Cyberscope

Audit Report

Galaxy Fox

May 2024

Repository https://github.com/humanshield89/galaxy-fox-token

Commit f0fc44d6a19a2cdfffc2fc86b696389ad0b88fca

Audited by © cyberscope

Table of Contents

Review

Repository	https://github.com/humanshield89/galaxy-fox-token
Commit	f0fc44d6a19a2cdfffc2fc86b696389ad0b88fca

Audit Updates

Initial Audit	13 May 2024

Source Files

Filename	SHA256
GfoxClaim.sol	e2c774b1f5c68d003f0e678c785b62f2153148c9c8ea47365ef53157f80493b2

Overview

The GfoxClaim contract is designed to manage the claiming process of tokens in a staggered manner, integrating with a staking mechanism for a token termed as `gfoxToken`. Built upon blockchain, this contract uses a merkle proof verification system to ensure that claims are legitimate and in accordance with pre-determined allocations. The contract's governance is centralized under an owner, typical in admin-controlled smart contracts, providing specific controls over the operational parameters.

Claim Functionality

The `claim` function allows users to claim their allocated tokens based on a merkle proof that verifies their entitlement. This function checks if the claim has started and ensures the claim is not yet fully redeemed. The claimable amount is calculated based on the time elapsed since the start of the claiming period, allowing a portion of the total allocation to be claimed incrementally over time. Tokens are then transferred directly to the user's wallet.

Claim and Stake Functionality

The `claimAndStake` function extends the basic claim functionality by not only allowing users to claim their tokens but also directly staking these tokens on behalf of the user into a specified pool. This is facilitated through an external staking contract. This feature adds a layer of convenience, enabling users to participate in staking immediately upon claiming without requiring a separate transaction.

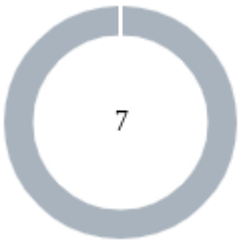
Owner Functionalities

The owner of the contract has several administrative capabilities. They can set or update the merkle root used for claim verification, which is crucial when updating the set of claims. The owner can also initiate the claiming process by setting the start time. Additionally, the `recoverTokens` function allows the owner to retrieve any type of ERC20 tokens of the contract, ensuring asset safety and administrative control.

Reward Distribution Mechanics

The reward distribution is designed to allow an initial claim of a certain percentage at the Token Generation Event (TGE), followed by additional claims that can be made at regular intervals (every week). The distribution mechanics are programmed to allow the full amount to be claimable progressively over a series of weeks, incentivizing long-term engagement from token holders. This phased approach ensures a balanced release of tokens into circulation, aligning with the contract's aim to manage the token economy effectively.

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	7

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	7	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	CR	Code Repetition	Unresolved
	CCR	Contract Centralization Risk	Unresolved
	MPC	Merkle Proof Centralization	Unresolved
	TSI	Tokens Sufficiency Insurance	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L07	Missing Events Arithmetic	Unresolved
	L19	Stable Compiler Version	Unresolved

CR - Code Repetition

Criticality	Minor / Informative
Location	GfoxClaim.sol#L62,80
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.

Specifically the `claim` and `claimAndStake` functions share similar code segments.

```
    function claim(
        address _account,
        uint256 _amount,
        bytes32[] calldata _proof
    ) external claimStarted {
        _verify(_account, _amount, _proof);

        uint256 amountToClaim = _getClaimableAmount(_amount);

        amountToClaim -= claimedAmount[_account];

        claimedAmount[_account] += amountToClaim;

        emit Claimed(_account, amountToClaim, _amount);

        gfoxToken.safeTransfer(_account, amountToClaim);
    }

    function claimAndStake(
        address _account,
        uint256 _amount,
        bytes32[] calldata _proof,
        uint256 _poolId
    ) external claimStarted {
        _verify(_account, _amount, _proof);

        uint256 amountToClaim = _getClaimableAmount(_amount);

        amountToClaim -= claimedAmount[_account];

        claimedAmount[_account] += amountToClaim;

        emit Claimed(_account, amountToClaim, _amount);

        gfStaking.stakeFor(_account, _poolId, amountToClaim);
    }
```

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.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	GfoxClaim.sol#L99,111
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 owner has the authority to manage critical aspects of the GfoxClaim contract's operation. Through the `recoverTokens` function, the owner can claim any token balance from the contract, transferring it to any address, thereby controlling the disbursement of assets held by the contract. Additionally, the `setClaimStart` function grants the owner the exclusive right to set the initial claim start date. This ability allows the owner to define the commencement of the claim period, ensuring that it aligns with the project's timeline and objectives, while also preventing any changes once the date is established, thereby upholding the integrity and predictability of the claim distribution schedule.

```
    function recoverTokens(
        address _token,
        address _to,
        uint256 _amount
    ) external onlyOwner {
        IERC20(_token).safeTransfer(_to, _amount);
    }

    function setClaimStart(uint256 _claimStart) external onlyOwner {
        // require not already set
        require(claimStart == 0, "GfoxClaim: Already set");
        claimStart = _claimStart;
    }
```

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.

MPC - Merkle Proof Centralization

Criticality	Minor / Informative
Location	GfoxClaim.sol#L107
Status	Unresolved

Description

The contract uses a Merkle Proof mechanism in order to define many applicable addresses. The verification process is based on an off-chain configuration. The contract owner is responsible for updating the in-chain “Merkle Root” in order to validate correctly the provided message.

```
    function setMerkleRoot(bytes32 _merkleRoot) external
onlyOwner {
        merkleRoot = _merkleRoot;
    }
```

Recommendation

We state that the Merkle Proof algorithm is required for proper protocol operations and gas consumption decrease. Thus, we emphasize that the Merkle proof algorithm is based on an off-chain mechanism. Any off-chain mechanism could potentially be compromised and affect the on-chain state unexpectedly. The team should carefully manage the private keys of the 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.

TSI - Tokens Sufficiency Insurance

Criticality	Minor / Informative
Location	GfoxClaim.sol#L42,77,96
Status	Unresolved

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.

```
gfoxToken = IERC20(_gfoxToken);
...
gfoxToken.safeTransfer(_account, amountToClaim);
...
gfStaking.stakeFor(_account, _poolId, amountToClaim);
```

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 presale 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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	GfoxClaim.sol#L63,64,65,81,82,83,84,100,101,102,107,111
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 _account
uint256 _amount
bytes32[] calldata _proof
uint256 _poolId
address _token
address _to
bytes32 _merkleRoot
uint256 _claimStart
```

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	GfoxClaim.sol#L114
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.

```
claimStart = _claimStart
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	GfoxClaim.sol#L3
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 ^0.8.23;
```

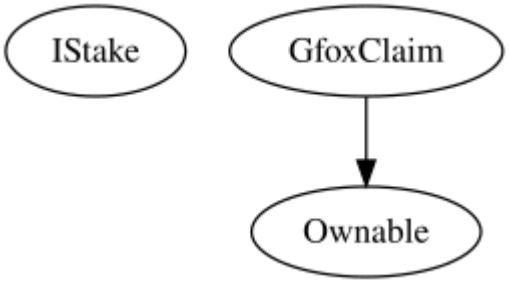
Recommendation

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.

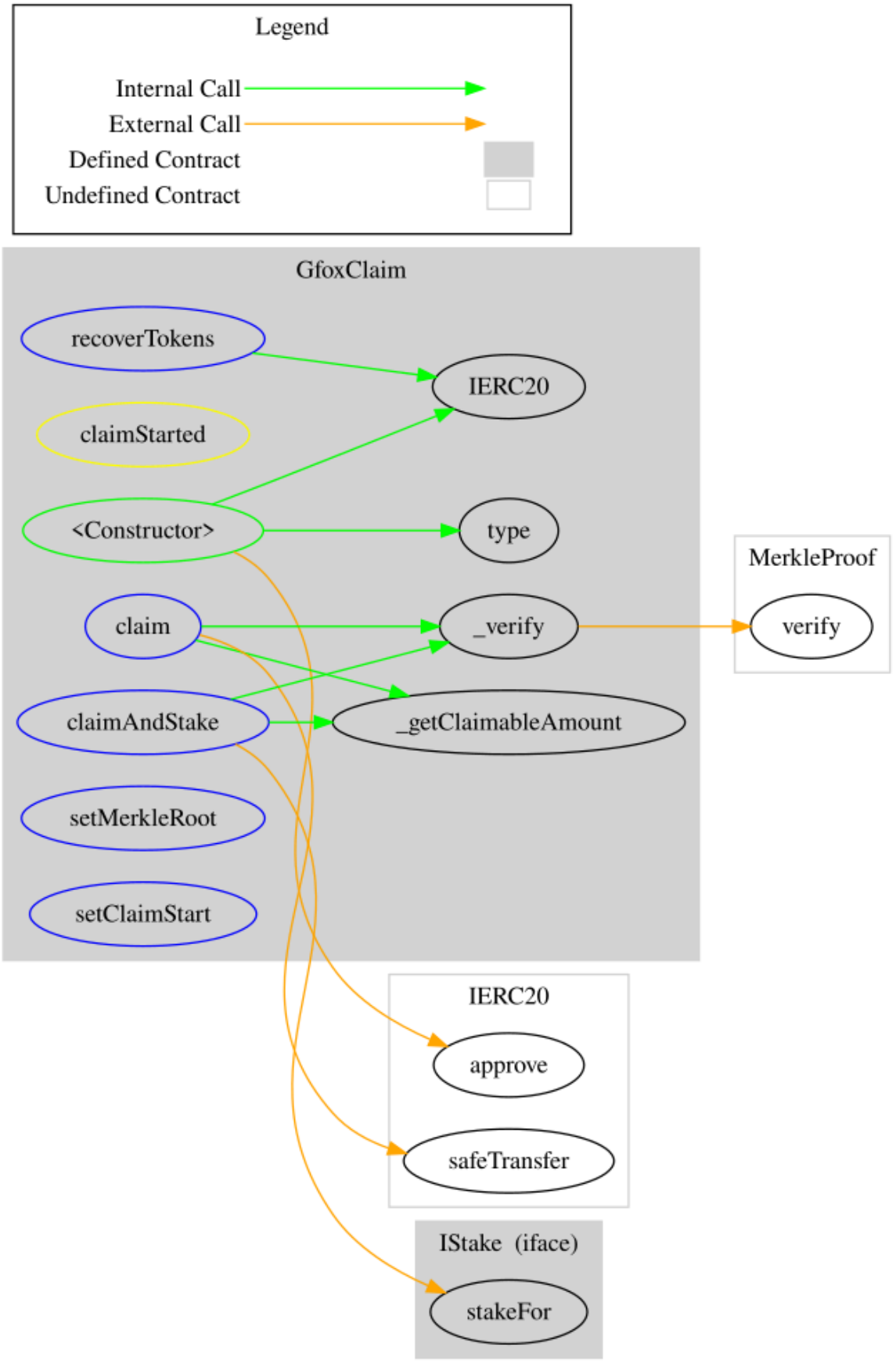
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IStake	Interface			
	stakeFor	External	✓	-
GfoxClaim	Implementation	Ownable		
		Public	✓	Ownable
	claim	External	✓	claimStarted
	claimAndStake	External	✓	claimStarted
	recoverTokens	External	✓	onlyOwner
	setMerkleRoot	External	✓	onlyOwner
	setClaimStart	External	✓	onlyOwner
	_verify	Internal		
	_getClaimableAmount	Internal		

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
GfoxClaim
recoverTokens
IERC20
claimStarted
<Constructor>
type
claim
_verify
claimAndStake
_getClaimableAmount
setMerkleRoot
setClaimStart
MerkleProof
verify
IERC20
approve
safeTransfer
IStake (iface)
stakeFor

Summary

The Galaxy Fox contract implements a token claim mechanism with an emphasis on secure, verifiable, and staggered token distribution. This audit investigates security issues, business logic concerns, and potential improvements, focusing on the implementation of Merkle proof verification for claims, and the integration with an external staking contract. The contract also features owner specific functionalities that allow manipulation of critical settings and recovery of tokens, ensuring both flexibility and control over the contract's operations.

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