Cyberscope

Audit Report

Openmesh

November 2023

Repository https://github.com/Openmesh-Network/token-deployment/tree/main/contracts

Commit 3726daf86fa50a5c91a8028d2d08457489fc5e7d

Audited by © cyberscope

Table of Contents

Review

Repository	https://github.com/Openmesh-Network/token-deployment/tree/main/contracts
Commit	3726daf86fa50a5c91a8028d2d08457489fc5e7d

Audit Updates

Initial Audit	27 Nov 2023

Source Files

Filename	SHA256
VerifiedContributorStaking.sol	8329d345b9699dd0f701c55f6b929327f12972a2a38527dba1e44f0caf23bca6
VerifiedContributor.sol	8fee2c2633e04ab73fbea2a9733a6f3d04c9bfd39e29431b6aa4abc994cf4268
ValidatorPass.sol	033b9628e26775944031cf7d8b7eb9c5bef1805edaef5217b3c4ccf59933eeed
OpenStaking.sol	79357479a1fb6a893863264691b990cad77316d3302388ce0c945c11b1351a2c
OPEN.sol	de0eeaff58dd01df24c4f4c23dbea939d064fe53caf0b5b52464bba86dc053a9
IOpenStaking.sol	e93702e1adf513c965e5ad41d4fcbae4e424141902cef2bee8f2e343ef9b422e
IERC721Mintable.sol	711d9141d8bef2075ff797cb6864bca999a158a385632b3daad489bebf9f52a6

IERC20MintBurnable.sol	b88f47c61a7fc557acc2ec58a2d85c96225d2442013544ec537e118ca463347d
Fundraiser.sol	152617b459c6310ccee56aad6b4da95af6f69a8c5deda6a8b2eee7660538c17f

Test Contracts

Initial Audit, 27 Nov 2023

The following addresses represent the testnet contracts deployed on the Sepolia network

Contract Name	Contract Address
OPEN	https://sepolia.etherscan.io/address/0x9524c3dF8B40fbD55Eff4c6e48E4386B0Ec114F9
ValidatorPass	https://sepolia.etherscan.io/address/0xca05e4185a5e8d07dc7da06f6359db63fa090e33
VerifiedContributor	https://sepolia.etherscan.io/address/0xd6cb52d72c6f849b24b1228f1c16fe60f68e5177
Fundraiser	https://sepolia.etherscan.io/address/0x373E6a32B4279A2e263C879eCbCD0709Df4D72bA
OpenStaking	https://sepolia.etherscan.io/address/0xbea82A4c2f0315fFc5a3A5434c4bCcee781cd862
VerifiedContributor Staking	https://sepolia.etherscan.io/address/0x8F0749708fA011343e366713966612236640c4B8

Overview

OPEN functionality

The OPEN contract is an ERC20 token with a capped maximum supply, designed to be mintable up to a specified limit. It incorporates the ERC20Votes and AccessControl from OpenZeppelin, allowing for governance-related functionalities. The minting of these tokens can be managed by external smart contracts. Control over minting rights is centralized to a single address, which could be a DAO or a multisig wallet, with the ability to transfer these rights to a new address. Additionally, users have the capability to burn tokens from their own wallets. The contract ensures that minting does not exceed the maximum supply and provides interfaces for various ERC standards, enhancing its interoperability and functionality within the Ethereum ecosystem.

Validator Pass functionality

The Validator Pass contract is an ERC721 token, characterized by its non-transferability, ensuring that these tokens cannot be sold on marketplaces. It inherits from OpenZeppelin's ERC721 and AccessControl, and implements the IERC721Mintable interface. The contract is designed with a unique `MINT_ROLE`, allowing specific addresses (potentially a DAO or multisig wallet) to mint new tokens. The minting process is tracked by a counter (`mintCounter`), which increments with each new token minted. The contract also includes a metadata URI for the token, providing a link to its associated metadata. A key feature of this contract is the overridden `transferFrom` function, which includes a condition to prevent the transfer of tokens, enforcing their non-transferable nature. This design choice aligns with the intent to restrict the secondary market trading of these tokens, focusing on their utility within the designated ecosystem rather than as tradable assets.

Verified Contributor functionality

The Verified Contributor contract is an ERC721 token, designed with a focus on non-transferability to prevent trading on secondary markets. It extends OpenZeppelin's ERC721Enumerable and AccessControl, providing both a comprehensive token enumeration and flexible access control mechanisms. The contract defines two distinct roles the `MINT_ROLE` and `BURN_ROLE`, each represented by unique hashes. These roles enable specific addresses, potentially managed by a DAO or a multisig wallet, to mint

and burn tokens. The contract also includes a metadata URI, which links to the token's metadata, typically used for representing the token's attributes or artwork. Additionally this contract is also override the `transferFrom` function, which includes a check to prevent the transfer of tokens, thereby ensuring their non-transferability. This feature aligns with the intent to use these tokens within a specific ecosystem, focusing on their utility and role rather than as tradable assets. The mint and burn functions are marked as external and can only be called by addresses with the appropriate roles, providing a controlled environment for token creation and destruction.

Fundraiser functionality

The Fundraiser contract is a smart contract designed for a time-limited fundraising event, where participants contribute ETH and receive OPEN ERC20 tokens in return, based on a fixed rate that varies weekly. The contract includes a mapping to track individual contributions and arrays to define the tokens-per-Wei rate and the fundraising period's end dates. It employs two interfaces, the IERC20MintBurnable for the OPEN token and IERC721Mintable for the Validators Pass NFT. The contract sets minimum and maximum contribution limits per account and includes a fund storage address where the raised ETH will be sent post-fundraiser. The fundraising mechanism is triggered through fallback and receive functions, calculating the token amount based on the current week's rate. If a participant hits the maximum contribution limit, they are also mint an NFT. After the fundraising period, the contract facilitates the transfer of collected funds to the designated storage and burns any remaining OPEN tokens, ensuring the adherence to the hard cap. This contract effectively combines fundraising with incentivized participation through tiered rewards and exclusive NFT minting for maximum contributors.

OpenStaking functionality

The OpenStaking contract is a part of the OPEN token ecosystem, focusing on staking. The contract allows users to stake OPEN tokens by transferring them to the contract, which then burns these tokens, marking them as staked.

Withdrawals are unique, using EIP712 signatures for authentication, with an initial setup where the contract owner's signature is necessary for withdrawal approval. This process involves minting new OPEN tokens for the withdrawer. The contract also uses a nonce system for each address to prevent replay attacks.

This contract represents a mix of centralized control (via EOA signatures for withdrawals) and decentralized execution (staking and withdrawal processes on the blockchain), offering a secure and flexible approach for early-stage staking in the OPEN ecosystem.

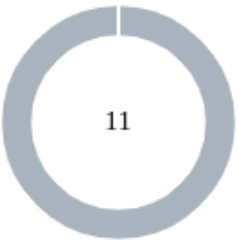
Verified Contributor Staking functionality

The Verified Contributor Staking contract rewards holders of a specific ERC721 (NFT)token, with ERC20 tokens over time. Staking an ERC721 token in this contract starts the accumulation of ERC20 rewards at a set rate per second. This continues until the token is unstaked or the staking period concludes. Rewards are minted in real-time upon claiming, ensuring accuracy and timeliness.

Key features include the flexibility for token holders to stake and unstake at will. Safeguards ensure that only the NFT's current owner can interact with the staking process, aligning with the non-transferable nature of the NFT. If the NFT is burned or transferred, staking benefits stop.

In summary, this contract incentivizes holding specific NFTs by rewarding holders with ERC20 tokens, striking a balance between simplicity and adaptability for staking non-transferable NFTs.

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	11

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	11	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	TOI	Timestamp Ordering Insurance	Unresolved
	PSFE	Potential Stake Functionality Enhancement	Unresolved
	DPF	Distinctive Presale Functionality	Unresolved
	CCR	Contract Centralization Risk	Unresolved
	MC	Missing Check	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L07	Missing Events Arithmetic	Unresolved
	L14	Uninitialized Variables in Local Scope	Unresolved
	L16	Validate Variable Setters	Unresolved
	L19	Stable Compiler Version	Unresolved
	L20	Succeeded Transfer Check	Unresolved

TOI - Timestamp Ordering Insurance

Criticality	Minor / Informative
Location	Fundraiser.sol#L27,62
Status	Unresolved

Description

The contract is setting the `end` array during the constructor. This array contains block timestamp values reflecting the weeks time. However, the contract does not verify if the values passed to the `end` array are in ascending order. As a result, if non-ordered values are passed, the if condition, which checks the current `block.timestamp` against the end array values, will fail if a value in the `end` array is less than its previous value.

```
   constructor(
        uint256[] memory _tokensPerWei,

        IERC20MintBurnable _token,
        IERC721Mintable _nft,
        address payable _fundStorage,
        uint32 _start,

        uint32[] memory _end,
        uint256 _minWeiPerAccount,
        uint256 _maxWeiPerAccount
    ) {
        tokensPerWei = _tokensPerWei;
        token = _token;
        nft = _nft;
        fundStorage = _fundStorage;
        start = _start;
        end = _end;
        ...
    }
    ...
function _currentTokensPerWei() public view returns (uint256) {
...
 if (block.timestamp < end[i]) {
                return tokensPerWei[i];
            }
    ...
    }
```

Recommendation

It is recommended to add additional checks during the setting of the `end` array to ensure that the values being passed are in ascending order. This validation will help prevent unexpected behavior in the contract and ensure that the conditional check comparing block timestamps with `end` array values is reliable. Adding input validation is crucial for maintaining the integrity and functionality of the contract.

PSFE - Potential Stake Functionality Enhancement

Criticality	Minor / Informative
Location	OpenStaking.sol#L26
Status	Unresolved

Description

The contract contains the `stake` function which transfers the user's tokens to the contract address and subsequently burns them. This burn of staked tokens poses a risk to accurately measuring the total value locked within the contract, since the total value locked is reduced, leading to an inability to accurately track the total value of the tokens within the contract.

```
    function stake(uint256 _amount) external {
        token.transferFrom(msg.sender, address(this), _amount);
        token.burn(_amount);
        emit TokensStaked(msg.sender, _amount);
    }
```

Recommendation

It is recommended to reconsider the `stake` function implementation. Instead of instantly burning the tokens, the contract could modify the logic to maintain a record of the total staked amount. Subsequently, rewards can be minted based on this accumulated total staked amount, ensuring a more accurate representation of the total value locked for each user.

DPF - Distinctive Presale Functionality

Criticality	Minor / Informative
Location	Fundraiser.sol#L84,92
Status	Unresolved

Description

The contract is designed to distribute tokens directly during the `fundraise` phase. Typically, in a presale context, tokens are not immediately released to buyers but are instead held until the project's owner adds liquidity to a decentralized exchange (DEX) like Uniswap. This approach ensures that the token's value is stabilized and prevents early buyers from setting up their own liquidity pools with potentially unfavorable or unbalanced exchange rates. If a user were to establish liquidity on Uniswap with disproportionate ratios compared to the presale rates, it could create market confusion and potentially discourage participation in the presale. Such actions could lead to a loss of confidence in the token's value and stability, adversely affecting the overall success of the presale.

```
function _fundraise() internal {
        ...

        token.transfer(msg.sender, msg.value * _currentTokensPerWei());
        if (personalContribution == maxWeiPerAccount) {
            nft.mint(msg.sender);
        }
        ...
    }

    function fundsToStorage() external {
        if (block.timestamp < end[end.length - 1]) {
            revert FundraiserNotOverYet();
        }

        (bool succes, ) = fundStorage.call{value:
address(this).balance}("");
        if (!succes) {
            revert FundStorageReverted();
        }

        ...
    }
```

Recommendation

It is recommended to modify the `fundraise` process to withhold the immediate distribution of tokens. Instead, tokens should be released to presale participants only after the project owner has successfully added liquidity to a DEX. This approach will help maintain a consistent and fair market value for the token, aligned with the presale valuation. Additionally, implementing a lock-up period or vesting schedule for presale tokens could further prevent early participants from disproportionately influencing the market. These changes will likely enhance investor confidence and participation in the presale, contributing to a more stable and successful token launch.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	OPEN.sol#L47ValidatorPass.sol#L42 VerifiedContributor.sol#L48 Fundraiser.sol IOpenStaking.sol#L33 VerifiedContributorStaking.sol
Status	Unresolved

Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

Specifically, the contract is currently structured in a way that centralizes significant control and authority in the hands of the contract owner, along with designated minter and burner accounts. The owner has the exclusive ability to set addresses authorized to mint tokens and NFTs to specific target addresses, and to burn NFTs. Additionally, the owner have the authority to transfer tokens to the `Fundraiser` contract and to execute withdrawals through the `OpenStaking` contract. Furthermore, the owner have to supply the `rewardToken` to the `VerifiedContributorStaking`, which is essential for users to claim their rewards, and also sets the end time of the staking period. This concentration of power in a single role raises concerns about potential misuse or single points of failure, which could undermine the trust and security of the contract.

```
    function mint(
        address account,
        uint256 amount
    ) external onlyRole(MINT_ROLE) {
        if (totalSupply() + amount > maxSupply) {
            revert SurpassMaxSupply();
        }

        _mint(account, amount);
    }

    function mint(address account) external onlyRole(MINT_ROLE) {
        _mint(account, mintCounter++);
    }

    function mint(
        address to,
        uint256 tokenId
    ) external virtual onlyRole(MINT_ROLE) {
        _mint(to, tokenId);
    }

    function burn(uint256 tokenId) external virtual
onlyRole(BURN_ROLE) {
        _burn(tokenId);
    }
    function withdraw(
        ...
    ) external {
        ...
        if (signer != owner()) {
            revert InvalidProof();
        }

        token.mint(_withdrawer, _amount);
        ...
    }
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's

self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

MC - Missing Check

Criticality	Minor / Informative
Location	Fundraiser.sol#L25
Status	Unresolved

Description

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

Specifically, the contract sets the `minWeiPerAccount` and `maxWeiPerAccount` variables during its constructor, but lacks a crucial check to ensure `_minWeiPerAccount` is less than `_maxWeiPerAccount`. This oversight could lead to operational issues. Additionally, the contract uses `_end` and `_tokensPerWei` arrays to denote week numbers and corresponding token rewards, but it doesn't verify if these arrays are of equal length, risking reward distribution errors.

```
    constructor(
        uint256[] memory _tokensPerWei,
        ...
        uint32[] memory _end,
        uint256 _minWeiPerAccount,
        uint256 _maxWeiPerAccount
    ) {
        tokensPerWei = _tokensPerWei;
        ...
        end = _end;
        minWeiPerAccount = _minWeiPerAccount;
        maxWeiPerAccount = _maxWeiPerAccount;
    }
```

Recommendation

The team is advised to properly check the variables according to the required specifications. It is recommended to ensure that `_minWeiPerAccount` is always less than `_maxWeiPerAccount` to maintain logical consistency, and verify that length of `_end` and `_tokensPerWei` arrays are equal to align week numbers with their

respective token rewards, ensuring accurate reward distribution. These improvements will significantly bolster the contract's reliability and functionality.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	VerifiedContributorStaking.sol#L36,49,62,86,95 VerifiedContributor.sol#L27 OpenStaking.sol#L26,34,35,36,37,38,64 OPEN.sol#L34Fundraiser.sol#L53
Status	Unresolved

Description

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

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

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

```
uint256 _tokenId
uint64 _stakingOver
bytes4 _interfaceId
uint256 _amount
uint8 _v
bytes32 _r
bytes32 _s
address _withdrawer
address _account

...
```

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/v0.8.17/style-guide.html#naming-convention.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	VerifiedContributorStaking.sol#L100
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
stakingOver = _stakingOver
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	Fundraiser.sol#L58
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.

```
uint i
```

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	Fundraiser.sol#L38
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.

```
fundStorage = _fundStorage
```

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	VerifiedContributorStaking.sol#L2 VerifiedContributor.sol#L2ValidatorPass.sol#L2 OpenStaking.sol#L2OPEN.sol#L2 IOpenStaking.sol#L2Fundraiser.sol#L2
Status	Unresolved

Description

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

```
pragma solidity ^0.8.0;
```

Recommendation

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

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	OpenStaking.sol#L27 Fundraiser.sol#L84
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.

```
token.transferFrom(msg.sender, address(this), _amount)
token.transfer(msg.sender, msg.value * _currentTokensPerWei())
```

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.

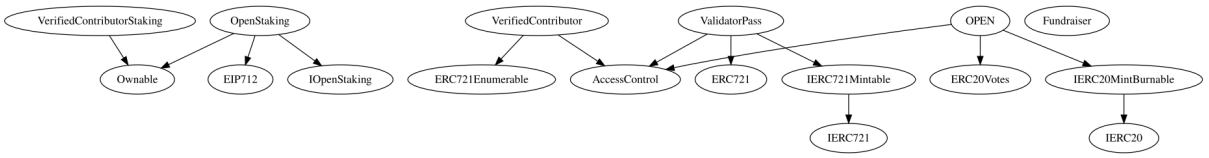
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
VerifiedContributorStaking	Implementation	Ownable		
		Public	✓	Ownable
	stake	External	✓	-
	unstake	External	✓	-
	claimable	Public		-
	claim	External	✓	-
	setStakingEnd	External	✓	onlyOwner
	_toUint64	Internal		
	_claim	Internal	✓	
VerifiedContributor	Implementation	ERC721Enumerable, AccessControl		
		Public	✓	ERC721
	supportsInterface	Public		-
	tokenURI	Public		-
	mint	External	✓	onlyRole
	burn	External	✓	onlyRole
	transferFrom	Public	✓	-

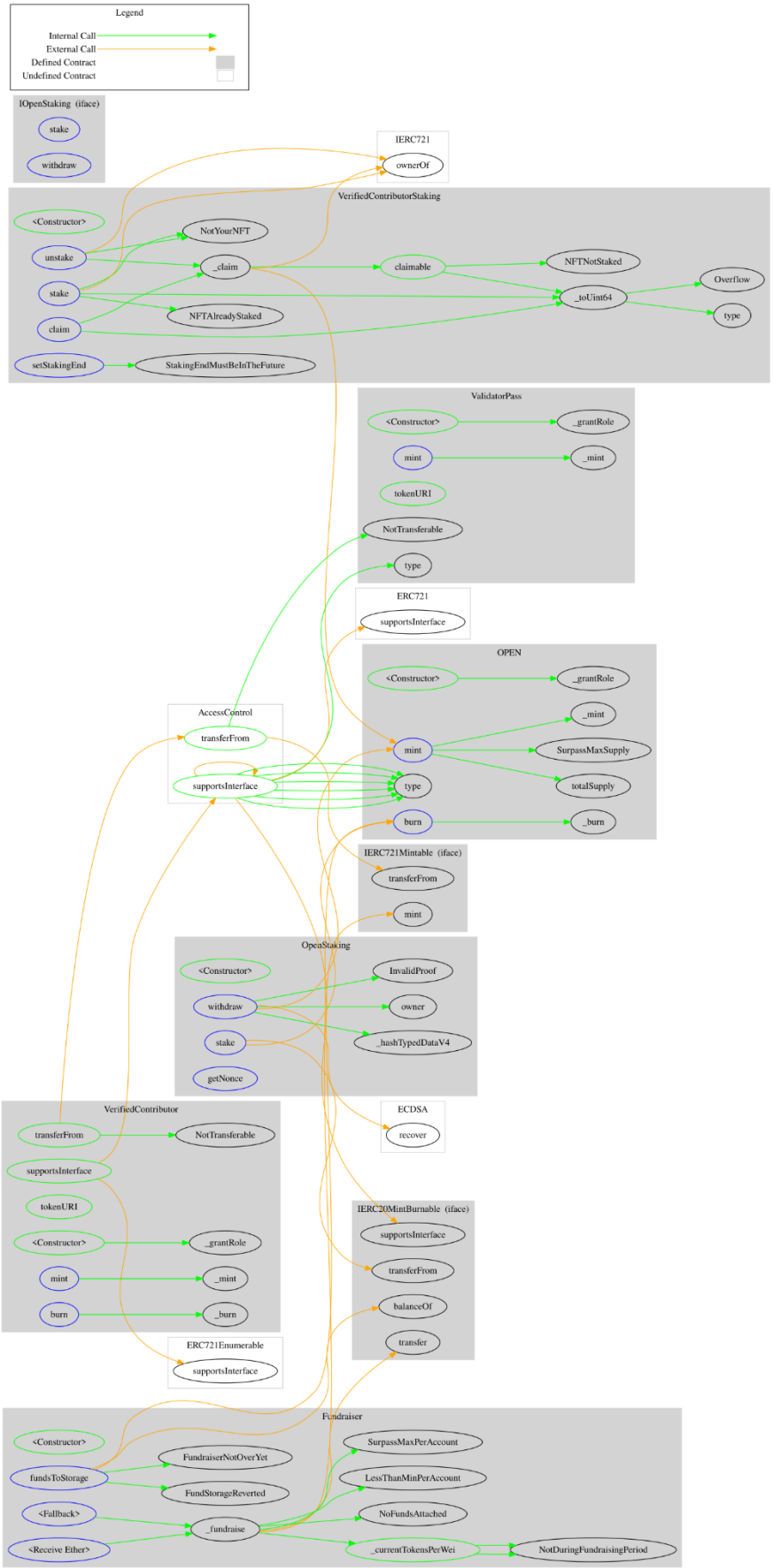
ValidatorPass	Implementation	ERC721, AccessControl, IERC721Mintable		
		Public	✓	ERC721
	supportsInterface	Public		-
	mint	External	✓	onlyRole
	tokenURI	Public		-
	transferFrom	Public	✓	-
OpenStaking	Implementation	Ownable, EIP712, IOpenStaking		
		Public	✓	Ownable EIP712
	stake	External	✓	-
	withdraw	External	✓	-
	getNonce	External		-
OPEN	Implementation	ERC20Votes, AccessControl, IERC20MintBurnable		
		Public	✓	ERC20 EIP712
	supportsInterface	Public		-
	mint	External	✓	onlyRole
	burn	External	✓	-

IOpenStaking	Interface			
	stake	External	✓	-
	withdraw	External	✓	-
IERC721Mintable	Interface	IERC721		
	mint	External	✓	-
IERC20MintBurnable	Interface	IERC20		
	mint	External	✓	-
	burn	External	✓	-
Fundraiser	Implementation			
		Public	✓	-
		External	Payable	-
		External	Payable	-
	_currentTokensPerWei	Public		-
	_fundraise	Internal	✓	
	fundsToStorage	External	✓	-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IOpenStaking (iface)
stake
withdraw
IERC721
ownerOf
VerifiedContributorStaking
<Constructor>
NotYourNFT
unstake
_claim
claimable
NFTNotStaked
Overflow
stake
_toUint64
type
NFTAlreadyStaked
claim
setStakingEnd
StakingEndMustBeInTheFuture
ValidatorPass
<Constructor>
_grantRole
mint
_mint
tokenURI
NotTransferable
type
ERC721
supportsInterface
OPEN
<Constructor>
_grantRole
_mint
AccessControl
transferFrom
mint
SurpassMaxSupply
supportsInterface
type
totalSupply
burn
_burn
IERC721Mintable (iface)
transferFrom
mint
OpenStaking
<Constructor>
InvalidProof
withdraw
owner
stake
_hashTypedDataV4
getNonce
VerifiedContributor
transferFrom
NotTransferable
ECDSA
recover
supportsInterface
tokenURI
IERC20MintBurnable (iface)
<Constructor>
_grantRole
supportsInterface
mint
_mint
transferFrom
burn
_burn
balanceOf
transfer
ERC721Enumerable
supportsInterface
Fundraiser
<Constructor>
SurpassMaxPerAccount
FundraiserNotOverYet
fundsToStorage
LessThanMinPerAccount
FundStorageReverted
<Fallback>
NoFundsAttached
_fundraise
<Receive Ether>
_currentTokensPerWei
NotDuringFundraisingPeriod

Summary

The Openmesh contract introduces a unique staking mechanism, integrating both token and NFT assets, to incentivize network participation and validate transactions. This audit focuses on identifying security vulnerabilities, evaluating the business logic, and suggesting potential enhancements to ensure robustness and efficiency.

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

https://www.cyberscope.io