GameSwift Launchpool Security Review

Duration: March 28, 2025 - April 1, 2025

April 30, 2025

Conducted by **KeySecurity**
Georgi Krastenov, Lead Security Researcher

Table of Contents

1 About KeySecurity

KeySecurity is an innovative Web3 security company that hires top talented security researchers for your project. We have conducted over 40 security reviews for different projects which hold over $500,000,000 in TVL. For security audit inquiries, you can reach out to us on Twitter/X or Telegram @gkrastenov, or check our previous work `here`.

2 About GameSwift

GameSwift is a Layer 2 modular blockchain optimized for gaming, powered by the $GSWIFT gas token. Using AI and computing power to drive the mass adoption of Web3 gaming.

3 Disclaimer

Audits are a time, resource, and expertise bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to identify as many vulnerabilities as possible. Audits can show the presence of vulnerabilities **but not their absence**.

4 Risk classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4.1 Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost or a functionality of the protocol is affected.
- **Low** - any kind of unexpected behaviour that's not so critical.

4.2 Likelihood

- **High** - direct attack vector; the cost is relatively low to the amount of funds that can be lost.
- **Medium** - only conditionally incentivized attack vector, but still relatively likely.
- **Low** - too many or too unlikely assumptions; provides little or no incentive.

4.3 Actions required by severity level

- **Critical** - client **must** fix the issue.
- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.

5 Executive summary

Overview

Project Name	GameSwift
Repository	https://github.com/GameSwift/launchpool/tree/feat/new-impelementation
Commit hash	370ea5b7102e0ad045fcdab52c7f58520aede2eb
Review Commit hash	cb0a90aea78a6d9b05ee48b2a25f62aaa5cac11d
Documentation	N/A
Methods	Manual review

Scope

FixedStaking.sol
FixedStakingStToken.sol
FixedStakingStakedToken.sol
Launchpool.sol
sToken.sol
sTokenPublicMintable.sol

Timeline

March 28, 2025	Audit kick-off
April 1, 2025	Mitigation review
April 1, 2025	Final report

Issues Found

Severity	**Count**
High	0
Medium	0
Low	0
Information	2
Code Improvement	3
Total	**5**

6 Findings

6.1 Information

6.1.1 The stakeAs function can only be called by the owner

Severity: *Information*

Context: FixedStaking.sol#L87

Description: The stakeAs function is expected to be called by every user who wants to stake their tokens to another user or to their other wallet. Currently, the function has an onlyOwner modifier, which means it can only be called by the contract owner.

Recommendation: Remove the `onlyOwner` modifier.

Resolution and Client comment: Resolved.

6.1.2 Unnecessary override of the _restake function.

Severity: *Information*

Context: FixedStakingStToken.sol#L79

Description: The `FixedStakingStToken` contract inherits the `FixedStaking` contract. In the child contract, the main change is that the staked tokens are directly minted to the user when they `stake` or `restake` and burned when they `unstake` their tokens.

The external `restake` function is also overridden from the parent class, where new logic is added. The overridden function does not call the internal `_restake` function, which means that the new overridden `_restake` function will never be called from the child class.

```solidity
    function restake() external override nonReentrant {
        UserInfo storage user = userInfo[msg.sender];

        uint256 pending = calculateReward(msg.sender) + user.waitingRewards;
        require(pending != 0, "restake: nothing to restake");

        user.waitingRewards = 0;

        user.amount += pending;
        totalStaked += pending;

        user.lastStakeTime = block.timestamp;

        sToken.mintTo(msg.sender, pending);
        emit Restaked(msg.sender, pending);
    }
```

Recommendation: Remove the redundant`_restake` function from the `FixedStakingStToken` contract.

Resolution and Client comment: Resolved.

6.2 Code Improvement

6.2.1 Remove payable modifier

Severity: *Code Improvement*

Context: Launchpool.sol#L142

Description: The payable modifier should be used only when it is expected that msg.sender will send native tokens to the contract through the calling of a function. The function `unstake` in `Launchpool` contract have a payable modifier, but the caller is not expected to send native tokens.

Recommendation: Remove the payable modifier.

Resolution and Client comment: Resolved.

6.2.2 Redundant error

Severity: *Code Improvement*

Context: Launchpool.sol#L8

Description: The error `BeforeReleaseTime` is redundant in the ILaunchpool interface.

Recommendation: Remove the redundant error.

Resolution and Client comment: Resolved.

6.2.3 Use custom error instead of require statement

Severity: *Code Improvement*

Context: Global

Description: Use custom error instead of a require statement in the `FixedStaking` and `FixedStakingStToken` contracts.

Recommendation: Use custom error.

Resolution and Client comment: Acknowledged.