Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

2 Audit Methodology

3 Project Overview

3.1 Project Introduction

3.2 Vulnerability Information

4 Code Overview

4.1 Contracts Description

4.2 Visibility Description

4.3 Vulnerability Summary

5 Audit Result

6 Statement

1 Executive Summary

On 2024.04.30, the SlowMist security team received the Galxe team's security audit application for Galxe Staking, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box lead, black, grey box assists" to conduct a complete security test on the project in the way closest to the real attack.

The test method information:

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project team should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.
Suggestion	There are better practices for coding or architecture.

2 Audit Methodology

The security audit process of SlowMist security team for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

Serial Number	Audit Class	Audit Subclass
1	Overflow Audit	-
2	Reentrancy Attack Audit	-
3	Replay Attack Audit	-
4	Flashloan Attack Audit	-
5	Race Conditions Audit	Reordering Attack Audit
6	Permission Vulnerability Audit	Access Control Audit
		Excessive Authority Audit
7	Security Design Audit	External Module Safe Use Audit
		Compiler Version Security Audit
		Hard-coded Address Security Audit
		Fallback Function Safe Use Audit
		Show Coding Security Audit
		Function Return Value Security Audit
		External Call Function Security Audit

Serial Number	Audit Class	Audit Subclass
7	Security Design Audit	Block data Dependence Security Audit
		tx.origin Authentication Security Audit
8	Denial of Service Audit	-
9	Gas Optimization Audit	-
10	Design Logic Audit	-
11	Variable Coverage Vulnerability Audit	-
12	"False Top-up" Vulnerability Audit	-
13	Scoping and Declarations Audit	-
14	Malicious Event Log Audit	-
15	Arithmetic Accuracy Deviation Audit	-
16	Uninitialized Storage Pointer Audit	-

3 Project Overview

3.1 Project Introduction

The project for this audit was a staking contract for Galxe. Users can stake specified tokens and make withdrawals only after the lock-up time. The owner role can directly modify the address and multiplier of tokens that need to be staked, or can upgrade tokens directly through an external upgrade contract.

3.2 Vulnerability Information

The following is the status of the vulnerabilities found in this audit:

NO	Title	Category	Level	Status
N1	Rounding issues in withdrawals	Arithmetic Accuracy Deviation Vulnerability	Low	Acknowledged
N2	Risk of excessive authority	Authority Control Vulnerability Audit	Medium	Fixed

4 Code Overview

4.1 Contracts Description

Audit Version:

https://github.com/Galxe/staking-contract

commit: 915c2f9b56cd6fb9c8b236917f58c8556128ebee

The main network address of the contract is as follows:

https://etherscan.io/address/0x7bbe09e066077b3888432eedec958f64b2e47239

https://bscscan.com/address/0x7bBE09E066077b3888432EedEC958F64B2E47239

4.2 Visibility Description

The SlowMist Security team analyzed the visibility of major contracts during the audit, the result as follows:

GalxeStaking			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	onlyOwner
upgradeToken	External	Can Modify State	onlyOwner
changeTokenAndMultiplier	External	Can Modify State	onlyOwner
pause	Public	Can Modify State	onlyOwner

GalxeStaking			
unpause	Public	Can Modify State	onlyOwner
setLockDuration	Public	Can Modify State	onlyOwner
getUnStakeInfo	External	-	-
stakeToken	Public	Can Modify State	whenNotPaused
unStake	Public	Can Modify State	whenNotPaused
revokeUnStake	Public	Can Modify State	whenNotPaused
withdrawUnLockedToken	Public	Can Modify State	whenNotPaused
withdrawAllUnLockedToken	Public	Can Modify State	whenNotPaused
getStakeAmount	External	-	-
_transferToken	Private	Can Modify State	-
recoverERC20	External	Can Modify State	onlyOwner
_removeAtIndex	Private	Can Modify State	-
_removeBeforeIndex	Private	Can Modify State	-

4.3 Vulnerability Summary

[N1] [Low] Rounding issues in withdrawals

Category: Arithmetic Accuracy Deviation Vulnerability

Content

In the GalxeStaking contract, the user can call the unStake function to submit a request to release the stake and withdraw tokens from their stakes by calling the withdrawUnLockedToken and withdrawAllUnLockedToken functions.

However, there is a situation where if the value of the amount parameter passed in by the user when calling the unStake function is less than the multiplier value, the withdrawal will be rounded to 0, and there is no checking, which will result in the user's balance being deducted as normal, but no tokens will be awarded.

Code Location:

contracts/GalxeStaking.sol

```
    function unStake(uint256 amount) public whenNotPaused {
        ...

        // sub balances
        balances[msg.sender] = balances[msg.sender] - amount;
        unStakeInfoMap[msg.sender].push(unStakeInfo);

        unStakeInfo.amount /= multiplier;
        emit TokenUnStaked(msg.sender, unStakeInfo);
    }

    ...

    function withdrawUnLockedToken(uint256 index) public whenNotPaused {
        ...

        if (block.timestamp >= unStakeInfoMap[msg.sender][index].unStakeTime +
lockTimePeriod) {
            UnStakeInfo memory tmpInfo = _removeAtIndex(index);
            tmpInfo.amount = tmpInfo.amount / multiplier;
            _transferToken(address(this), msg.sender, tmpInfo.amount);
            emit TokenWithdraw(msg.sender, tmpInfo);
        } else {
            revert("Tokens are only available after correct time period has elapsed");
        }
    }

    ...
    function withdrawAllUnLockedToken() public whenNotPaused {
        ...

        if (amount > 0) {
            _removeBeforeIndex(index);
            _transferToken(address(this), msg.sender, amount / multiplier);
        } else {
            revert("Tokens are only available after correct time period has elapsed");
        }
    }
```

Solution

It is recommended to verify in the unStake function that the number of withdrawals is greater than the value of

multiplier and to check that the number of tokens transferred cannot be 0 at the time of withdrawal.

Status

Acknowledged; Project team response: Since the project is already online, it will be restricted on the front end.

[N2] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

In the GalxeStaking contract, The owner role can arbitrarily change the allowedToken and multiplier values by calling the upgradeToken and changeTokenAndMultiplier functions. If the privilege is lost or misused, there may be an impact on the user's funds.(Examples include changing the allowedToken to a worthless token, or changing the value of multiplier to an unusually large value so that the user can't get back the tokens in the stakes.)

Code Location:

contracts/GalxeStaking.sol

```
    function upgradeToken(IERC20 _newToken, ITokenUpgrade upgrade, uint256
_multiplier) external onlyOwner {
        require(_multiplier > 0, "Multiplier must be greater than 0");
        uint256 amount = allowedToken.balanceOf(address(this)) * multiplier;
        require(_newToken.approve(address(upgrade), amount), "Approve token cannot
fail");
        require(upgrade.upgradeToken(amount), "Upgrade token cannot fail");
        allowedToken = _newToken;
        multiplier = _multiplier;
        emit TokenUpgraded(_newToken, upgrade, amount, _multiplier);
    }

    // only change token and multiplier, token upgrade should be done somehow
manually.
    function changeTokenAndMultiplier(IERC20 _token, uint256 _multiplier) external
onlyOwner {
        require(_multiplier > 0, "Multiplier must be greater than 0");
        allowedToken = _token;
        multiplier = _multiplier;
        emit TokenAndMultiplierChanged(_token, _multiplier);
    }
```

Solution

In the short term, transferring owner ownership to multisig contracts is an effective solution to avoid single-point risk. But in the long run, it is a more reasonable solution to implement a privilege separation strategy and set up multiple privileged roles to manage each privileged function separately. The authority involving user funds should be managed by the community, and the authority involving emergency contract suspension can be managed by the EOA address. This ensures both a quick response to threats and the safety of user funds.

Status

Fixed; the tx of the owner permission transfer:

https://etherscan.io/tx/0x4d872fe2700ea6a3d69c262b79c3cd7feb678f0584ce54d00dc3c2b6dae8b2e5

https://bscscan.com/tx/0xf13f809a19a94b7728bdcc94446ad9e0bd7e19ff20b2dbd03ec3128a94c7ef58

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002404300005	SlowMist Security Team	2024.04.30 - 2024.04.30	Low Risk

Summary conclusion: The SlowMist security team uses a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 2 medium risk vulnerabilities. 1 medium risk vulnerability is acknowledged, another finding was fixed. The code has been deployed to the mainnet. Permission for the owner role has been moved to multi-signature contract.

6 Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist