SUB7
Web3 Security

Holoride - Holoride DeFi
Security Assessment Findings Report

Date: July 24, 2023
Project: HOL
Version 1.0

Contents

1 Confidentiality statement

This document is the exclusive property of Holoride and Sub7 Security. This document contains proprietary and confidential information. Duplication, redistribution, or use, in whole or in part, in any form, requires consent of both Holoride and Sub7 Security.

2 Disclaimer

A smart contract security audit is considered a snapshot in time. The findings and recommendations reflect the information gathered during the assessment and not any changes or modifications made outside of that period.

Time-limited engagements do not allow for a full evaluation of all security controls. Sub7 Security prioritized the assessment to identify the weakest security controls an attacker would exploit. Sub7 Security recommends conducting similar assessments on an annual basis by internal or third-party assessors to ensure the continued success of the controls

3 About Sub7

Sub7 is a Web3 Security Agency, offering Smart Contract Auditing Services for blockchain-based projects in the DeFi, Web3 and Metaverse space.

Learn more about us at https://sub7.xyz

4 Project Overview

Holoride is turning vehicles into moving theme parks while revolutionizing in-car entertainment for you. Transit time will never be the same again. Buckle up, put on your VR headset and explore games and experiences with the holoride app, motion-synced with the car.

RIDE is the utility token of the holoride platform that sits at the heart of its ecosystem, built on the MultiversX Network. RIDE is designed to supercharge the connection between holoride users, creators, and business partners by providing additional benefits and enhanced user engagement.

RIDE is being built on top of the proprietary core technology provided by holoride GmbH. The RIDE token is issued by holoride AG, Liechtenstein.

RIDE staking and farming smart contracts for EVM blockchain (currently developed for Ethereum).

5 Executive Summary

Sub7 Security has been engaged to what is formally referred to as a Security Audit of Solidity Smart Contracts, a combination of automated and manual assessments in search for vulnerabilities, bugs, unintended outputs, among others inside deployed Smart Contracts.

The goal of such a Security Audit is to assess project code (with any associated specification, and documentation) and provide our clients with a report of potential security-related issues that should be addressed to improve security posture, decrease attack surface and mitigate risk.

2 (Two) Security Researchers/Consultants were engaged in this activity.

5.1 Scope

Assessment:

- https://github.com/holoride/ride-defi/commit/0557444ec0b3a4df44e0cf6c9bcd8e06eaea63c4

Verification:

- https://github.com/holoride/ride-defi/pull/3

5.2 Timeline

Assessment: 21st June 2023 to 27th June 2023

Verification: 24th July 2023

5.3 Summary of Findings Identified

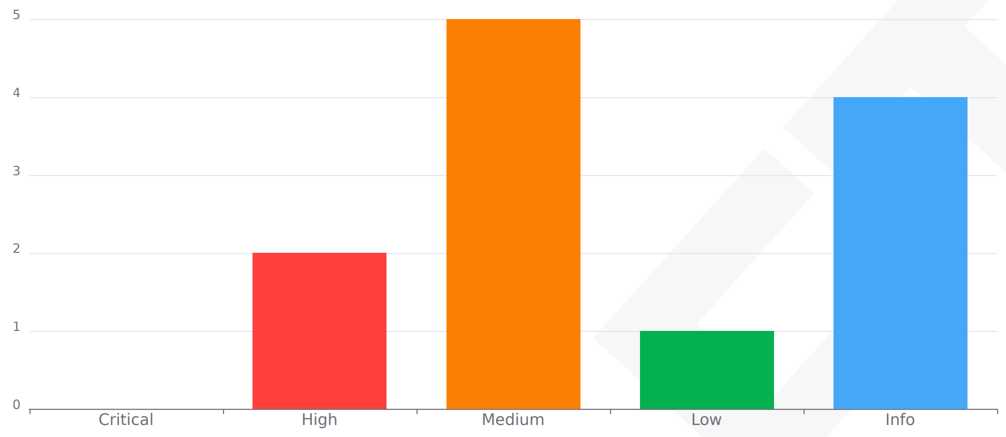

Figure 1: Executive Summary

# 1 High The actor funding the farming contract may lose funds – ***Fixed***

# 2 High Reward tokens are lost in the period where there are no stakers, but reward distribution has started – ***Fixed***

# 3 Medium Protocol does not support fee-on-transfer or rebasing tokens – ***Fixed***

# 4 Medium Absence of `safeTransfer` usage for transferring unknown erc20 tokens – ***Fixed***

# 5 Medium Wrong amount of tokens minted – ***Fixed***

# 6 Medium Rewards are lost for stakers due to sanity check in `_rewards()` – ***Fixed***

# 7 Medium Users cannot `withdraw` or `deposit` if `pendingAmount == 0` for tokens that disallow 0 value transfer – ***Fixed***

# 8 Low No input sanitization for endBlock and startBlock – ***Fixed***

# 9 Informational Use of redundant libraries – ***Fixed***

# 10 Informational Redundant check implemented – ***Fixed***

# 11 Informational Initialisation of variables to 0 is not neccessary – ***Fixed***

# 12 Informational Add input sanitisation for `STAKING_TERM` – ***Fixed***

5.4 Methodology

Sub7 Security follows a phased assessment approach that includes thorough application profiling, threat analysis, dynamic and manual testing. Security Auditors/Consultants utilize automated security tools, custom scripts, simulation apps, manual testing and validation techniques to scan for, enumerate and uncover findings with potential security risks.

As part of the process of reviewing solidity code, each contract is checked against lists of known smart contract vulnerabilities, which is crafted from various sources like SWC Registry , DeFi threat and previous audit reports.

The assessment included (but was not limited to) reviews on the following attack vectors:

Oracle Attacks | Flash Loan Attacks | Governance Attacks | Access Control Checks on Critical Function | Account Existence Check for low level calls | Arithmetic Over/Under Flows | Assert Violation | Authorization through tx.origin | Bad Source of Randomness | Block Timestamp manipulation | Bypass Contract Size Check | Code With No Effects | Delegatecall | Delegatecall to Untrusted Callee | DoS with (Unexpected) revert | DoS with Block Gas Limit | Logical Issues | Entropy Illusion | Function Selector Abuse | Floating Point and Numerical | Precision | Floating Pragma | Forcibly Sending Ether to a Contract | Function Default Visibility | Hash Collisions With Multiple Variable Length Arguments | Improper Array Deletion | Incorrect interface | Insufficient gas griefing | Unsafe Ownership Transfer | Loop through long arrays | Message call with hardcoded gas amount | Outdated Compiler Version | Precision Loss in Calculations | Price Manipulation | Hiding Malicious Code with External Contract | Public burn() function | Race Conditions / Front Running | Re-entrancy | Requirement Violation | Right-To-Left-Override control character (U+202E) | Shadowing State Variables | Short Address/Parameter Attack | Signature Malleability | Signature Replay Attacks | State Variable Default Visibility | Transaction Order Dependence | Typographical Error | Unchecked Call Return Value | Unencrypted Private Data On-Chain | Unexpected Ether balance | Uninitialized Storage Pointer | Unprotected Ether Withdrawal | Unprotected SELFDESTRUCT Instruction | Unprotected Upgrades | Unused Variable | Use of Deprecated Solidity Functions | Write to Arbitrary Storage Location | Wrong inheritance | Many more…

6 Findings and Risk Analysis

6.1 The actor funding the farming contract may lose funds

Severity: High
Status: Fixed

Description

The function `fund` doesn't check if the amount to fund is multiple of `rewardPerBlock`. Thus, if the amount to fund isn't the multiple of `rewardPerBlock`, then the remainder of `amount/rewardPerBlock` gets lost in the farming contract and can't be used as a reward. The ideal behavior is to refund the excess fund to the funder since those funds won't be able to be used by anyone and will get lost.

Location

https://github.com/sub7security/holoride-defi/blob/main/contracts/Farming.sol#L89-L96

Recommendation

It would be better to disallow funding of `amount` that is not perfectly divisible by `rewardPerBlock` .

Client Comments

None

6.2 Reward tokens are lost in the period where there are no stakers, but reward distribution has started

Severity: High
Status: Fixed

Description

Each time funders fund the farming contract, `endBlock` duration is extended based on the amount funded, some multiples of `rewardPerBlock`. This reward will only be fully utilised if in the duration from `startBlock` to `endBlock`, there must be at least 1 user who has deposited liquidity into the pool, i.e. `pool.lpToken.balanceOf(address(this))> 0`.

Here are a few scenerios that the loss will happen in.

1. Farming contract is funded, no stakers have staked LP tokens yet, but `block.number >= startBlock` .
2. All of liquidity is withdrawn by users before the end of farming period and farming period has already started.

Impact is that reward tokens funded into contract will be lost propotionately to the amount of blocks that have been mined for as long as there are no liquidity added, and will not be recoverable even if liquidity is added at a later block.

Location

https://github.com/sub7security/holoride-defi/blob/0557444ec0b3a4df44e0cf6c9bcd8e06eaea63c4/contracts/Farming.sol#L89-L96

Recommendation

One way we can prevent loss of funded tokens, would be for owner to deposit 1 wei of LP token before the `startPeriod` and only withdraw after `endPeriod`. We can either choose to implement this deposit on the smart contract level, or document it such that owner will remember to deposit when the farming contract is deployed. This way, all farming rewards will be captured and since it is only 1 wei, it will not impact rewards that actual user can farm by much.

Client Comments

None

6.3 Protocol does not support fee-on-transfer or rebasing tokens

Severity: Medium
Status: Fixed

Description

If fee-on-transfer tokens or rebasing tokens are used, protocol may have missing rewards that are not accounted for, causing users to be unable to claim what they thought they are able to.

For example, in the staking contract, when rewards are added, `amount` is added to `availableRewards`. This is the amount that is transferred, and not the post-fee `amount` that actually exist in the contract. This means that there is an over computation.

Some of the more popular tokens that fall into these categories would be USDT, while currently do not have fee on transfer, its contract implementation allows fee to be taken on transfer. stETH would be a popular token example of rebasing token.

Location

https://github.com/sub7security/holoride-defi/blob/0557444ec0b3a4df44e0cf6c9bcd8e06eaea63c4/contracts/Staking.sol#L213-L221

Recommendation

Holoride should be careful to not use tokens that are either fee-on-transfer or rebasing. Otherwise, if Holoride wants to support such tokens, changes to the implementation must be done.

Client Comments

None

6.4 Absence of `safeTransfer` usage for transferring unknown erc20 tokens

Severity: Medium
Status: Fixed

Description

Reward tokens and lpTokens are ERC20 tokens which may have different behaviour for different tokens. Some tokens may revert on transfer while some may return true/false while some may not return anything. In this case, it is advisable to use `SafeERC20` library for token transfers. This library is used in the contract but lacks usage of `safeTransfer` at above location.

Location

https://github.com/sub7security/holoride-defi/blob/main/contracts/Farming.sol#L298

Recommendation

Use `safeTransfer` instead of normal `transfer` to avoid weird behaviors of ERC20 tokens.

Client Comments

None

6.5 Wrong amount of tokens minted

Severity: Medium
Status: Fixed

Description

The amount of tokens minted to the `msg.sender` i.e. deployer of contract is different from what is intended. The comment says to mint `1 million` tokens but the tokens minted is `1 billion`.

Location

https://github.com/sub7security/holoride-defi/blob/main/contracts/mocks/GenericERC20.sol#L18-L19

Recommendation

Either mint the `1 million` tokens or correct the documentation(comment) if tokens intended to mint is `1 billion`.

Client Comments

None

6.6 Rewards are lost for stakers due to sanity check in `_rewards()`

Severity: Medium
Status: Fixed

Description

In the calculations for staking rewards, a sanity check is done as seen below.

```
// Sanity check. Do not transfer more than what's in the contract

if (totalRewards > availableRewards) {

 totalRewards = availableRewards;

}
```

This ensures that the rewards given to stakers cannot exceed what is available. This appears sound, but it means that stakers who happen to unstake when the contract is not funded sufficiently yet will lose the rewards they are entitled to. Their stake position will be removed, and they claim less than what they should have.

We also want to highlight that this implementation is inconsistent with 'unstakeSingle()'. No such sanity check is found in `unstakeSingle()`, and if the contract is insufficiently funded, transaction will simply revert, preventing user from unstaking.

Location

https://github.com/sub7security/holoride-defi/blob/0557444ec0b3a4df44e0cf6c9bcd8e06eaea63c4/contracts/Staking.sol#L291-L292

Recommendation

Consider removing the sanity check, and instead ensure that contract is well funded. Otherwise, we should implement the same sanity check in `unstakeSingle()`.

Client Comments

None

6.7 Users cannot `withdraw` or `deposit` if `pendingAmount == 0` for tokens that disallow 0 value transfer

Severity: Medium
Status: Fixed

Description

Some tokens such as the old aave token LEND will revert when a value of 0 is transferred. If such tokens are used, it can prevent `withdraw` or `deposit` in some cases. This will happen on the condition that

> `uint256 pendingAmount = user.amount.mul(pool.accERC20PerShare).div(1e36).sub(user.rewardDebt) = 0`.

In what cases will this happen?

This happens only when `user.amount.mul(pool.accERC20PerShare).div(1e36).sub(user.rewardDebt)== 0`, i.e. `user.amount.mul(pool.accERC20PerShare).div(1e36)== user.rewardDebt`. There are 2 cases where this will occur. Specifically, it happens when no rewards are accrued yet as we are on the same block or actions are made before `startBlock`.

1. Some combination of depositing and withdrawing in the same block. For eg, depositing twice in the same block, or depositing and withdrawing in the same block, or withdrawing twice a different amount in the same block.
2. Deposit happens when `block.number == endBlock`. Note that this is possible as deposit checks `require(block.number <= endBlock, "deposit: cannot deposit after end block");`. In this case, the normal withdraw will not be possible since no rewards are accured.
3. A combination of any of the above is done before `startBlock`, for eg, a user deposits some amount, and wants to make another deposit will not be able to do so since reward accuring has not started.

Fortunately, user's LP stake position will not be lost as they can `emergencyWithdraw`. However, since we believe this issue is not intended and it is a form of DOS for some tokens, we put it at medium severity.

Location

https://github.com/sub7security/holoride-defi/blob/0557444ec0b3a4df44e0cf6c9bcd8e06eaea63c4/contracts/Farming.sol#L247

https://github.com/sub7security/holoride-defi/blob/0557444ec0b3a4df44e0cf6c9bcd8e06eaea63c4/contracts/Farming.sol#L269

Recommendation

Consider making the ERC20 transfer only if `pendingAmount > 0` in both `deposit` and `withdraw`.

Client Comments

None

6.8 No input sanitization for endBlock and startBlock

Severity: Low
Status: Fixed

Description

`erc20`, `rewardPerBlock`, `endBlock` and `startBlock` are set in the constructor and can't be modified later. The arguments passed are not checked if they are valid and non-zero. Also, contract can be funded only when `block.number < endBlock`. So, if `startBlock` is set to `block.number`, then contract won't be funded and will be useless. Also, it needs to be ensured that `startBlock` is not too much in the future otherwise contract will be open for funding but reward distribution will start too long in the future.

Location

https://github.com/sub7security/holoride-defi/blob/main/contracts/Farming.sol#L64-L73

Recommendation

Add basic sanitization for input value considering above cases.

Client Comments

None

6.9 Use of redundant libraries

Severity: Informational
Status: Fixed

Description

The contract uses `^0.8.18` version which has inbuilt arithmetic overflow and underflow protection at compiler level. Thus, usage of `SafeMath` library is not required since any arithmetic underflow or overflow due to addition, substraction or multiplication will lead to error.

Location

https://github.com/sub7security/holoride-defi/blob/main/contracts/Farming.sol#L11

Recommendation

Remove the redundant `SafeMath` library.

Client Comments

None

6.10 Redundant check implemented

Severity: Informational
Status: Fixed

Description

In function `pending`, variable `lastBlock` is calculated as the minimum of current block(i.e. block.number) and endBlock (i.e. `lastBlock = min(block.number, endBlock)`). The `if` condition check is as follows:

```
if (lastBlock > pool.lastRewardBlock && block.number > pool.lastRewardBlock && lpSupply != 0)
```

the check of `block.number > pool.lastRewardBlock` is not necessary since `lastBlock > pool.lastRewardBlock` already checks this.

Location

https://github.com/sub7security/holoride-defi/blob/main/contracts/Farming.sol#L171-L173

Recommendation

Remove the redundant check of `block.number > pool.lastRewardBlock`

Client Comments

None

6.11 Initialisation of variables to 0 is not neccessary

Severity: Informational
Status: Fixed

Description

In solidity, all variables defaults to the 0 value. There is no need to explicitly set them to 0 since they have an addtional gas cost.

Location

https://github.com/sub7security/holoride-defi/blob/0557444ec0b3a4df44e0cf6c9bcd8e06eaea63c4/contracts/Farming.sol#L51

https://github.com/sub7security/holoride-defi/blob/0557444ec0b3a4df44e0cf6c9bcd8e06eaea63c4/contracts/Farming.sol#L42

Recommendation

Consider making the below changes in farming contract.

```
Consider making the below changes in farming contract.

- uint256 public paidOut = 0;
+ uint256 public paidOut;

- uint256 public totalAllocPoint = 0;
+ uint256 public totalAllocPoint;
```

Client Comments

None

6.12 Add input sanitisation for `STAKING_TERM`

Severity: Informational
Status: Fixed

Description

`STAKING\_TERM` is used to determine how long a token should be staked, before rewards are given. If this value is set to too large, no stakers can receive rewards in a reasonable timespan. This means that the reward tokens can potentially be stuck in the contract, since reward tokens can only be withdrawn by a staker who has staked the duration of `STAKING\_TERM`.

Location

https://github.com/sub7security/holoride-defi/blob/0557444ec0b3a4df44e0cf6c9bcd8e06eaea63c4/contracts/Staking.sol#L56

Recommendation

Consider restricting `STAKING\_TERM` to a reasonable timespan. For example, `require(\_stakingTerm < 1 year` in the constructor to prevent admin mistake.

Client Comments

None