
	massApproval	External	✓	-
	burnFrom	External	✓	-
	burn	External	✓	-
	enableTrading	External	✓	-
	setTreasury	External	Payable	onlyOwner
	setStakingContracts	External	✓	onlyOwner
	updateSwapTokensAtPercent	External	✓	onlyOwner
	updateSwapEnabled	External	✓	onlyOwner
	updateBuyFees	External	✓	onlyOwner
	updateSellFees	External	✓	onlyOwner
	excludeFromFees	Public	✓	onlyOwner
	excludeFromMaxTransaction	Public	✓	onlyOwner
	updateTeamWallet	External	✓	onlyOwner
	withdrawStuckToken	External	✓	onlyOwner
	withdrawStuckEth	External	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Custom Error Call
Defined Contract
Undefined Contract
NoCap
limits
totalSupply
swapTokensAtAmount
owner
setAutomatedMarketMakerPair
_setAutomatedMarketMakerPair
<Receive Ether>
updateTeamWallet
excludeFromFees
setTreasury
excludeFromMaxTransaction
<Constructor>
_mint
massApproval
type
_transfer
withdrawStuckEth
_approve
isExcludedFromFees
swapBack
address
enableTrading
addLiquidity
setStakingContracts
withdrawStuckToken
IERC20
updateSwapTokensAtPercent
burnFrom
_burnFrom
allowance
updateSwapEnabled
swapTokensForEth
updateBuyFees
balanceOf
_burn
updateSellFees
burn
IUniswapV2Factory (iface)
createPair
INoCappersNFTs (iface)
updateFoundersEarnings
IUniswapV2Router02 (iface)
WETH
swapExactTokensForETHSupportingFeeOnTransferTokens
addLiquidityETH
factory
NOCAPPERS
<Constructor>
IERC20
setNoCapToken
_refundExcess
setTeamWallet
_safeMint
setTreasury
getNFTPrice
safeMint
totalSupply
balanceOf
walletOfOwner
tokenOfOwnerByIndex
tokenURI
_baseURI
updateFoundersEarnings
_exists
collectFoundersEarnings
address
setBaseURI
setBaseExtension
ownerOf
pause
batchCollectFoundersEarnings
_beforeTokenTransfer
_afterTokenTransfer
supportsInterface
_depositAmount
div
founderTokenBag
add
sub
IERC20
safeTransferFrom
transfer
totalClaimEarnings
add
Strings
toString
NoCapTreasury
<Constructor>
<Receive Ether>
depositForRedemption
address
claimRedemption
IERC20
withdrawStuckToken
INOCAP (iface)
uniswapV2Pair
burnFrom
IWETH (iface)
deposit

Summary

The NOCAP protocol implements a token, NFT, and treasury-backed redemption mechanism. This audit investigates security issues, business logic concerns, and potential improvements. The ERC20 token contract applies an initial trading fee of 30% during the first 30 seconds, decreasing to 18% and 12% in the next phases, and stabilizing at 6%. The NFT contract allows ETH-based minting with dynamic pricing tiers and enables holders to claim token rewards, while the treasury contract provides redemption rights after 180 days.

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