IceCreamSwap

A Multi-Chain DeFi Ecosystem

IceCreamSwap Security Scan Results

by Pessimistic

This is not a security audit

This report is public

May 27, 2024

Abstract

This report considers the security of smart contracts of the IceCreamSwap protocol. Our task is to find and describe security issues using the static-analysis tools Slither and Slitherin and help resolve them.

The work is financially covered by the Arbitrum Foundation grant.

Disclaimer

Current work does not give any warranties on the security of the code. It is not an audit or its replacement. Performing this scan, we focused on finding as many crucial issues as possible rather than making sure that the protocol was entirely secure. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

Summary

In this report, we described issues found in smart contracts of the IceCreamSwap protocol.

We scanned the codebase and manually rejected or verified all automated findings, revealing eight relevant issues.

The developers acknowledged seven issues and commented on one.

The entire process is described in the section below.

Scan process

Under the Arbitrum Foundation grant, we researched and developed Arbitrum-specific detectors. They became publicly available with Slitherin v0.6.0 release.

Workflow

This work consisted of five stages:

1. For the scan, we were provided with the IceCreamSwap project on the following GitHub repositories:
    - IceCreamSwap-smart-contracts, commit: 62dc6989007ee3be1b6f8b4918bd0387aac61e91, the scope of the scan includes **Soulbound** and **Staked-Core** projects;
    - icecreamswap-v3-contracts, commit: e672cbd417d2e8161e87db0f6c1b89a1e576bbaa.
2. For the analysis of the protocol, we launched Slither v0.10.1 and Slitherin v0.6.0 on the provided codebase.
3. One auditor manually checked (rejected or accepted) all findings reported by the tools. The second auditor verified this work. We shared all relevant issues with the protocol developers and answered their questions.
4. The developers reviewed the findings, acknowledged seven of them, and gave comments on issues they do not intend to fix.
5. We prepared this final report summarizing all the issues and comments from the developers.

Issue categories

Within the confines of this work, we were looking for:

- Arbitrum-specific problems;
- Standard vulnerabilities like re-entrancy, overflow, arbitrary calls, etc;
- Non-compliance with popular standards like ERC20 and ERC721;
- Some access control problems;
- Integration issues with some popular DeFi protocols;
- A wide range of code quality and gas efficiency improvement opportunities.

This scan does not guarantee that these issues are not present in the codebase.

Scan results

Issue category	Number of detectors	Status
Compilation	1	Passed
Arbitrum Integration	3	1 issue found
AAVE Integration	1	Passed
Uniswap V2 Integration	7	Passed
OpenZeppelin	2	Passed
ERC-20	7	Passed
ERC-721	2	Passed
Known Bugs	15	2 issues found
Access Control	3	Passed
Arbitrary Call	5	Passed
Re-entrancy	6	Passed
Weak PRNG	2	Passed
Upgradability	2	Passed
Ether Handling	3	Passed
Low-level Calls	2	Passed
Assembly	2	Passed
Inheritance	3	Passed
Arithmetic	2	Passed
Old Solidity Versions Bugs	10	Passed
Code Quality	15	2 issues found
Best Practices	4	1 issue found
Gas	7	2 issues found

Discovered Issues

Lack of returned value check

Several contracts do not check the returned value from the `pool.swap` call:

- In the **MixedRouteQuoterV1** contract at line 149 in the `quoteExactInputSingleV3` function;
- In the **Quoter** contract at line 81 in `quoteExactInputSingle` function;
- In the **Quoter** contract at line 127 in the `quoteExactOutputSingle` function;
- In the **QuoterV2** contract at line 128 in the `quoteExactInputSingle` function;
- In the **QuoterV2** contract at line 204 in the `quoteExactOutputSingle` function.

Lack of verification of these values can be potentially vulnerable to pool manipulation attacks, where a swap is performed before and after a transaction. We recommend checking returned values to avoid such attacks.

> *Comment from the developers: The **Quoter** contracts are not used on-chain, but rather as a read-only method to query important values for the UI*

Arbitrum block.timestamp behavior

The `getBlockStartingAndCurrentTick` function of the **OracleSlippage** contract at line 34 uses the `block.timestamp` value inside the code of the Arbitrum contract. It behaves differently than on Ethereum, allowing consecutive blocks to have the same `block.timestamp`.

For further details check block-numbers-and-time. We recommend verifying that contract logic is intact for these differences.

The developers decided to take a deep look into Arbitrum block specifications.

Public functions could be turned into external

There are several contracts where functions can be declared as `external` instead of `public`:

- The `batchValidate` function of the **TokenValidator** contract;
- The `getStableAmountsIn`, `getAmountOut`, `getAmountsIn`, and `getPool` functions of the **SmartRouterHelper** contract;
- The `quoteExactInput` function of the **MixedRouteQuoterV1** contract;
- The `recalculateTotalStaked` function of the **StakedCore** contract.

Consider declaring functions as `external` instead of `public` when possible to improve code readability and optimize gas consumption.

The developers decided to fix this issue in future versions of the code.

Signature recover

The `recoverSigner` function of the **KycedContractMinter** contract does not check if the `ecrecover` function returns zero address. If the ownership of the contract is renounced, it may result in the `require` check in the `checkSignature` function being valid in case of a wrong signature. We recommend using the **ECDSA** library from OpenZeppelin for recovering and managing Ethereum account ECDSA signatures.

The developers decided to fix this issue in future versions of the code.

Returned value not checked

In the `delegate` function of the **KycedContractMinter** contract, the returned value is not checked after interactions with token contracts. According to the ERC20 token standard :

> *Callers MUST handle `false` from returns `(bool success)`. Callers MUST NOT assume that `false` is never returned!*

We recommend using the `safeTransfer` function from the OpenZeppelin **SafeERC20** library to transfer ERC20 tokens.

The developers decided to fix this issue in future versions of the code.

Missing event

The several owner setter functions do not emit an event:

- The `setFees` function of the **KycedContractMinter** contract;
- The `updatePerformanceFeecontract` function of the **StakedCore** contract.

Emitting of events in setter functions allows to notify the contract owner and relevant parties about important state changes within the contract.

The developers decided to fix this issue in future versions of the code.

Immutable variable

The `kycedContract` variable in the **KycedContractMinter** is set during contract deployment and never changed later. We recommend declaring it as `immutable` to reduce gas consumption.

The developers decided to fix this issue in future versions of the code.

Redundant check

The `updatePerformanceFee` function of the **StakedCore** contract contains redundant check in the `require` statement. The `0 <= performanceFee` check is redundant for variables of the `uint256` type.

The developers decided to fix this issue in future versions of the code.

This analysis was performed by Pessimistic:

Oleg Bobrov, Security Engineer
Rasul Yunusov, Security Engineer
Egor Dergunov, Junior Security Engineer
Nikita Kirillov, Product Owner
Evgeny Marchenko, CTO
Konstantin Zherebtsov, Business Development Lead

May 27, 2024