Cyberscope

Audit Report

Findex Exchange

December 2023

Network	ETH
TokenContract	0xb77bC8B14D6F12a5B847379bA4eE5119564cB1b6
TokenVesting	0xFA6526E7AA86178995F51689F334646620D76247
Audited by	© cyberscope

Table of Contents

Review

Explorer (TokenVesting)	https://etherscan.io/address/0xFA6526E7AA86178995F51689F334646620D76247
Explorer (TokenContract)	https://etherscan.io/address/0xb77bC8B14D6F12a5B847379bA4eE5119564cB1b6

Audit Updates

Initial Audit	04 Dec 2023

Source Files

Filename	SHA256
TokenVesting.sol	78fb9ce7591f94d93de84b2160316b23665df25acf6087af0930ef5414cabbe0
TokenContract.sol	28d26029eba1faf6e0af9a543a3ad12e649177df4c01df2c4c7f853c3673058e
@openzeppelin/contracts/utils/Strings.sol	cb2df477077a5963ab50a52768cb74ec6f32177177a78611ddbbe2c07e2d36de
@openzeppelin/contracts/utils/Context.sol	1458c260d010a08e4c20a4a517882259a23a4baa0b5bd9add9fb6d6a1549814a
@openzeppelin/contracts/utils/math/SignedMath.sol	420a5a5d8d94611a04b39d6cf5f02492552ed4257ea82aba3c765b1ad52f77f6
@openzeppelin/contracts/utils/math/Math.sol	85a2caf3bd06579fb55236398c1321e15fd524a8fe140dff748c0f73d7a52345
@openzeppelin/contracts/utils/introspection/IERC165.sol	701e025d13ec6be09ae892eb029cd83b3064325801d73654847a5fb11c58b1e5

@openzeppelin/contracts/utils/introspection/ERC165.sol	8806a632d7b656cadb8133ff8f2acae4405b3a64d8709d93b0fa6a216a8a6154
@openzeppelin/contracts/token/ERC20/IERC20.sol	7ebde70853ccafcf1876900dad458f46eb9444d591d39bfc58e952e2582f5587
@openzeppelin/contracts/token/ERC20/ERC20.sol	d20d52b4be98738b8aa52b5bb0f88943f62128969b33d654fbca731539a7fe0a
@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol	af5c8a77965cc82c33b7ff844deb9826166689e55dc037a7f2f790d057811990
@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol	0344809a1044e11ece2401b4f7288f414ea41fa9d1dad24143c84b737c9fc02e
@openzeppelin/contracts/access/Ownable.sol	a8e4e1ae19d9bd3e8b0a6d46577eec098c01fbaffd3ec1252fd20d799e73393b
@openzeppelin/contracts/access/IAccessControl.sol	d03c1257f2094da6c86efa7aa09c1c07ebd33dd31046480c5097bc2542140e45
@openzeppelin/contracts/access/AccessControl.sol	afd98330d27bddff0db7cb8fcf42bd4766dda5f60b40871a3bec6220f9c9edf7

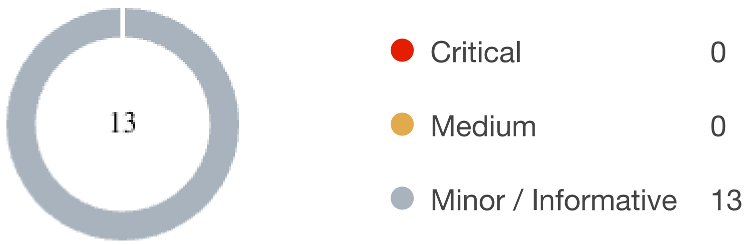
Overview

The Findex Exchange ecosystem consists of a token contract named “TokenContract” and a token vesting contract named “TokenVesting”. The “TokenContract” is an ERC-20 token that extends the OpenZeppelin ERC-20 and ERC-20Burnable contracts, incorporating basic token functionalities with burn capabilities. It also includes an ownership management system through the Ownable contract, allowing the contract owner to blacklist specific addresses and burn tokens associated with blacklisted addresses. Transfer functions are modified to restrict transactions involving blacklisted addresses.

On the other hand, the “TokenVesting” contract is designed for managing token allocations and vesting schedules. It utilizes the OpenZeppelin AccessControl contract to enforce role-based access control. The contract supports various allocation types, such as Ecosystem, Advisors, Marketing, Partners, Presale, Private1, Private2, and Public, each with its own lockup period and vesting schedule. The contract includes functions to set allocations for specific addresses, cancel allocations during a specified cancellation period, and allow recipients to claim their allocated tokens based on predefined vesting rules. Additionally, there are functions to burn tokens allocated for specific purposes, with conditions related to the contract's lockup and vesting periods.

Both contracts aim to provide a robust and flexible framework for managing token-related activities, including transfers, blacklisting, allocations, and vesting schedules. The “TokenVesting” contract, in particular, offers a comprehensive solution for handling different types of token allocations, catering to specific roles and time-based conditions.

Findings Breakdown



Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	13	0	0	0

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	EIS	Excessively Integer Size	Unresolved
●	OO	Operator Optimization	Unresolved
●	PSU	Potential Subtraction Underflow	Unresolved
●	CR	Code Repetition	Unresolved
●	PCI	Percentage Calculation Inconsistency	Unresolved
●	MCM	Misleading Comment Messages	Unresolved
●	MU	Modifiers Usage	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved

EIS - Excessively Integer Size

Criticality	Minor / Informative
Location	TokenVesting.sol#L157,166,175,226,235
Status	Unresolved

Description

The contract is using a bigger unsigned integer data type than the maximum size that is required. By using an unsigned integer data type larger than necessary, the smart contract consumes more storage space and requires additional computational resources for calculations and operations involving these variables. This can result in higher transaction costs, longer execution times, and potential scalability bottlenecks.

For instance, the `october1_2024` variable can be stored in a `log2(1727730000) = 30.68 -> uint32` variable.

```
uint256 october1_2024 = 1727730000;
uint256 september23_2024 = 1727049600;
```

Recommendation

To address the inefficiency associated with using an oversized unsigned integer data type, it is recommended to accurately determine the required size based on the range of values the variable needs to represent.

OO - Operator Optimization

Criticality	Minor / Informative
Location	TokenVesting.sol#L94
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

In the contract, there is a requirement check on certain variables, which are of type uint256. The condition checks if is zero is less than these variables. However, since the variables are unsigned integers (uint256), their value is always greater than or equal to zero by default. Therefore, using the “<” operator can be optimised, by replacing it with the “!=” operator.

```
require(0 < amount_, 'Allocated amount must be greater than 0');
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

PSU - Potential Subtraction Underflow

Criticality	Minor / Informative
Location	TokenVesting.sol#L106
Status	Unresolved

Description

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

```
_allocationTypes[uint256(allocationType_)].availableAmount -= amount_;
```

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.

CR - Code Repetition

Criticality	Minor / Informative
Location	TokenVesting.sol#L154,223
Status	Unresolved

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.

```
uint256 newPercentage = 0;

if (a.allocationType == AllocationType.Ecosystem) {
    uint256 october1_2024 = 1727730000;
    if (block.timestamp >= october1_2024) {
        uint256 periodsAfterOctober = (block.timestamp - october1_2024) /
(10 * MONTH);
        newPercentage = 25 * (periodsAfterOctober + 1);
        if (newPercentage > 100) {
            newPercentage = 100;
        }
    }
} else if (a.allocationType == AllocationType.Marketing) {
...
```

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.

PCI - Percentage Calculation Inconsistency

Criticality	Minor / Informative
Location	TokenVesting.sol#L174,234
Status	Unresolved

Description

The functions `claimTokens` and `canClaimTokens` in the contract display an inconsistency in the calculation of the claimable amount percentage when the allocation type is set to `AllocationType.Public`. In the `claimTokens` function, the percentage is assigned to 100 if the current date is greater than 23 September 2024. However, in the `canClaimTokens` function, the percentage is calculated proportionally based on the number of quarters passed since 23 September 2024. This inconsistency may lead to users receiving conflicting indications of whether they can claim their tokens or not.

```
else if (a.allocationType == AllocationType.Public) {
    uint256 september23_2024 = 1727049600;
    if (block.timestamp >= september23_2024) {
        newPercentage = 100;
    }
}

else if (a.allocationType == AllocationType.Marketing || a.allocationType
== AllocationType.Public) {
    uint256 september23_2024 = 1727049600;
    if (block.timestamp >= september23_2024) {
        uint256 quartersAfterSeptember = (block.timestamp -
september23_2024) / (3 * MONTH);
        newPercentage = 10 + (30 * quartersAfterSeptember);
        if (newPercentage > 100) {
            newPercentage = 100;
        }
    }
}
```

Recommendation

The team is advised to ensure consistency in the percentage calculation for public allocations by aligning the percentage calculation in both the `claimTokens` and `canClaimTokens` functions. By aligning the percentage calculation logic, the team will ensure that users receive consistent information regarding their claiming ability, leading to a more predictable user experience.

MCM - Misleading Comment Messages

Criticality	Minor / Informative
Location	TokenVesting.sol#L112
Status	Unresolved

Description

The contract is using misleading comment messages. These comment messages do not accurately reflect the actual implementation, making it difficult to understand the source code. As a result, the users will not comprehend the source code's actual implementation.

```
/// Sets allocation for the given recipient with corresponding amount.
function burn(AllocationType allocationType_) public { ... }
```

Recommendation

The team is advised to carefully review the comment in order to reflect the actual implementation. To improve code readability, the team should use more specific and descriptive comment messages.

MU - Modifiers Usage

Criticality	Minor / Informative
Location	TokenVesting.sol#L96,120,267,276
Status	Unresolved

Description

The contract is using repetitive statements on some methods to validate some preconditions. In Solidity, the form of preconditions is usually represented by the modifiers. Modifiers allow you to define a piece of code that can be reused across multiple functions within a contract. This can be particularly useful when you have several functions that require the same checks to be performed before executing the logic within the function.

```
_checkRole(_msgSender(), allocationType_);
require(hasRole(DEFAULT_ADMIN_ROLE, _msgSender()), 'Must have admin role
to refund');
```

Recommendation

The team is advised to use modifiers since it is a useful tool for reducing code duplication and improving the readability of smart contracts. By using modifiers to perform these checks, it reduces the amount of code that is needed to write, which can make the smart contract more efficient and easier to maintain.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	TokenContract.sol#L29
Status	Unresolved

Description

The contract modifies the state of the following variables without checking if their current value is the same as the one given as an argument. As a result, the contract performs redundant storage writes, when the provided parameter matches the current state of the variables, leading to unnecessary gas consumption and inefficiencies in contract execution.

```
blacklist[_address] = blacklisted;
```

Recommendation

The team is advised to implement additional checks within to prevent redundant storage writes when the provided argument matches the current state of the variables. By incorporating statements to compare the new values with the existing values before proceeding with any state modification, the contract can avoid unnecessary storage operations, thereby optimizing gas usage.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	TokenContract.sol#L29TokenVesting.sol#L271,277
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
blacklist[_address] = blacklisted;
require(erc20.transfer(recipientAddress_, balance), 'Cannot transfer
tokens');
recipientAddress_.transfer(address(this).balance);
```

Recommendation

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

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	TokenContract.sol#L28,32,36
Status	Unresolved

Description

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

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

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

```
address _address
```

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	TokenVesting.sol#L168,169,190,191,237,238,251,252
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.

```
uint256 quartersAfterSeptember = (block.timestamp - september23_2024) / (3
* MONTH)
newPercentage = 10 + (30 * quartersAfterSeptember)
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	TokenVesting.sol#L277
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.

```
recipientAddress_.transfer(address(this).balance)
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	TokenVesting.sol#L2TokenContract.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 ^0.8.19;
```

Recommendation

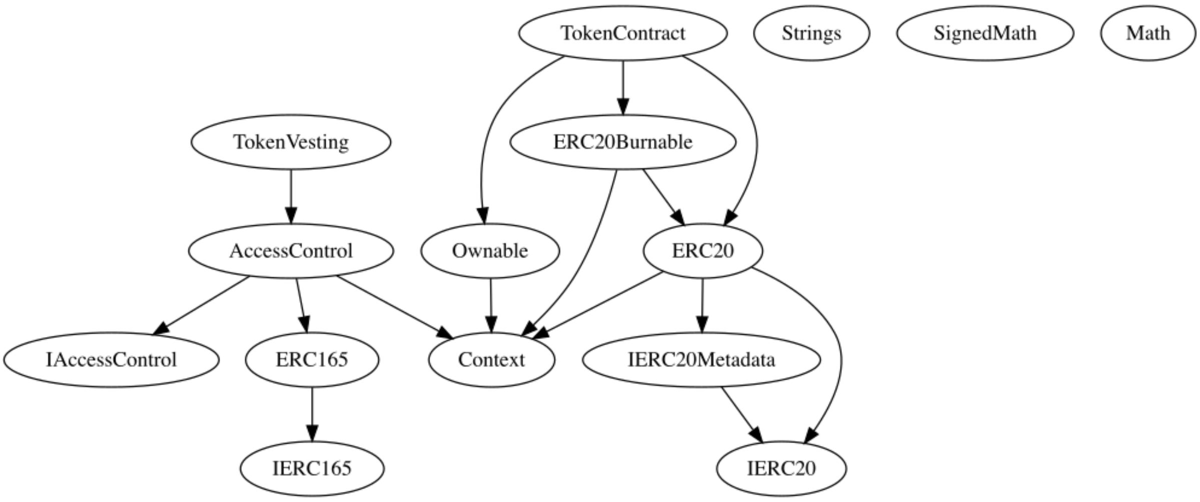
The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest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

Functions Analysis

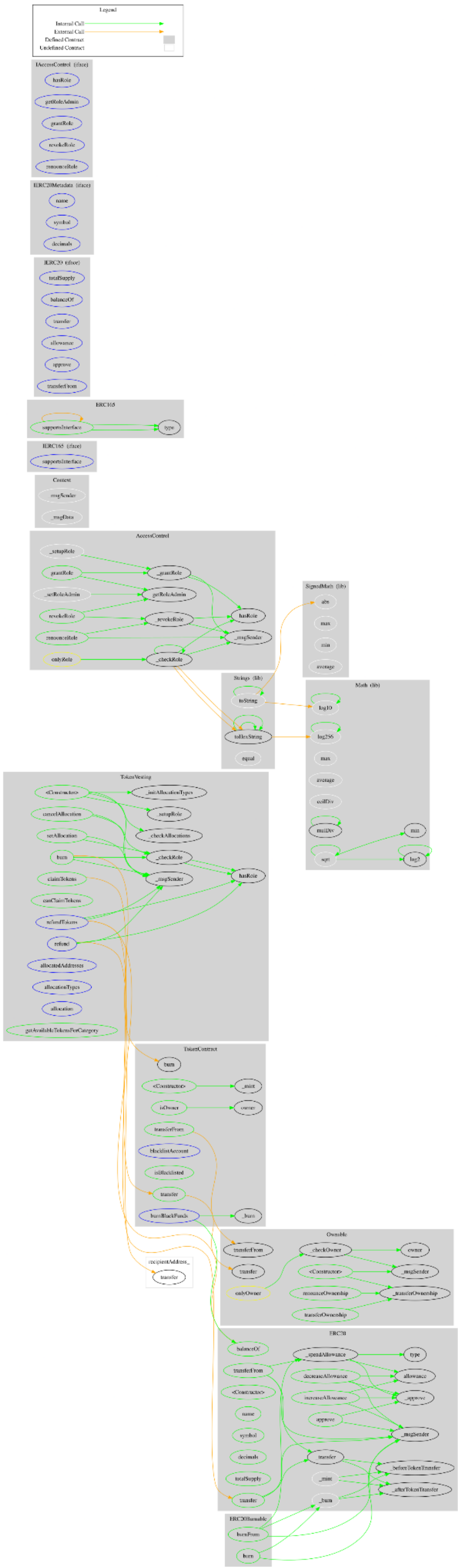
Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
TokenVesting	Implementation	AccessControl		
		Public	✓	-
	setAllocation	Public	✓	-
	burn	Public	✓	-
	cancelAllocation	Public	✓	-
	claimTokens	Public	✓	-
	canClaimTokens	Public		-
	refundTokens	External	✓	-
	refund	External	Payable	-
	allocatedAddresses	External		-
	allocationTypes	External		-
	allocation	External		-
	_initAllocationTypes	Private	✓	
	_checkAllocations	Private		
	_checkRole	Private		
	getAvailableTokensForCategory	Public		-

TokenContract	Implementation	ERC20, ERC20Burnable, Ownable		
		Public	✓	ERC20
	isOwner	Public		-
	blacklistAccount	External	✓	onlyOwner
	isBlacklisted	Public		-
	burnBlackFunds	External	✓	onlyOwner
	transfer	Public	✓	-
	transferFrom	Public	✓	-

Inheritance Graph



Flow Graph



Summary

Findex Exchange contract implements a token and vesting mechanism. This audit investigates security issues, business logic concerns and potential improvements.

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