Cyberscope

Audit Report

Openmesh

August 2025

Repository	https://github.com/OpenxAI-Network/openxai-token
Commit	60cf4ab5b37aafdc976496edb6c65f85c4302d6a
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/OpenxAI-Network/openxai-token
Commit	60cf4ab5b37aafdc976496edb6c65f85c4302d6a

Audit Updates

Initial Audit	19 Feb 2025 https://github.com/cyberscope-io/audits/blob/main/4-open/v1/audit.pdf
Corrected Phase 2	26 Feb 2025 https://github.com/cyberscope-io/audits/blob/main/4-open/v2/audit.pdf
Corrected Phase 3	06 Aug 2025

Source Files

Filename	SHA256
Rescue.sol	6177bf9537b8df5465a4570576431de42be1d8e0b08449e98cd085f85d574835
OpenxAIGenesis.sol	a2447d41034f9e38e85c99f7c9038b1f7ec57070c9b7b891d4dd1f587589c13d
OpenxAIClaimer.sol	f59d3006fe21fcf0f1aadc06e66372c06d40eb2a60c036597de0ecd37aff81c7
OpenxAI.sol	f3eaf9d1191b54fde242284c3d8a11c8e59db036b96c791b3e056a8b294b38c8
IMintable.sol	197cdc994f81acafd59306a1b8f44b640396acb7fc771de26fd67e18b845746e

Overview

The OpenxAI ecosystem is designed to facilitate a decentralised token economy, integrating token minting, fundraising, claim-based distributions, and a secure fund recovery mechanism. The contracts enable users to contribute native, wrapped or ERC-20 tokens toward predefined tiers, claim rewards based on off-chain proofs, and securely manage token supply. The system ensures transparency and controlled token distribution through smart contract-based logic, enhancing trust and efficiency in the token's lifecycle.

OpenxAI contract

The OpenxAI contract serves as the primary token contract, implementing an upgradeable ERC-20Votes with minting and burning capabilities. It enforces access control, ensuring only authorised roles can mint new tokens. The contract also supports future upgrades, allowing adaptability while maintaining a secure tokenomics structure.

OpenxAIGenesis contract

The OpenxAIGenesis contract functions as the fundraising mechanism, allowing contributions in ETH, wrapped ETH, and stablecoins. It categorises contributions into predefined tiers, allocating funds to a specific receiver address based on contribution size. It integrates Chainlink price feeds to ensure accurate conversions of ETH to stablecoin value, preventing price manipulation.

OpenxAIClaiming contract

The OpenxAIClaiming contract enables users to claim tokens by providing a valid cryptographic proof. It ensures that claims are only processed if they meet pre-defined spending limits and are verified against an off-chain signer. The contract maintains a tracking system to prevent double claims and enforces spending caps per period to maintain controlled token distribution.

Rescue contract

The Rescue contract provides a recovery mechanism, allowing a predefined rescue address to withdraw stuck ERC-20 tokens or native ETH. This ensures that misplaced funds can be recovered securely while restricting unauthorised access, maintaining the overall security and integrity of the ecosystem.

Roles

Admin

The `ADMIN` can interact with the following functions:

- `grantRole(bytes32 role, address account)`
- `revokeRole(bytes32 role, address account)`
- `hasRole(bytes32 role, address account)`

Minter

The `MINT_ROLE` can interact with the following function:

- `mint(address account, uint256 amount)`

Upgrade Role

The `UPGRADE_ROLE` can interact with the following function:

- `_authorizeUpgrade(address newImplementation)`

Rescue Address

The `RESCUE` address can interact with the following function:

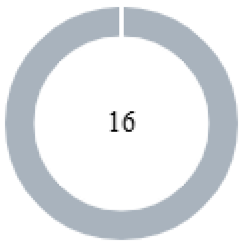
- `rescue(IERC20 _token, address payable _receiver, uint256 _amount)`

Users

Users can interact with the following functions:

- `burn(uint256 amount)`
- `transfer_native()`
- `transfer_erc20(IERC20 _token, uint256 _amount)`
- `claim(uint8 _v, bytes32 _r, bytes32 _s, address _claimer, uint256 _total)`

Findings Breakdown



●	Critical	0
●	Medium	0
●	Minor / Informative	16

Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	0	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	2	14	0	0

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	CR	Code Repetition	Acknowledged
●	CCR	Contract Centralization Risk	Acknowledged
●	ITI	Inefficient Tier Iteration	Acknowledged
●	MT	Mints Tokens	Acknowledged
●	METH	Missing Empty Tiers Handling	Acknowledged
●	MPV	Missing Parameter Validations	Acknowledged
●	MZVC	Missing Zero Value Check	Acknowledged
●	ODM	Oracle Decimal Mismatch	Acknowledged
●	POSD	Potential Oracle Stale Data	Acknowledged
●	PSU	Potential Subtraction Underflow	Unresolved
●	PTAI	Potential Transfer Amount Inconsistency	Acknowledged
●	PTRP	Potential Transfer Revert Propagation	Acknowledged
●	L04	Conformance to Solidity Naming Conventions	Acknowledged
●	L14	Uninitialized Variables in Local Scope	Acknowledged

●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Acknowledged

CR - Code Repetition

Criticality	Minor / Informative
Location	OpenxAIGenesis.sol#L76,109,139
Status	Acknowledged

Description

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

```
function _native_contribution() internal {...}
function _wrappedeth_contribution(IERC20 _token, uint256
_amount) internal {...}
function _stablecoin_contribution(IERC20 _token, uint256
_amount) internal {...}
```

Recommendation

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

Team Update

The team has acknowledged that this is not a security issue and states:

Code repetition of tier/loop traversal can be prevented, however as the difference lies in the inside of the loop, not much reduction is achieved. Repetition is chosen due to its easier readability and faster developer time. As the contract will only be used once for a limited time, maintainability is not a concern.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	OpenxAIClaimer.sol#L73 Rescue.sol#L11 OpenxAI.sol#L39
Status	Acknowledged

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.

```
address signer = ECDSA.recover(
  _hashTypedDataV4(keccak256(abi.encode(CLAIM_TYPEHASH,
_proofId, _claimer, _amount))), _v, _r, _s);

if (signer != owner()) {
  revert InvalidProof();
}
```

```
function rescue(IERC20 _token, address payable _receiver,
uint256 _amount) external {
  if (msg.sender != RESCUE) {
    revert NotRescue(msg.sender, RESCUE);
  }
  ...
}
```

```
function mint(address account, uint256 amount) external
onlyRole(MINT_ROLE)
```

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.

Team Update

The team has acknowledged that this is not a security issue and states:

The centralised signer is preferred due to its flexibility and convenience in this stage of the project. Rescue is added in case of user mistakes and under normal circumstances will/can never be used.

ITI - Inefficient Tier Iteration

Criticality	Minor / Informative
Location	OpenxAIGenesis.sol#L81,114,143
Status	Acknowledged

Description

The contract is iterating over the entire `tiers` array, including with zero value. This results in unnecessary computational overhead, as the contract processes every element from the beginning of the array, even when certain tiers contain no relevant data. This inefficiency increases gas consumption, particularly in scenarios where the array is large and contains multiple zero-value tiers.

```
for (uint256 i; i < tiers.length; i++) {
  uint256 usdInTier = tiers[i];
  if (usdInTier == 0) {
    continue;
  }
  ...
}
```

Recommendation

It is recommended to optimise the contract by maintaining an index that tracks the last non-zero tier and iterating only from that point forward. This approach reduces unnecessary iterations, improving gas efficiency and enhancing contract performance.

Team Update

The team has acknowledged that this is not a security issue and states:

Tier iteration is done this way to prevent (costly) storage writes. Although increasing total gas consumption, it will be distributed amongst participants, instead of only the tier transition transactions paying significantly more (which can lead to unexpected issues, e.g. not enough gas provided in case your transaction unexpectedly traverses tiers).

MT - Mints Tokens

Criticality	Minor / Informative
Location	OpenxAI.sol#L39 OpenxAIClaimer.sol#L48
Status	Acknowledged

Description

The `MINT_ROLE` and the owner of the `OpenxAIClaimer` have the authority to mint tokens. These entities may take advantage of it by calling the `mint` and `claim` function. As a result, the contract tokens will be highly inflated.

```
 function mint(address account, uint256 amount) external
onlyRole(MINT_ROLE) {
     _mint(account, amount);
   }
```

Recommendation

The team should carefully manage the private keys of the `MINT_ROLE` and 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.

Team Update

The team has acknowledged that this is not a security issue and states:

MINT_ROLE will only be granted to smart contracts. ADMIN_ROLE to grant MINT_ROLE to addresses is locked behind governance.

METH - Missing Empty Tiers Handling

Criticality	Minor / Informative
Location	OpenxAIGenesis.sol#L81,114,143
Status	Acknowledged

Description

The contract does not revert when all tiers have an amount of zero in `_native_contribution` , `_wrappedeth_contribution` , and `_stablecoin_contribution` , leading to silent failures. If `ethRemaining` (for ETH contributions) or `usdRemaining` (for ERC-20 contributions) equals `msg.value` or the provided token amount at the end of execution, no valid contribution has occurred, yet the user still incurs gas costs. Additionally, looping through empty tiers before terminating consumes unnecessary gas, and the lack of an explicit error message can mislead users into thinking their contribution was processed.

```
for (uint256 i; i < tiers.length; i++) {
  uint256 usdInTier = tiers[i];
  if (usdInTier == 0) {
    continue;
  }
  ...
}
```

Recommendation

It is recommended to add a validation check at the start of these functions to revert if all tiers are empty. Additionally, the contract should explicitly revert if `ethRemaining == msg.value` or `usdRemaining` equals the ERC-20 contribution amount at the end of execution, ensuring users do not pay gas fees for failed transactions.

Team Update

The team has acknowledged that this is not a security issue and states:

An explicit error could be thrown to indicate that no funds were taking and reducing gas for these transactions. It is chosen to prevent this confusion in the frontend instead.

MPV - Missing Parameter Validations

Criticality	Minor / Informative
Location	OpenxAIClaimer.sol#L31
Status	Acknowledged

Description

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

The contract is missing essential validation checks for constructor parameters, specifically `_spendingPeriodDuration` , `_token` , and `_signer` . These parameters are assigned directly without ensuring they hold valid, nonzero values. If `_spendingPeriodDuration` is set to zero, it will lead to incorrect calculations in time-dependent logic. Similarly, if `_token` is assigned an invalid (zero) address, interactions with the token may fail, potentially breaking core functionality. The lack of validation for `_signer` also introduce security risks, as an improperly set signer might prevent signature-based authentication from functioning as intended.

```
constructor(
    IMintable _token,
    uint256 _tokenSpendingLimit,
    uint256 _spendingPeriodDuration,
    address _signer
  ) Ownable(_signer) EIP712("OpenxAIClaiming", "1") {
    token = _token;
    tokenSpendingLimit = _tokenSpendingLimit;
    spendingPeriodDuration = _spendingPeriodDuration;
  }
```

Recommendation

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

It is recommended to enforce explicit validation checks within the constructor to ensure that `_spendingPeriodDuration` , `_token` , and `_signer` are not zero. This can be achieved by implementing `require` statements that revert the transaction if any of these values are invalid. Proper validation will enhance contract reliability, prevent misconfigurations, and reduce the risk of unintended behaviours or security vulnerabilities.

Team Update

The team has acknowledged that this is not a security issue and states:

Parameters are checked before signing the transaction. Unexpected zero parameters are easy to spot and unnecessary to add to the contract explicitly.

MZVC - Missing Zero Value Check

Criticality	Minor / Informative
Location	OpenxAIClaimer.sol#L76,109,139
Status	Acknowledged

Description

The contract is missing validation checks in multiple functions that handle native and ERC-20 token contributions (`_native_contribution` , `_wrappedeth_contribution` , and `_stablecoin_contribution`). These functions allow funds to be transferred to an escrow address but do not prevent execution when the contributed amount is zero. As a result, a caller can invoke these functions with zero value, triggering event emissions such as `Participated` without making an actual contribution. This could be exploited to spam the blockchain with fake participation events, misleading external systems that rely on these events for tracking contributions. Additionally, since there are no other variables explicitly tracking genuine participation, this issue can lead to inaccurate reporting of contributions and potential abuse of event-based logic in off-chain systems.

```
function _native_contribution() internal {
  uint256 usdTotal = (msg.value * _eth_price()) / (10 ** 18);
  uint256 usdRemaining = usdTotal;
  uint256 ethRemaining = msg.value;
  ...
}

function _wrappedeth_contribution(IERC20 _token, uint256
_amount) internal {
  uint256 usdTotal = (_amount * _eth_price()) / (10 ** 18);
  uint256 usdRemaining = usdTotal;
  uint256 ethRemaining = _amount;
  ....
}

function _stablecoin_contribution(IERC20 _token, uint256
_amount) internal {
  uint256 usdTotal = _amount;
  uint256 usdRemaining = usdTotal;
  ...
}
```

Recommendation

It is recommended to include explicit `require` checks in all relevant functions to prevent zero-value contributions. These checks should ensure that `msg.value` (for native contributions) and `_amount` (for ERC-20 contributions) are greater than zero before proceeding. This will prevent event spamming, enhance contract security, and ensure that only valid contributions trigger participation events.

Team Update

The team has acknowledged that this is not a security issue and states:

Zero value participation events do not cause any issues in off-chain components.

ODM - Oracle Decimal Mismatch

Criticality	Minor / Informative
Location	OpenxAIGenesis.sol#L66
Status	Acknowledged

Description

The contract relies on data retrieved from an external Oracle to make critical calculations. However, the contract does not include a verification step to align the decimal precision of the retrieved data with the precision expected by the contract's internal calculations. This mismatch in decimal precision can introduce substantial errors in calculations involving decimal values.

```
function _eth_price() internal view returns (uint256 price) {
  (, int256 ethPrice, , , ) = ethOracle.latestRoundData();
  if (ethPrice < 0) {
    revert InvalidPrice(ethPrice);
  }
  // Assume decimals of price oracle is 8, while stable coin is
6
  price = uint256(ethPrice / 100);
}
```

Recommendation

The team is advised to retrieve the decimals precision from the Oracle API in order to proceed with the appropriate adjustments to the internal decimals representation.

Team Update

The team has acknowledged that this is not a security issue and states:

Oracle decimal retrieval complicates logic and increases gas usage. As the contract has a limited lifespan (less than a month), the decimals of the Oracle is unlikely to change.

POSD - Potential Oracle Stale Data

Criticality	Minor / Informative
Location	OpenxAIGenesis.sol#L66
Status	Acknowledged

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.

```
function _eth_price() internal view returns (uint256 price) {
  (, int256 ethPrice, , , ) = ethOracle.latestRoundData();
  if (ethPrice < 0) {
    revert InvalidPrice(ethPrice);
  }
  price = uint256(ethPrice / 100);
}
```

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.

Team Update

The team has acknowledged that this is not a security issue and states:

Not accepting participation is a bigger concern than outdated Oracle data. The Oracle is an improvement of a hardcoded value and not of great importance if outdated.

PSU - Potential Subtraction Underflow

Criticality	Minor / Informative
Location	OpenxAIClaimer.sol#L55
Status	Unresolved

Description

The contract subtracts two values, the second value may be greater than the first value if the contract owner misuses the configuration. As a result, the subtraction may underflow and cause the execution to revert.

```
uint256 amount = _total - claimed[_claimer];
```

Recommendation

The team is advised to properly handle the code to avoid underflow subtractions and ensure the reliability and safety of the contract. The contract should ensure that the first value is always greater than the second value. It should add a sanity check in the setters of the variable or not allow executing the corresponding section if the condition is violated.

PTAI - Potential Transfer Amount Inconsistency

Criticality	Minor / Informative
Location	OpenxAIGenesis.sol#L123,130,152,158
Status	Acknowledged

Description

The `safeTransferFrom` function is 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

```
SafeERC20.safeTransferFrom(_token, msg.sender, receiver,
ethRemaining);
...
SafeERC20.safeTransferFrom(_token, msg.sender, receiver,
ethUsed);
...
SafeERC20.safeTransferFrom(_token, msg.sender, receiver,
usdRemaining);
...
SafeERC20.safeTransferFrom(_token, msg.sender, receiver,
usdInTier);
```

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`

Team Update

The team has acknowledged that this is not a security issue and states:

The ERC20 contracts that will be used do not have such a tax. Adding this to the contract adds unnecessary complexity

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	OpenxAIGenesis.sol#L90
Status	Acknowledged

Description

The contract sends funds to a `receiver` as part of the transfer flow. This address 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.

```
Address.sendValue(receiver, ethRemaining);
```

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.

Team Update

The team has acknowledged that this is not a security issue and states:

Funds should not be accepted if they cannot be securely stored in the designated escrow.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	Rescue.sol#L12,13,14 OpenxAIGenesis.sol#L52,56,66,76,109,139 OpenxAIClaimer.sol#L49,50,51,52,53
Status	Acknowledged

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 _token
address payable _receiver
uint256 _amount

function transfer_native() external payable {
        _native_contribution();
    }

...
```

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.

Team Update

The team has acknowledged that this is not a security issue and states:

Programmer preference: naming this way gives more clarity.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	OpenxAIGenesis.sol#L43,46,81,114,143
Status	Acknowledged

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 i
```

Recommendation

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

Team Update

The team has acknowledged that this is not a security issue and states:

Programmer preference: do not initialize variables to their default value.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	OpenxAIGenesis.sol#L41
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.

```
receiver = _receiver
```

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	Rescue.sol#L2 OpenxAIGenesis.sol#L2 OpenxAIClaimer.sol#L2 OpenxAI.sol#L2
Status	Acknowledged

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.20;
pragma solidity ^0.8.0;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

Team Update

The team has acknowledged that this is not a security issue and states:

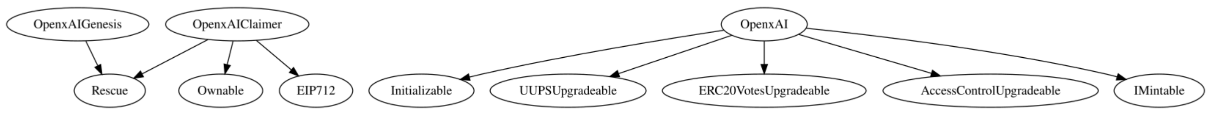
Any accepted Solidity version does not cause issues with these contracts.

Functions Analysis

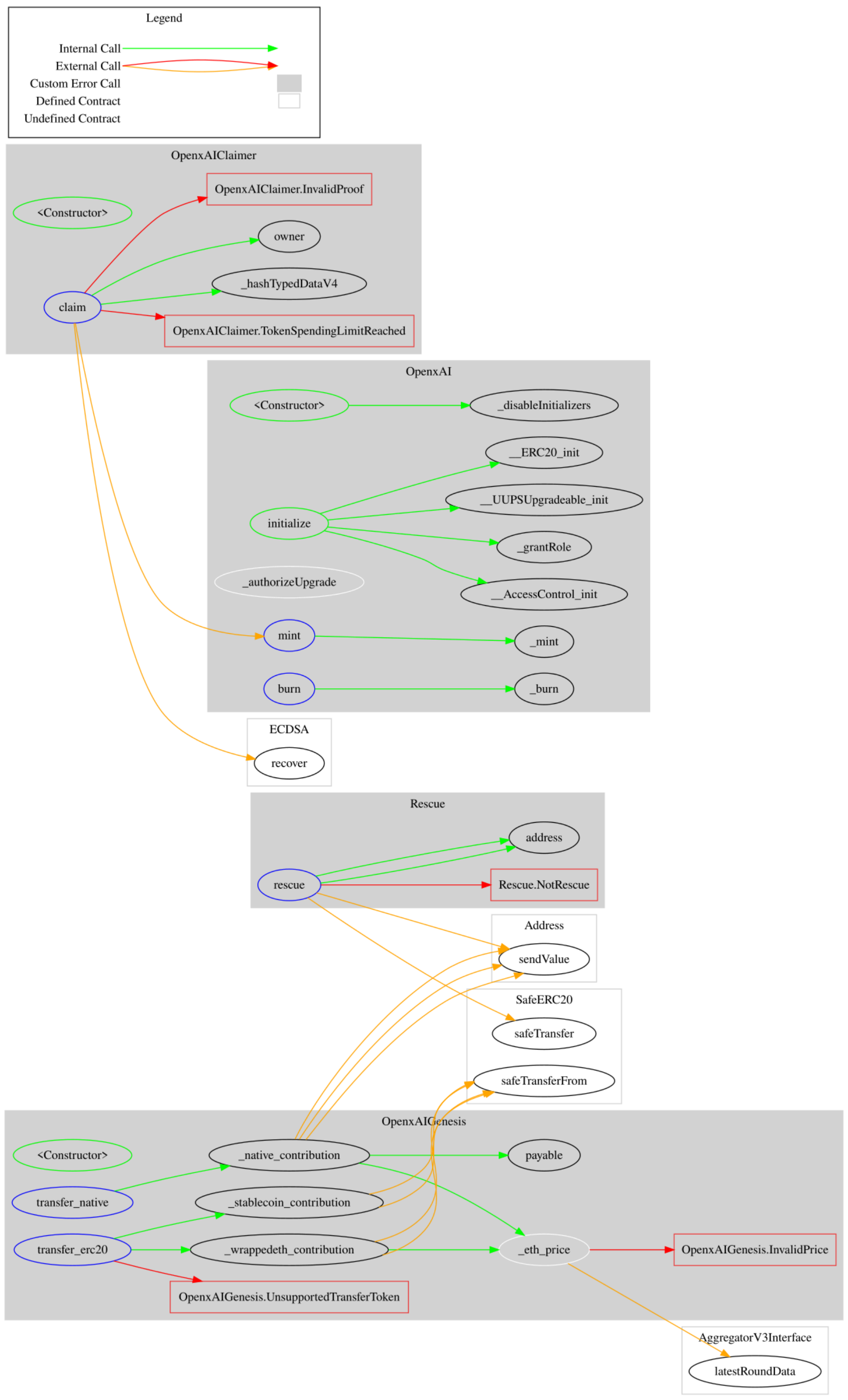
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Rescue	Implementation			
	rescue	External	✓	-
OpenxAIGenesis	Implementation	Rescue		
		Public	✓	-
	transfer_native	External	Payable	-
	transfer_erc20	External	✓	-
	_eth_price	Internal		
	_native_contribution	Internal	✓	
	_wrappedeth_contribution	Internal	✓	
	_stablecoin_contribution	Internal	✓	
OpenxAIClaimer	Implementation	Ownable, EIP712, Rescue		
		Public	✓	Ownable EIP712
	claim	External	✓	-
OpenxAI	Implementation	Initializable, UUPSUpgradeable, ERC20VotesUpgradeable, AccessContr		

		olUpgradeab le, IMintable		
		Public	✓	-
	initialize	Public	✓	initializer
	_authorizeUpgrade	Internal	✓	onlyRole
	mint	External	✓	onlyRole
	burn	External	✓	-
IMintable	Interface			
	mint	External	✓	-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Custom Error Call
Defined Contract
Undefined Contract
OpenxAIClaimer
<Constructor>
OpenxAIClaimer.InvalidProof
owner
_hashTypedDataV4
claim
OpenxAIClaimer.TokenSpendingLimitReached
OpenxAI
<Constructor>
_disableInitializers
__ERC20_init
__UUPSUpgradeable_init
initialize
_grantRole
_authorizeUpgrade
__AccessControl_init
mint
_mint
burn
_burn
ECDSA
recover
Rescue
address
rescue
Rescue.NotRescue
Address
sendValue
SafeERC20
safeTransfer
safeTransferFrom
OpenxAIGenesis
<Constructor>
_native_contribution
payable
transfer_native
_stablecoin_contribution
transfer_erc20
_wrappedeth_contribution
_eth_price
OpenxAIGenesis.InvalidPrice
OpenxAIGenesis.UnsupportedTransferToken
AggregatorV3Interface
latestRoundData

Summary

The OpenxAI ecosystem implements a decentralized token distribution and fundraising mechanism, integrating minting, claiming, and fund allocation functionalities. This audit investigates security vulnerabilities, business logic flaws, and potential improvements to ensure the system operates securely, efficiently, and in alignment with its intended functionality.

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