Cyberscope

Audit Report

eXchange1

October 2025

SHA256 :

f3a0ed397dd219acb1c91f58390d418031eeb008144ad0c4cefddf250cffc585

Audited by © cyberscope

Table of Contents

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
Critical	Highly Likely / High Impact
Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
Minor / Informative	Unlikely / Low to no Impact

Review

Audit Updates

Initial Audit	17 Oct 2025

Source Files

Filename	SHA256
ex1_cyberscope.zip	f3a0ed397dd219acb1c91f58390d418031eeb008144ad0c4cefddf250cffc585

Overview

The suite of five smart contracts, EX1, Ex1ICO , ICOVesting, Ex1Staking, and PrivateVesting, collectively forms a comprehensive ecosystem for managing the issuance, sale, vesting, staking, and private allocation of the EX1 token, ensuring secure, transparent, and controlled token distribution. Built with OpenZeppelin libraries, the EX1 contract is a multi-signature ERC20 token with upgradeable functionality, enforcing restricted address transfers through a multi-approval process. The Ex1ICO contracts facilitate token sales with customizable ICO stages, price feeds, and purchase limits. The ICOVesting contract manages linear vesting of ICO-purchased tokens, allowing periodic claims, while the Ex1Staking contract incentivizes long-term holding by enabling users to stake ICO tokens for dynamic rewards before vesting begins. The PrivateVesting contract handles linear vesting for non-ICO participants, with flexible schedules and revocation options. Together, these contracts provide a robust, role-based, and upgradeable framework for token management, cross-chain sales, and incentivized holding, tailored for both public and private stakeholders.

ex1Token file

The EX1 contract is a multi-signature ERC20 token contract with upgradeable functionality, designed to ensure secure and controlled token transfers through a robust governance framework. Built on OpenZeppelin's Initializable, ERC20Upgradeable, AccessControlUpgradeable, and UUPSUpgradeable standards, it implements a multi-step process for proposing, approving, and executing transfers, particularly for restricted addresses, requiring a predefined number of approvals from designated approvers. It features three roles, `OWNER_ROLE` for managing restricted addresses and approvers, `APPROVER_ROLE` for voting on proposals, and `EXECUTOR_ROLE` for finalizing transfers, along with mechanisms to manage restricted address lists, update approver settings, and query pending proposals. This ensures transparent, secure, and decentralized token management, with flexibility for future upgrades.

ex1ICOv2 file

The Ex1ICO contract is an upgradeable, role-based smart contract designed to manage Initial Coin Offerings (ICOs) for the EX1 token, enabling secure token sales with support for

USDC, USDT, ETH, and BTC payments. Leveraging OpenZeppelin's Initializable, ReentrancyGuardUpgradeable, AccessControlUpgradeable, and UUPSUpgradeable contracts, it facilitates the creation and management of ICO stages with customizable parameters like token price and duration, while integrating Chainlink price feeds for real-time ETH and BTC pricing. The contract supports on-chain purchases with USDC/USDT/ETH, and off-chain BTC transactions recorded by authorized roles, and tracks tokens sold, funds raised, and buyer data, with purchase limits and administrative controls for setting parameters, ensuring a scalable, secure, and transparent token sale process.

ex1ICOVesting file

The ICOVesting contract is an upgradeable, role-based smart contract designed to manage the vesting and claiming of EX1 tokens purchased during an ICO, ensuring rewards are distributed over time on a linear basis. Built with OpenZeppelin libraries, it enables authorized users to create and update vesting schedules for specific ICO stages, defining start/end times, claim intervals, and slice periods for gradual token release. Token holders can claim their vested tokens based on these periodic schedules, with claimable amounts calculated linearly according to elapsed time, and reentrancy protection ensures security. Integrated with the Ex1ICO contract to verify user deposits, it supports role-based access (`OWNER` , `UPGRADER` , `VESTING_AUTHORISER`) for administrative tasks like updating schedules and interfaces, offering a secure and flexible vesting solution.

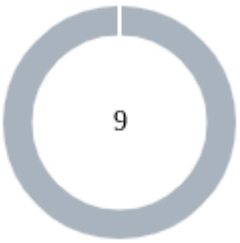
ex1Staking file

The Ex1Staking contract is an upgradeable smart contract designed to facilitate staking of EX1 tokens based on the amount purchased during an ICO and before the vesting period starts, incentivizing long-term holding through rewards. Built with OpenZeppelin libraries, it allows users to choose to stake their tokens, with staking parameters (percentage return, time period, and ICO stage) set by authorized roles. Rewards are calculated dynamically based on the staked amount and elapsed time, and users can claim them or unstake tokens at any point, with rewards adjusted accordingly. Integrated with the Ex1ICO and ICOVesting contracts to verify deposits and vesting schedules, it allows the staking of the user deposits before the vesting. It supports role-based access (`OWNER` , `UPGRADER` , `STAKING_AUTHORISER`) for administrative updates, ensuring secure and transparent staking operations.

ex1PrivateVesting file

The PrivateVesting contract is an upgradeable, role-based smart contract designed to manage token vesting schedules with role-based access control, specifically for addresses that do not participate in the ICO, ensuring controlled token distribution over time on a linear basis. Built with OpenZeppelin libraries, it allows authorized users to create, update, and revoke vesting schedules for beneficiaries, defining parameters like total amount, start/end times, claim intervals, cliff periods, and slice periods for gradual token release. Beneficiaries can claim vested tokens periodically after the cliff period, with claimable amounts calculated linearly based on elapsed time. The contract supports role-based access (`OWNER` , `UPGRADER` , `VESTING_CREATOR`) for administrative tasks, tracks vesting schedules and claim histories, and includes revocation options for revocable schedules, providing a secure and flexible vesting solution for private allocations.

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	9

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	0	9	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	BC	Blacklists Addresses	Acknowledged
●	CCR	Contract Centralization Risk	Acknowledged
●	IDU	Inconsistent Data Updates	Acknowledged
●	IVS	Inefficient Vesting Setup	Acknowledged
●	POSD	Potential Oracle Stale Data	Acknowledged
●	PUR	Potentially Unclaimed Rewards	Acknowledged
●	TSI	Tokens Sufficiency Insurance	Acknowledged
●	UIC	Unreachable If Condition	Acknowledged
●	L13	Divide before Multiply Operation	Acknowledged

BC - Blacklists Addresses

Criticality	Minor / Informative
Location	ex1Token.sol#L173,229
Status	Acknowledged

Description

The `OWNER_ROLE` has the authority to stop addresses from transactions. The owner may take advantage of it by calling the `addRestrictedAddress` function.

```
Shell
function addRestrictedAddress(address _address) external
onlyRole(OWNER_ROLE) {
        restrictedAddresses.add(_address);
}

function transfer(address _to, uint256 _value) public virtual
override returns (bool) {
        if (isAddressRestricted(_msgSender()) == true) {
            _proposeTransfer(_msgSender(), _to, _value);
        } else {
            _transfer(_msgSender(), _to, _value);
        }
        return true;
}

function transferFrom(address _from, address _to, uint256
_value) public virtual override returns (bool) {
        if (isAddressRestricted(_from) == true) {
            _proposeTransfer(_from, _to, _value);
        } else {
            address spender = _msgSender();
            _spendAllowance(_from, spender, _value);
            _transfer(_from, _to, _value);
        }
return true; }
```

Recommendation

The team should carefully manage the private keys of the `OWNER_ROLE'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.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	ex1ICOv2.sol ex1Staking.sol ex1PrivateVesting.sol ex1ICOVesting.sol
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.

Specifically, the contract is heavily reliant on centralized role authorities, such as `OWNER_ROLE`, `UPGRADER_ROLE`, and `ICO_AUTHORISER_ROLE`, to configure, modify, and manage critical contract parameters. This centralization introduces a potential risk. If the individuals or entities holding these roles make an error, act maliciously, or fail to act in a timely manner, the entire contract and its associated processes can be compromised. Without proper checks and balances, the system's integrity depends on the correct and honest execution of these centralized roles, making it vulnerable to human error or abuse of power.

Shell

```
bytes32 public constant OWNER_ROLE = keccak256("OWNER_ROLE");
bytes32 public constant UPGRADER_ROLE =
keccak256("UPGRADER_ROLE");
bytes32 public constant ICO_AUTHORISER_ROLE =
keccak256("ICO_AUTHORISER_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.

IDU - Inconsistent Data Updates

Criticality	Minor / Informative
Location	ex1ICOv2.sol#L234 ex1ICOVesting.sol#L117 ex1Staking.sol#L101 ex1PrivateVesting.sol#L208
Status	Acknowledged

Description

The contract is managing multiple functionalities, presale, vesting, and staking, based on a shared `_icoStageID`. However, updates to variables such as the stage parameters (e.g. in ICO) occur independently within each contract, without a mechanism to propagate these changes across the related functionalities. For example, if the ICO stage parameters are modified in one contract, other contracts that depend on these parameters may not reflect the updated values. This disconnect can lead to inconsistencies, as calculations or conditions in one contract may no longer align with the updated data in another.

Shell
```
function updateICOStage(
    uint256 _stageID,
    uint256 _startTime,
    uint256 _endTime,
    uint256 _tokenPriceUSD
) external onlyRole(ICO_AUTHORISER_ROLE) {

...
```

Shell
```
function claimTokens( uint256 _icoStageID ) external
nonReentrant {
...
```

```
uint256 deposits = icoInterface.UserDepositsPerICOStage(
_icoStageID, _msgSender());
require(deposits > 0, "ex1Presale: No Tokens to Claim!");
...
```

Shell

```
function stake( uint256 _icoStageID ) external returns(bool) {
      uint256 deposits =
      icoInterface.UserDepositsPerICOStage(_icoStageID,
      _msgSender());
      ( , uint256 startTime, , , ) =
      vestingInterface.claimSchedules(_icoStageID);
      ...
}
```

Recommendation

It is recommended to implement a decentralized mechanism or shared interface that ensures changes to ICO stage parameters are consistently applied across all related contracts. This could involve storing the parameters in a single source of truth or using events and callbacks to notify dependent contracts of updates. By aligning data and logic across all functionalities, the system can prevent inconsistencies, reduce maintenance overhead, and improve reliability.

IVS - Inefficient Vesting Setup

Criticality	Minor / Informative
Location	ex1PrivateVesting.sol#L147
Status	Acknowledged

Description

The current approach requires a central role (VESTING_CREATOR_ROLE) to manually set the vesting schedules for all participating users. This centralized process is inefficient and gas-intensive, especially when the number of beneficiaries increases. Additionally, handling multiple user entries in this manner increases the risk of errors or incorrect data entries, potentially causing mismanagement of vesting schedules and related funds.

Shell

```
function setVestingSchedule(
    address _beneficiary,
    uint256 _totalAmount,
    uint256 _startTime,
    uint256 _endTime,
    uint256 _claimInterval,
    uint256 _cliffPeriod,
    uint256 _slicePeriod,
    bool _isRevocable
    ) external onlyRole(VESTING_CREATOR_ROLE) {
        ...
        latestVestingScheduleID ++;

            vestingSchedules[latestVestingScheduleID] =
           VestingSchedule({
            beneficiary: _beneficiary,
            vestingScheduleID: latestVestingScheduleID,
            totalAmount: _totalAmount,
            startTime: _startTime,
            endTime: _endTime,
            claimInterval: _claimInterval,
            cliffPeriod: _cliffPeriod,
```

```
            slicePeriod: _slicePeriod,
            releasedAmount: 0,
            isRevocable: _isRevocable,
            isRevoked: false
        });

        ...
    }
```

Recommendation

It is recommended to consider a more decentralized approach to vesting schedule creation and validation. By allowing users or their authorized agents to initiate their own vesting schedules—while still adhering to contract-defined rules and checks—this would reduce the gas cost per action and minimize the administrative overhead for a single role. Implementing self-service or automated validation mechanisms can enhance efficiency and reduce the likelihood of incorrect data entries.

Team Update

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

Allocation targets specific wallets (treasury, liquidity, marketing), only ~8 beneficiaries.

POSD - Potential Oracle Stale Data

Criticality	Minor / Informative
Location	ex1ICOv2.sol#L272
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.

```shell
function getLatestETHPrice() public view returns (uint256) {
        (, int256 price, , , ) =
aggregatorInterfaceETH.latestRoundData();
        return uint256(price);
}


function getTokenPriceInETH( uint256 amount, uint256
_icoStageID ) public view returns  (uint256 ethAmount) {
      uint256 usdPrice = calculatePrice(amount,_icoStageID);
      uint256 latestEthPrice = getLatestETHPrice() *
     (10 ** 10);
     uint256 _ethAmount = (usdPrice * (10 ** 18)) /
     latestEthPrice;
      return _ethAmount;
 }
```

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.

PUR - Potentially Unclaimed Rewards

Criticality	Minor / Informative
Location	ex1Staking.sol#L200
Status	Acknowledged

Description

The `stake` method distributes rewards based on the duration each user has staked. The `userRewardPerSecond` is calculated proportionally to the user's deposit and the full `timePeriodInSeconds` of the staking stage. This approach may lead to inconsistencies in reward distribution, as users who stake for shorter durations still have their rewards spread across the entire staking period. Consequently, users who stake late in the stage may only be able to claim a small portion of their expected rewards.

```shell
uint256 userRewardPerSecond = userPercentage /
stakingParameters[_icoStageID].timePeriodInSeconds;
```

Recommendation

The team is advised to validate the implementation logic of the reward distribution to ensure it aligns with the intended design. Alternatively, the team may consider distributing rewards proportionally based on the remaining time period.

TSI - Tokens Sufficiency Insurance

Criticality	Minor / Informative
Location	ex1ICOv2.sol#L370 ex1ICOVesting.sol#L145,181 ex1Staking.sol#L132,307 ex1Token.sol#L349
Status	Acknowledged

Description

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

Shell
```
bool success = IERC20(ex1Token).transfer(_recipient, _amount);
require( success, "Token Transfer Failed!" );
```

Shell
```
ex1Token.safeTransfer(_msgSender(), claimableAmount);
```

Shell
```
bool success = IERC20(ex1Token).transfer(_msgSender(),
reward);
```

Recommendation

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

UIC - Unreachable If Condition

Criticality	Minor / Informative
Location	ex1ICOVesting.sol#L181 ex1PrivateVesting.sol#L292
Status	Acknowledged

Description

The contract includes an `if` condition that checks whether the number of elapsed vesting slices is equal to the total number of slices (`if (elapsedSlices == totalNumberOfSlices)`) in order to unlock the full deposit. However, this condition is logically unreachable due to a preceding check: `if (block.timestamp >= schedule.endTime)` , which is triggered before the slice comparison is evaluated. Since reaching the total number of slices implies that the current timestamp is equal to or beyond the schedule's `endTime` , the earlier `if` condition will always be satisfied first. This makes the later check redundant and introduces ambiguity about the intended logic, potentially leading to confusion or maintenance issues in future updates.

```
Shell
function calculateClaimableAmount(address _caller, uint256
_icoStageID) public view returns (uint256) {
        ...

uint256 totalNumberOfSlices = (schedule.endTime -
schedule.startTime) / schedule.slicePeriod;

if (block.timestamp >= schedule.endTime) {
return totalDeposits - claimedAmount[_icoStageID][_caller];
}

uint256 elapsedSlices =
prevClaimTimestamp[_icoStageID][_caller] == 0
? (block.timestamp - schedule.startTime) /
schedule.slicePeriod
: (block.timestamp - prevClaimTimestamp[_icoStageID][_caller])
/ schedule.slicePeriod;
```

```
uint256 unlocked = totalDeposits * elapsedSlices /
totalNumberOfSlices;

if (elapsedSlices == totalNumberOfSlices) {
unlocked = totalDeposits;
}

return unlocked;
}
```

Shell

```
function calculateClaimableAmount(uint256 _vestingScheduleID)
public
view
onlyValidSchedule(_vestingScheduleID)
notRevoked(_vestingScheduleID)
returns (uint256)
{
...
if (elapsed == totalSlices) {
unlocked = schedule.totalAmount;
}
return unlocked - schedule.releasedAmount;
}
```

Recommendation

It is recommended to reconsider the code structure and remove the unreachable `if (elapsedSlices == totalNumberOfSlices)` condition. This will simplify the logic and clarify the intended flow, ensuring that all conditions contribute meaningfully to the function's behaviour.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	ex1Staking.sol#L153,172,175,180,183,200,221,225,231,237,260,279,282,287,290 ex1PrivateVesting.sol#L313,318 ex1ICOVesting.sol#L196,200
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.

```shell
uint256 userRewardPerSecond = userPercentage /
stakingParameters[_icoStageID].timePeriodInSeconds
reward =
(stakingEndTime -
previousStakingRewardClaimTimestamp[_icoStageID][_caller]) *
userRewardPerSecond)

uint256 elapsed = (block.timestamp - schedule.startTime) /
schedule.slicePeriod
uint256 unlocked = (schedule.totalAmount * elapsed) /
totalSlices

uint256 unlocked = totalDeposits * elapsedSlices /
totalNumberOfSlices
uint256 elapsedSlices =
prevClaimTimestamp[_icoStageID][_caller] == 0
? (block.timestamp - schedule.startTime) /
schedule.slicePeriod
```

```
: (block.timestamp - prevClaimTimestamp[_icoStageID][_caller])
/ schedule.slicePeriod
```

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.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Ex1Staking	Implementation	Initializable, ReentrancyGuardUpgradeable, AccessControlUpgradeable, UUPSUpgradeable		
		Public	✓	-
	initialize	Public	✓	initializer
	createStakingRewardsParamaters	External	✓	onlyRole
	stake	External	✓	nonReentrant
	claimStakingRewards	External	✓	nonReentrant
	_calculateStakeRewardView	Internal		
	calculateStakeReward	Internal	✓	
	viewClaimableRewards	External		-
	getEligibleStakableToken	Public		-
	unstake	External	✓	-
	updateIcoInterface	External	✓	onlyRole
	updateVestingInterface	External	✓	onlyRole
	updateEX1Token	External	✓	onlyRole
	_authorizeUpgrade	Internal	✓	onlyRole

PrivateVesting	Implementation	Initializable, AccessControlUpgradeable, ReentrancyGuardUpgradeable, UUPSUpgradeable		
		Public	✓	-
	initialize	Public	✓	initializer
	setVestingSchedule	External	✓	onlyRole
	updateVesting	External	✓	onlyValidSchedule onlyRole
	claimTokens	External	✓	nonReentrant onlyValidSchedule notRevoked
	calculateClaimableAmount	Public		onlyValidSchedule notRevoked
	revokeSchedule	External	✓	onlyRole onlyValidSchedule notRevoked
	nextClaimTime	Public		onlyValidSchedule notRevoked
	getBalanceLeftToClaim	Public		-
	getAllVestingSchedules	Public		-
	getBeneficiarySchedules	External		-
	setEx1TokenSaleContract	External	✓	onlyRole
	_authorizeUpgrade	Internal	✓	onlyRole
Ex1ICO	Implementation	Initializable, ReentrancyGuardUpgradeable, AccessControlUpgradeable, UUPSUpgradeable		

		Public	✓	-
	initialize	Public	✓	initializer
	_authorizeUpgrade	Internal	✓	onlyRole
	createICOStage	External	✓	onlyRole
	getAllICOStages	External		-
	nextICOStageID	External		-
	getCurrentOrNextActiveICOStage	External		-
	updateICOStage	External	✓	onlyRole
	getLatestETHPrice	Public		-
	getLatestBTCPrice	Public		-
	getTokenPriceInETH	Public		-
	getTokenPriceInBTC	External		-
	calculatePrice	Public		-
	buyWithUSDC	External	✓	checkSaleStatus checkAddressNotRestricted
	buyWithUSDT	External	✓	checkSaleStatus checkAddressNotRestricted
	_processTokenAllocation	Internal	✓	
	_buyWithUSD	Internal	✓	checkSaleStatus checkAddressNotRestricted
	purchasedViaEth	External	Payable	nonReentrant checkSaleStatus checkAddressNotRestricted
	purchasedViaBTC	External	✓	checkSaleStatus checkAddressN

				otRestricted onlyRole
	setMaxTokenLimitPerAddress	External	✓	onlyRole
	setTokenSaleAddress	External	✓	onlyRole
	setMaxTokenLimitPerTransaction	External	✓	onlyRole
	setReceiverWallet	External	✓	checkZeroAddress onlyRole
	setTokenReleasable	External	✓	onlyRole
	setIAggregatorInterfaceETH	External	✓	checkZeroAddress onlyRole
	setIAggregatorInterfaceBTC	External	✓	checkZeroAddress onlyRole
	setUSDTAddress	External	✓	checkZeroAddress onlyRole
	setUSDCAddress	External	✓	checkZeroAddress onlyRole
	withdraw	External	✓	onlyRole
ICOVesting	Implementation	Initializable, AccessControlUpgradeable, ReentrancyGuardUpgradeable, UUPSUpgradeable		
		Public	✓	-
	initialize	Public	✓	initializer
	setClaimSchedule	External	✓	onlyRole
	updateClaimSchedule	External	✓	onlyRole
	claimTokens	External	✓	nonReentrant
	calculateClaimableAmount	Public		-

	nextClaimTime	Public		-
	getBalanceLeftToClaim	Public		-
	updateInterface	External	✓	onlyRole
	updateEX1Token	External	✓	onlyRole
	_authorizeUpgrade	Internal	✓	onlyRole
EX1	Implementation	Initializable, ERC20Upgradeable, AccessControlUpgradeable, UUPSUpgradeable		
		Public	✓	-
	initialize	Public	✓	initializer
	removeApprovers	External	✓	onlyRole
	addApprover	External	✓	onlyRole
	addRestrictedAddress	External	✓	onlyRole
	removeRestrictedAddress	External	✓	onlyRole
	isAddressRestricted	Public		-
	checkRestrictedAddress	Public		-
	checkApproversList	Public		-
	checkApprovers	Public		-
	transfer	Public	✓	-
	transferFrom	Public	✓	-
	_proposeTransfer	Internal	✓	

	approveTransfer	Public	✓	onlyRole txExists notExecuted notRevoked
	revokeApproval	Public	✓	onlyRole txExists notExecuted notRevoked
	_executeTransfer	Internal	✓	txExists notExecuted
	executeTransfer	External	✓	txExists notExecuted onlyRole
	getAllPendingProposals	External		-
	updateRequiredApprovers	External	✓	onlyRole
	_authorizeUpgrade	Internal	✓	onlyRole

Inheritance Graph

The inheritance graph for the eXchange1 contracts can be accessed through the link.

Flow Graph

The flow graph for the eXchange1 contracts can be accessed through the link.

Summary

The eXchange1 suite of smart contracts implements a comprehensive, role-based, and upgradeable ecosystem for secure EX1 token issuance, cross-chain ICO sales, linear vesting, staking, and private allocations, leveraging OpenZeppelin libraries and price feeds. This audit investigates security vulnerabilities, business logic inconsistencies, and potential optimizations across the EX1, Ex1ICO, ICOVesting, Ex1Staking, and PrivateVesting contracts to ensure robust and transparent token management.

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