Slingshot Finance contest Findings & Analysis Report

2021-11-17

Table of contents

Overview

About C4

Code 432n4 (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

A C4 code contest is an event in which community participants, referred to as Wardens, review, audit, or analyze smart contract logic in exchange for a bounty provided by sponsoring projects.

During the code contest outlined in this document, C4 conducted an analysis of Slingshot Finance smart contract system written in Solidity. The code contest took place between October 30—November 1 2021.

Wardens

18 Wardens contributed reports to the Slingshot Finance code contest:

1. WatchPug
2. daejunpark
3. gpersoon
4. hickuphh3
5. kenzo
6. pmerkleplant
7. pauliax
8. ye0lde
9. cmichel
10. TomFrench
11. csanuragjain
12. pants
13. onewayfunction
14. zer0dot
15. defsec
16. 0x0x0x
17. elprofesor

18. gzeon

This contest was judged by Alberto Cuesta Cañada.

Final report assembled by itsmetechjay and CloudEllie.

Summary

The C4 analysis yielded an aggregated total of 14 unique vulnerabilities and 55 total findings. All of the issues presented here are linked back to their original finding.

Of these vulnerabilities, 0 received a risk rating in the category of HIGH severity, 2 received a risk rating in the category of MEDIUM severity, and 12 received a risk rating in the category of LOW severity.

C4 analysis also identified 17 non-critical recommendations and 24 gas optimizations.

Scope

The code under review can be found within the C4 Slingshot Finance contest repository, and is composed of 16 smart contracts written in the Solidity programming language and includes 649 lines of Solidity code and 0 lines of JavaScript.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities according to a methodology based on OWASP standards.

Vulnerabilities are divided into three primary risk categories: high, medium, and low.

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling
- Escalation of privileges
- Arithmetic

- Gas use

Further information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on the C4 website.

Medium Risk Findings (2)

[M-01] `initialBalance` for native token is wrong

Submitted by WatchPug, also found by daejunpark, gpersoon, hickuphh3, kenzo, and pmerkleplant.

https://github.com/code-423n4/2021-10-slingshot/blob/9c0432cca2e43731d5a0ae9c151dacf7835b8719/contracts/Slingshot.sol#L65-L92

```
    function executeTrades(
        address fromToken,
        address toToken,
        uint256 fromAmount,
        TradeFormat[] calldata trades,
        uint256 finalAmountMin,
        address depricated
    ) external nonReentrant payable {
        depricated;
        require(finalAmountMin > 0, "Slingshot: finalAmountMin cannc
        require(trades.length > 0, "Slingshot: trades cannot be empt
        for(uint256 i = 0; i < trades.length; i++) {
            // Checks to make sure that module exists and is correct
            require(moduleRegistry.isModule(trades[i].moduleAddress)
        }

        uint256 initialBalance = _getTokenBalance(toToken);
        _transferFromOrWrap(fromToken, _msgSender(), fromAmount);

        executioner.executeTrades(trades);

        uint finalBalance;
        if (toToken == nativeToken) {
            finalBalance = _getTokenBalance(address(wrappedNativeTok
        } else {
            finalBalance = _getTokenBalance(toToken);
        }
        uint finalOutputAmount = finalBalance - initialBalance;
        ...
```

https://github.com/code-423n4/2021-10-slingshot/blob/9c0432cca2e43731d5a0ae9c151dacf7835b8719/contracts/Slingshot.sol#L157-L163

```
    function _getTokenBalance(address token) internal view returns (
        if (token == nativeToken) {
            return address(executioner).balance;
        } else {
            return IERC20(token).balanceOf(address(executioner));
        }
    }
```

When users swap to native token (ETH), the `initialBalance` should use the balance of `wrappedNativeToken` instead of native token balance, because `finalBalance` is the balance of `wrappedNativeToken`.

In the current implementation, when the `toToken` is the native token, `initialBalance` will be the ether balance of `executioner` contract. Therefore, when the ether balance of `executioner` is not 0, `finalOutputAmount` will be wrong.

The attacker can transfer a certain amount of ETH to the `executioner` contract and malfunction the protocol. Causing fund loss to users because `finalOutputAmount` is lower than the actual swapped amount, or DoS due to `finalAmountMin` cant be met.

Proof of Concept

Given:

- The attacker send 0.25 ETH to the `executioner` contract;

- The price of ETH in USDC is: 4000
- Alice swaps 5000 USDC to 1.25 ETH with `finalAmountMin` set to `1 ETH`;
- Alice will get 1 ETH out and lose 0.25 ETH;
- Bob swaps 1000 USDC to 0.25 ETH with `finalAmountMin` set to `1 wei`;
- Bob's transaction fails due to `finalAmountMin` cant being met.

Recommendation

Consider updating `_getTokenBalance()` and return `IERC20(wrappedNativeToken).balanceOf(address(executioner));` when `token == nativeToken`.

[tommyz7 (Slingshot) disagreed with severity](#):

> "Alice swaps 5000 USDC to 1.25 ETH with finalAmountMin set to 1 ETH;" this assumption is wrong because it's based on huge slippage assumption. There is no way a Slingshot transaction accepts 20% slippage so funds loss scenario is incorrect.

> duplicate of #18, medium risk since no user funds are at risk.

[M-02] Trades where toToken is feeOnTransferToken might send user less tokens than finalAmountMin

Submitted by kenzo.

Slingshot's `executeTrades` checks that the trade result amount (to be sent to the user) is bigger than `finalAmountMin`, and *after that* sends the user the amount. But if the token charges fee on transfer, the final transfer to the user will decrease the amount the user is getting, maybe below `finalAmountMin`.

Proof of Concept

Slingshot requires `finalOutputAmount >= finalAmountMin` *before* sending the funds to the user: https://github.com/code-423n4/2021-10-slingshot/blob/main/contracts/Slingshot.sol#L93:#L98 So if the token charges fees on transfer, the user will get less tokens than `finalOutputAmount` . The check of `finalOutputAmount` against `finalAmountMin` is premature.

Tools Used

Manual analysis

Recommended Mitigation Steps

Save the user's (not Executioner's) `toToken` balance in the beginning of `executeTrades` after `_transferFromOrWrap(fromToken, _msgSender(), fromAmount)` , and also in the very end, after `executioner.sendFunds(toToken, _msgSender(), finalOutputAmount)` has been called. The subtraction of user's initial balance from ending balance should be bigger than `finalAmountMin` .
https://github.com/code-423n4/2021-10-slingshot/blob/main/contracts/Slingshot.sol#L65:#L99

[tommyz7 (Slingshot) disagreed with severity]():

> Slingshot concern is to execute the trade as promised and make sure we are sending to the user what has been promised in trade estimation. If the token adds additional taxation on transfer, it is on the user side and users understand and accept this. We have seen this play out on production for the previous version of the contracts and we decided not to make that check. It seems the most practical decision.

> Personally, I think this is non-critical.

[alcueca (judge) commented]():

> The severity for the issue is right. The sponsor should add documentation to the fact that some tokens might not conform to common expectations.

Low Risk Findings (12)

- [[L-01] Unnecessary and risky `payable` annotation in swap() functions]()
  Submitted by daejunpark, also found by pauliax.

- [L-02] Inaccurate comment (rescueTokensFromExecutioner) *Submitted by yeOlde.*
- [L-03] BalancerV2ModuleMatic: Ensure tokenOut is not native token *Submitted by hickuphh3.*
- [L-04] Executioner: Restrict funds receivable to be only from wrapped native token *Submitted by hickuphh3.*
- [L-05] Slingshot: Unnecessary receive() *Submitted by hickuphh3.*
- [L-06] Function documentation incorrect for `ConcatStrings::appendUint` *Submitted by pmerkleplant.*
- [L-07] Function documentation incorrect for `Slingshot::_transferFromOrWrap` *Submitted by pmerkleplant.*
- [L-10] `LibERC20Token.approveIfBelow` should approve(0) first *Submitted by cmichel.*
- [L-11] Left-over tokens can be stolen *Submitted by cmichel.*
- [L-12] Confusing comment on IUniswapModule *Submitted by kenzo.*
- [L-13] Confusing comment in CurveModule *Submitted by kenzo.*
- [L-14] receive function *Submitted by pauliax.*

Non-Critical Findings (17)

- [N-01] ModuleRegistry doesn't need to know address of Slingshot.sol *Submitted by TomFrench.*
- [N-02] Malicious governance can abuse approvals to ApprovalHandler *Submitted by TomFrench.*
- [N-03] Flaws in Slingshot._sendFunds() *Submitted by daejunpark, also found by gpersoon, hickuphh3, kenzo, and pauliax.*
- [N-04] String concatenation in revert messages results in increased gas costs + code complexity *Submitted by TomFrench.*
- [N-05] Slingshot: Incorrect comment for rescueTokensFromExecutioner() *Submitted by hickuphh3.*
- [N-06] Slingshot: Index fromToken and toToken for Trade event *Submitted by hickuphh3.*

- [N-07] Inconsistent naming for functions in `ConcatStrings.sol` *Submitted by pmerkleplant.*
- [N-08] Error messages in `ModuleRegistry.sol` inconsistent to the rest of the project *Submitted by pmerkleplant.*
- [N-09] `Adminable::setupAdmin` uses deprecated function *Submitted by pmerkleplant.*
- [N-10] Code Style: Abstract contracts should not be prefixed by `I` *Submitted by WatchPug.*
- [N-11] `SlingshotI` is unnecessary *Submitted by WatchPug.*
- [N-12] Code Style: consistency *Submitted by WatchPug.*
- [N-13] Outdated compiler version *Submitted by WatchPug.*
- [N-14] Typos *Submitted by WatchPug.*
- [N-15] `Slingshot._sendFunds` function not used and wrong *Submitted by cmichel, also found by csanuragjain, WatchPug, WatchPug, and yeOlde.*
- [N-16] `CurveModule.sol#swap()` Unused parameter *Submitted by WatchPug.*
- [N-17] Redundant code *Submitted by WatchPug.*

Gas Optimizations (24)

- [G-01] Use of constant `keccak` variables results in extra hashing (and so gas). *Submitted by TomFrench.*
- [G-02] _transferFromOrWrap could be set private to save gas *Submitted by pants.*
- [G-03] The function _getTokenBalance could be set private to save gas *Submitted by pants.*
- [G-04] The function _sendFunds could be set private to save gas *Submitted by pants.*
- [G-05] A more efficient for loop index proceeding *Submitted by pants.*
- [G-06] nonReentrant modifier isn’t necessary for executeTrades function *Submitted by pants.*
- [G-07] getRouter methods could be set external instead public *Submitted by pants.*

- [G-08] Small gas improvement *Submitted by onewayfunction.*
- [G-09] Unnecessary Use of _msgSender() *Submitted by zer0dot.*
- [G-10] Redundant Code Statement *Submitted by defsec.*
- [G-11] Unused Named Returns (ConcatStrings.sol) *Submitted by yeOlde.*
- [G-12] Long Revert Strings *Submitted by yeOlde, also found by WatchPug.*
- [G-13] `> 0` can be replaced with `!= 0` for gas optimisation *Submitted by 0x0x0x.*
- [G-14] CurveModule: Redundant jToken *Submitted by hickuphh3, also found by pmerkleplant.*
- [G-15] ModuleRegistry: Rename modulesIndex → isModule *Submitted by hickuphh3.*
- [G-16] Avoid unnecessary code execution can save gas *Submitted by WatchPug.*
- [G-17] `IUniswapModule.sol` use an immutable variable `router` can save gas and simplify implementation *Submitted by WatchPug.*
- [G-18] Remove redundant access control checks can save gas *Submitted by WatchPug, also found by pauliax.*
- [G-19] Cache array length in for loops can save gas *Submitted by WatchPug, also found by 0x0x0x, pants, and pants.*
- [G-20] Adding unchecked directive can save gas *Submitted by WatchPug, also found by pauliax.*
- [G-21] Avoid unnecessary storage read can save gas *Submitted by WatchPug, also found by 0x0x0x.*
- [G-22] Combine external calls into one can save gas *Submitted by WatchPug.*
- [G-23] Gas: Use a constant instead of `block.timestamp` for the deadline *Submitted by cmichel.*
- [G-24] ConcatStrings prependNumber is not used *Submitted by kenzo, also found by pauliax.*

Disclosures

C4 is an open organization governed by participants in the community.