Cyberscope

# Audit Report

# Autonomi

March 2025

| | |
|---|---|
| Repository | https://github.com/lajosdeme/autonomi-claims |
| Files | FoundationEmissions.sol |
| Commit | 1d6e4ff78c307097df38ade0930364246ce6430f |
| 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

| | |
|---|---|
| **Repository** | https://github.com/lajosdeme/autonomi-claims |
| **Commit** | 1d6e4ff78c307097df38ade0930364246ce6430f |

## Audit Updates

| | |
|---|---|
| **Initial Audit** | 04 Mar 2025 |

## Source Files

| Filename | SHA256 |
|---|---|
| **FoundationEmissions.sol** | 77dba625923cdc22cc376ef2bbf4d4d95a9b9025a9594fb69a792f1bfc7477fd |

# Overview

## FoundationEmissions Contract

The `FoundationEmissions` contract manages the release of a specific allocation of ANT tokens for the foundation according to a structured emissions schedule. The contract defines a 50-year release period with token distribution percentages that vary each year, starting from 20% in the first year and gradually decreasing over time. The total token allocation for the foundation is 292,699,223 tokens, and the release of tokens is based on a fixed yearly vesting schedule.

### Claiming Mechanism

The `claim` function allows the designated foundation address to claim their allocated tokens based on the emissions schedule. Only the foundation's address can call this function, ensuring that no unauthorized parties can claim the tokens. The function calculates the claimable amount based on the percentage for the current year and transfers the tokens to the foundation. It also updates the total amount already claimed to prevent over-claiming in subsequent years. Once tokens are claimed, an event ( `Claimed` ) is emitted to log the amount and the timestamp of the claim.

### Token Emissions Calculation

The `_claimable` function calculates the claimable amount of tokens for the foundation based on the elapsed time since the vesting start timestamp. The function calculates the number of full years that have passed since the start and uses this to determine the percentage of the total allocation that has been unlocked. The percentages are predefined for each year in the release schedule and gradually decrease over time.

### Security Measures

The contract ensures that only the designated foundation address can claim the tokens by using a simple authorization check in the claim function. If an unauthorized address attempts to call the function, it will revert the transaction with an `Unauthorized` error. The `SafeERC20` library from OpenZeppelin is used to ensure safe token transfers, preventing any issues with transferring the allocated tokens.

# Roles

## Foundation

The foundation, designated as the authorized entity for claiming tokens, can interact with the following functions:

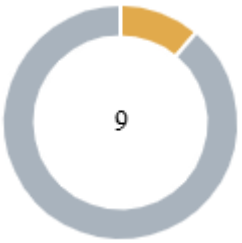
- `function claim()`

## Retrieval Functions

The following functions can be used to retrieve information:

- `function getClaimable()`

# Findings Breakdown




| | |
|---|---|
| Critical | 0 |
| Medium | 1 |
| Minor / Informative | 8 |

| Severity | Unresolved | Acknowledged | Resolved | Other |
|---|---|---|---|---|
| Critical | 0 | 0 | 0 | 0 |
| Medium | 1 | 0 | 0 | 0 |
| Minor / Informative | 8 | 0 | 0 | 0 |

# Diagnostics

Critical Medium Minor / Informative

| Severity | Code | Description | Status |
|---|---|---|---|
| Medium | EEP | Excluded Elapsed Period | Unresolved |
| Minor | PAI | Potential Allocation Inconsistency | Unresolved |
| Minor | ACI | Accidental Console Import | Unresolved |
| Minor | AAC | Allocation Amount Clarification | Unresolved |
| Minor | UTPD | Unverified Third Party Dependencies | Unresolved |
| Minor | CCR | Contract Centralization Risk | Unresolved |
| Minor | CCO | Claimable Calculations Optimization | Unresolved |
| Minor | L04 | Conformance to Solidity Naming Conventions | Unresolved |
| Minor | L16 | Validate Variable Setters | Unresolved |

# EEP - Excluded Elapsed Period

| | |
|---|---|
| **Criticality** | Medium |
| **Location** | FoundationEmissions.sol#L69 |
| **Status** | Unresolved |

## Description

The `_claimable` function does not account for 0 elapsed years. If zero years have passed and this function is called, 80% of the `TOTAL_FOUNDATION_ALLOCATION` will be claimed by the `FOUNDATION` .

```
function _claimable() internal view returns (uint256 claimable)
{
        uint256 elapsedTime = block.timestamp -
VESTING_START_TIMESTAMP;
        uint256 elapsedYears = elapsedTime /
BASE_VESTING_PERIOD;

        uint256 percentage;

        if (elapsedYears == 1) {
            percentage = 800;
        } else if (elapsedYears == 2) {
            //...
        }
        //...
        } else {
            percentage = 8000; // Capped at 80%
        }

        claimable = ((TOTAL_FOUNDATION_ALLOCATION * percentage)
/ 10000) - totalAlreadyClaimed;
    }
```

## Recommendation

The team is advised to consider the case of zero elapsed years and adjust the logic of the contract accordingly.

# PAI - Potential Allocation Inconsistency

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | FoundationEmissions.sol#L35 |
| **Status** | Unresolved |

## Description

The `TOTAL_FOUNDATION_ALLOCATION` is a constant number representing the amount of tokens allocated. However, this number does not account for the actual balance of the contract. This creates an inconsistency between the balance of the contract and the `TOTAL_FOUNDATION_ALLOCATION`.

The balance of the contract should be at least 80% of the `TOTAL_FOUNDATION_ALLOCATION` in order for `FOUNDATION` to receive the amount allocated for them.

```
uint256 public constant TOTAL_FOUNDATION_ALLOCATION =
292_699_223 ether;
```

## Recommendation

The team could consider adding the necessary logic to the contract to ensure that the balance of the contract matches the 80% of the `TOTAL_FOUNDATION_ALLOCATION`.

## ACI - Accidental Console Import

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | FoundationEmissions.sol#L4 |
| **Status** | Unresolved |

### Description

The contract imports the console from the `forge-std` lib. This is typically used for debugging purposes during development and testing but should be removed in production to avoid unnecessary dependencies and potential security risks.

```
import "forge-std/src/console.sol";
```

### Recommendation

It is recommended to remove imports used for testing when trying to deploy the contract on an actual network.

## AAC - Allocation Amount Clarification

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | FoundationEmissions.sol#L35 |
| **Status** | Unresolved |

### Description

The `TOTAL_FOUNDATION_ALLOCATION` is equal to `292_699_223 ether`. Additionally, the `TOTAL_FOUNDATION_ALLOCATION` serves the purpose of holding the amount of ant tokens allocated for the `FOUNDATION`. The `ether` suggests that the decimals of the token are 18. However, if the tokens have less than 18 decimals this value will be incorrect.

```
uint256 public constant TOTAL_FOUNDATION_ALLOCATION =
292_699_223 ether;
```

### Recommendation

It is recommended that the `TOTAL_FOUNDATION_ALLOCATION` is calculated in the constructor while also accounting for the decimals of the ant token. This could be achieved by directly calling the token's `decimals` function.

## UTPD - Unverified Third Party Dependencies

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | FoundationEmissions.sol#L60 |
| **Status** | Unresolved |

### Description

The contract uses an external contract in order to determine the transaction's flow. The external contract is untrusted. As a result, it may produce security issues and harm the transactions.

```
ANT_TOKEN.safeTransfer(FOUNDATION, claimable);
```

### Recommendation

The contract should use a trusted external source. A trusted source could be either a commonly recognized or an audited contract. The pointing addresses should not be able to change after the initialization.

## CCR - Contract Centralization Risk

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | FoundationEmissions.sol#L52,60 |
| **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 claim() external {
    if (msg.sender != FOUNDATION) {
        revert Unauthorized();
    }
    //...
}
```

Additionally, the contract logic works under the assumption that tokens will be held in the contract. However this essentially means that an external party will be responsible for providing the tokens to the contract.

```
ANT_TOKEN.safeTransfer(FOUNDATION, claimable);
```

### 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.

## CCO - Claimable Calculations Optimization

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | FoundationEmissions.sol#L75 |
| **Status** | Unresolved |

## Description

The `_claimable` function has a big `if` statement to calculate the percentage for the claimable tokens. As years pass, this if statement will be more gas expensive.

```
if (elapsedYears == 1) {
    percentage = 800;
} else if(elapsedYears == 2){
    //..
} //..
```

## Recommendation

The team could consider using a more efficient way to find the percentage. An approach would be using a local array of percentages or a state like a mapping or an array to sort them.

## L04 - Conformance to Solidity Naming Conventions

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | FoundationEmissions.sol#L36,38 |
| **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.

```
IERC20 public immutable ANT_TOKEN
address public immutable FOUNDATION
```

### 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.

## L16 - Validate Variable Setters

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | FoundationEmissions.sol#L48 |
| **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.

```
FOUNDATION = foundation
```

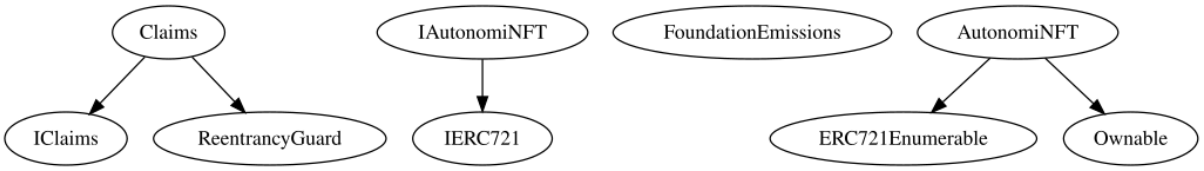
### Recommendation

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.

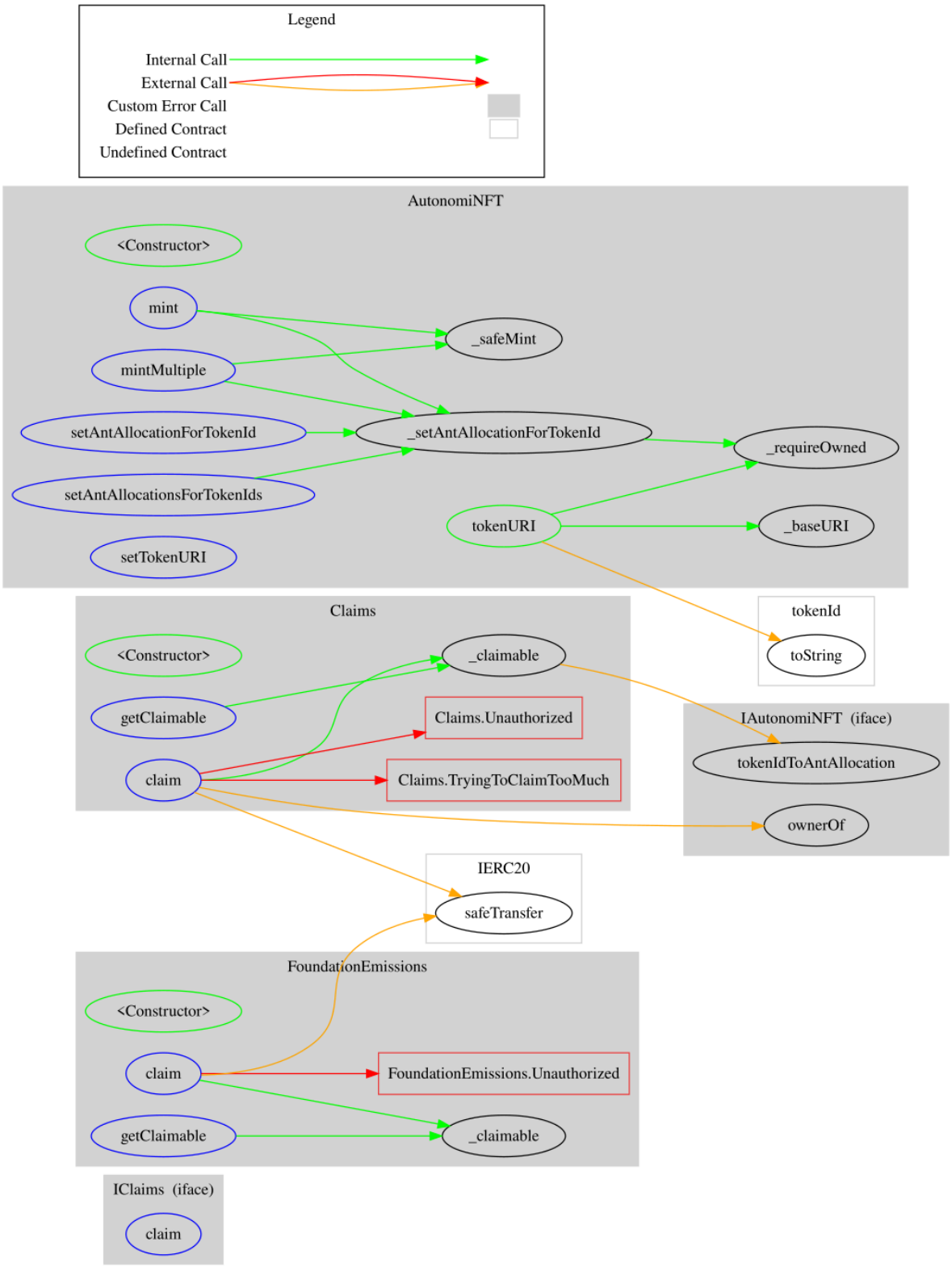
# Functions Analysis

| Contract | Type | Bases | | |
|---|---|---|---|---|
| | **Function Name** | **Visibility** | **Mutability** | **Modifiers** |
| | | | | |
| **FoundationEmissions** | Implementation | | | |
| | | Public | ✓ | - |
| | claim | External | ✓ | - |
| | getClaimable | External | | - |
| | _claimable | Internal | | |

# Inheritance Graph



# Flow Graph


Legend
Internal Call
External Call
Custom Error Call
Defined Contract
Undefined Contract
AutonomiNFT
<Constructor>
mint
mintMultiple
setAntAllocationForTokenId
setAntAllocationsForTokenIds
setTokenURI
_safeMint
_setAntAllocationForTokenId
tokenURI
_requireOwned
_baseURI
Claims
<Constructor>
getClaimable
claim
_claimable
Claims.Unauthorized
Claims.TryingToClaimTooMuch
tokenId
toString
IAutonomiNFT (iface)
tokenIdToAntAllocation
ownerOf
IERC20
safeTransfer
FoundationEmissions
<Constructor>
claim
getClaimable
FoundationEmissions.Unauthorized
_claimable
IClaims (iface)
claim

# Summary

Autonomi FoundationEmissions contract implements a vesting mechanism. This audit investigates security issues, business logic concerns and potential improvements. The Smart Contract analysis reported a compiler error and a critical issue.

# 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