SECURITY REVIEW REPORT

for

ALPHA FINANCE

Prepared By: Shuxiao Wang

Hangzhou, China
Oct. 5, 2020

Document Properties

Client	Alpha Finance
Title	Security Review Report
Target	Alpha Homora
Version	1.0
Author	Chiachih Wu
Auditors	Chiachih Wu, Huaguo Shi
Reviewed by	Jeff Liu
Approved by	Xuxian Jiang
Classification	Confidential

Version Info

Version	Date	Author(s)	Description
1.0	Oct. 5, 2020	Chiachih Wu	Final Release

Contact

For more information about this document and its contents, please contact PeckShield Inc.

Name	Shuxiao Wang
Phone	+86 173 6454 5338
Email	contact@peckshield.com

1 Introduction

PeckShield team has completed an informal security review of the Alphahomora smart contract, and here in this report we will summarize the result of the review. Alpha Homora is a protocol for leveraging your position in yield farming pools, or dYdX for yield farming on Uniswap. Users can participate in Alpha Homora protocol as yield farmers, ETH lenders, liquidators, or bounty hunters.

Here is the Git repository of reviewed files and the commit hash value used in this security review:

- https://github.com/AlphaFinanceLab/alphahomora (cde1a1b)

2 | Review Findings

Here is a summary of our findings after analyzing the alphahomora smart contract. We feel that the implementation can be improved by fixing the following issues, including 2 low-severity vulnerabilities, and 2 informational recommendations. The details of each issue are listed in this section.

2.1 PVE-001: Missed approve(spender, 0) Call

- **Severity:** Low
- **Details:** While interacting with Uniswap, Alpha Homora fails to `approve(spender,0)` before approving the `router` contract to spend the `fToken`. In most cases, there's no problem. However, there're some ERC20 contracts which revert the `approve()` call when the current approval is non-zero [1]. That's one solution to deal with the approve()/transferFrom() race condition [2]. Based on that, if `fToken` happens to be one of such ERC20 tokens, the `execute()` calls in `StrategyLiquidate` and `StrategyAllETHOnly` contracts would be reverted, which leads to a compatibility issue.
- **Status:** This issue had been addressed by adding `approve(spender,0)` calls in this commit: acdff19

2.2 PVE-002: Improved Ether Transfer Call

- **Severity:** Low
- **Details:** While reviewing the Ether/ERC20 transfer calls in Alpha Homora, we notice that the `SafeToken` contract is used to deal with the compatibility issues which cannot be properly handled with simple `transfer()` or `transferFrom()` calls. However, in `Gringotts::reducio()`, we identify one `msg.sender.transfer()` call which is not consistent with other ether transfers. We suggest to use `SafeToken.SafeTransferETH()` instead.

- **Status:** This issue had been addressed by replacing the `transfer()` call with `SafeToken.SafeTransferETH()` in this commit: 2c62d06

2.3 PVE-003: Randomly Received Ether Interfere With the Calculation

- **Severity:** Informational
- **Details:** While reviewing the `payable` functions in Alpha Homora codebase, we notice that there's a fallback function in the `Gringotts` contract for receiving ether from the `Goblin` contract. Since that fallback function has no permission check, random users could send ether into the `Gringotts` contract such that the `totalETH()` may contain some ether which are not in any book-keeping records. Therefore, the `engorgio()` and `reducio()` functions which do the math based on `totalETH()` could be inaccurate. We suggest to add permission checks in all fallback functions such that we can regulate which contract could send ether to which contracts and get rid of unintentionally ether transfers.
- **Status:** As we discussed with the team, the randomly received ether would not greatly interfere with the calculation. We leave it as is.

2.4 PVE-004: Known Issue in Solidity v0.5.16

- **Severity:** Informational
- **Details:** There is a known compiler issue that in all 0.5.x solidity prior to `Solidity 0.5.17`. Specifically, a private function can be overridden in a derived contract by a private function of the same name and types. Fortunately, there is no overriding issue in this code, but we still recommend using `Solidity 0.5.17` or above.
- **Status:** As we discussed with the team, Alpha Homora uses an old Solidity compiler (v0.5.16) for facilitating Synthetix and Uniswap contracts. Since there's no overriding issue in the code, we leave it as is.

3 | Disclaimer

This is an informal security review, not a full security audit, and it does not give any warranties on finding all possible security issues of the given smart contract(s), i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. Furthermore, we always recommend proceeding with several independent full audits and a public bug bounty program to ensure the security of smart contract(s). Lastly, this security review report should not be used as investment advice.

References

[1] dForce Network. dForce (DF). https://etherscan.io/address/0x431ad2ff6a9c365805ebad47ee021148d6f7dbe0#code.

[2] HaleTom. Resolution on the EIP20 API Approve / TransferFrom multiple withdrawal attack. https://github.com/ethereum/EIPs/issues/738.