HAECHI AUDIT

Ithil

Smart Contract Security Analysis

Published on : June 17, 2022

Version v2.0

HAECHI AUDIT

Smart Contract Audit Certificate

Ithil

Security Report Published by HAECHI AUDIT
v1.0 June 10, 2022
v2.0 June 17, 2022

Auditor : Felix Kim

Executive Summary

Severity of Issues	Findings	Resolved	Unresolved	Acknowledged	Comment
Critical	-	-	-	-	-
Major	-	-	-	-	-
Minor	1	1	-	-	-
Tips	-	-	-	-	-

TABLE OF CONTENTS

0 Issues (0 Critical, 0 Major, 1 Minor, 0 tips) Found

ABOUT US

HAECHI AUDIT believes in the power of cryptocurrency and the next paradigm it will bring.

We have the vision to *empower the next generation of finance*. By providing security and trust in the blockchain industry, we dream of a world where everyone has easy access to blockchain technology.

HAECHI AUDIT is a flagship service of HAECHI LABS, the leader of the global blockchain industry. HAECHI AUDIT provides specialized and professional smart contract security auditing and development services.

We are a team of experts with years of experience in the blockchain field and have been trusted by 300+ project groups. Our notable partners include Universe,1inch, Klaytn, Badger, etc.

HAECHI AUDIT is the only blockchain technology company selected for the Samsung Electronics Startup Incubation Program in recognition of our expertise. We have also received technology grants from the Ethereum Foundation and Ethereum Community Fund.

Inquiries: audit@haechi.io

Website: audit.haechi.io

INTRODUCTION

This report was prepared to audit the security of the smart contract created by the Ithil team. HAECHI AUDIT conducted the audit focusing on whether the smart contract created by the Ithil team is soundly implemented and designed as specified in the published materials, in addition to the safety and security of the smart contract.

CRITICAL	Critical issues must be resolved as critical flaws that can harm a wide range of users.
MAJOR	Major issues require correction because they either have security problems or are implemented not as intended.
MINOR	Minor issues can potentially cause problems and therefore require correction.
TIPS	Tips issues can improve the code usability or efficiency when corrected.

HAECHI AUDIT recommends that the Ithil team improve all issues discovered.

The following issue explanation uses the format of {file name}#{line number}, {contract name}#{function/variable name} to specify the code. For instance, *Sample.sol:20* points to the 20th line of Sample.sol file, and *Sample#fallback()* means the fallback() function of the Sample contract.

Please refer to the Appendix to check all results of the tests conducted for this report.

SUMMARY

The codes used in this Audit can be found on the following repository and commit hash.

Repository : https://github.com/Ithil-protocol/v1-core

Commit hash : 9377a0cd30d5b584a4d30cf476a2776a014d157d

Issues	HAECHI AUDIT found 0 critical issues, 0 major issues, and 1 minor issues. There are 0 Tips issues explained that would improve the code's usability or efficiency upon modification.
Update	[v2.0] - In a new commit "ad60b0e8bd6dba28b43c9ba60260bca505519bc3", 1 minor issue has been resolved.

Severity	Issue	Status
MINOR	The logic of Vault#stake() and Vault#stakeETH() does not match when the user stakes ETH or Wrapped ETH	(Found - v1.0) (Resolved - v2.0)

OVERVIEW

Contracts subject to audit

- ❖ ICurve
- ❖ IKyberNetworkProxy
- ❖ IStETH
- ❖ IStrategy
- ❖ IVault
- ❖ IWETH
- ❖ IWrappedToken
- ❖ IYearnPartnerTracker
- ❖ IYearnRegistry
- ❖ IYearnVault
- ❖ GeneralMath
- ❖ PositionHelper
- ❖ TransferHelper
- ❖ VaultMath
- ❖ VaultState
- ❖ WrappedTokenHelper
- ❖ Liquidator
- ❖ AbstractStrategy
- ❖ BaseStrategy
- ❖ LidoStrategy
- ❖ Liquidable
- ❖ MarginTradingStrategy
- ❖ YearnStrategy
- ❖ Vault
- ❖ WrappedToken

FINDINGS

MINOR

The logic of Vault#stake() and Vault#stakeETH() does not match when the user stakes ETH or Wrapped ETH. (Found - v.1.0) (Resolved - v.2.0)

```
   function stake(address token, uint256 amount) external override unlocked(token)
isValidAmount(amount) {
       checkWhitelisted(token);
       IWrappedToken wToken = IWrappedToken(vaults[token].wrappedToken);
       uint256 totalWealth = balance(token);
       uint256 stakingCap = vaults[token].stakingCap;

       if (totalWealth + amount > stakingCap) revert Vault__Staking_Cap_Exceeded(token,
totalWealth, stakingCap);

       (, amount) = IERC20(token).transferTokens(msg.sender, address(this), amount);

       uint256 toMint = wToken.mintWrapped(amount, msg.sender, totalWealth);

       emit Deposit(msg.sender, token, amount, toMint);
   }

   function stakeETH(uint256 amount) external payable override unlocked(weth)
isValidAmount(amount) {
       checkWhitelisted(weth);

       if (msg.value != amount) revert Vault__Insufficient_ETH(msg.value, amount);

       IWrappedToken wToken = IWrappedToken(vaults[weth].wrappedToken);
       uint256 totalWealth = balance(weth);
       IWETH(weth).deposit{ value: amount }();

       uint256 toMint = wToken.mintWrapped(amount, msg.sender, totalWealth);

       emit Deposit(msg.sender, weth, amount, toMint);
   }
```

[https://github.com/Ithil-protocol/v1-core/blob/master/contracts/Vault.sol#L145-L172]

Issue

User can stake the ethereum directly to the vault by the function *Vault#stakeETH()* or indirectly to the vault by the function *Vault#stake()* if the user has the Wrapped ETH. However, *Vault#stake()* checks the staking cap of the Wrapped ETH but *Vault#stakeETH()* don't. If Wrapped ETH's staking cap is set improperly, user cannot stake Wrapped ETH.

Recommendation

We recommend that you add a statement to the *Vault#stakeETH()* function to check the stakingCap. If you do not want to impose a stakingCap limit on Wrapped ETH, we recommend that you make an exception in the *Vault#stake()* function.

Update

[v2.0] - The issue is resolved by adding the logic to check stakingCap in *Vault#stakeETH()*.

DISCLAIMER

This report does not guarantee investment advice, the suitability of the business models, and codes that are secure without bugs. This report shall only be used to discuss known technical issues. Other than the issues described in this report, undiscovered issues may exist such as defects on DFX Finance. In order to write secure smart contracts, correction of discovered problems and sufficient testing thereof are required.

Appendix A. Test Results

The following results show the unit test results covering the key logic of the smart contract subject to the security audit. Parts marked in red are test cases that failed to pass the test due to existing issues.

```
Ithil
    Vault spec
      #constructor()
        ✔ set WETH properly
      #addStrategy()
        ✔ should fail if msg.sender is not owner
        valid case
          ✔ add new strategy
      #removeStrategy()
        ✔ should fail if msg.sender is not owner
        valid case
          ✔ remove strategy
      #toggleLock()
        ✔ should fail if msg.sender is not owner
        valid case
          ✔ set locked properly
      #whitelistToken()
        ✔ should fail if msg.sender is not owner
        ✔ should fail if token is already supported
        valid case
          ✔ set variables properly (42ms)
      #whitelistTokenAndExec()
        ✔ should fail if msg.sender is not owner
        ✔ should fail if token is already supported
        valid case
          ✔ set variables properly (41ms)
      #editMinimumMargin()
        ✔ should fail if msg.sender is not owner
        ✔ should fail if token is not whitelisted
        valid case
          ✔ set minimum margin properly
      #editCap()
        ✔ should fail if msg.sender is not owner
        ✔ should fail if token is not whitelisted
        valid case
          ✔ set staking cap properly
      #stake()
        ✔ should fail if token is locked
        ✔ should fail if token is not whitelisted
    ✔ should fail if amount is invalid
    ✔ should fail if try to stake over cap
    valid case
      ✔ transfer origin token to vault contract
      ✔ transfer wrapped token to msg.sender
  #stakeETH()
    ✔ should fail if WETH is locked
    ✔ should fail if WETH is not whitelisted
    ✔ should fail if amount is invalid
    ✔ should fail if msg.value is different from amount
    valid case
      ✔ deposit ETH
      ✔ transfer wrapped token to msg.sender
  #unstake()
    ✔ should fail if token is not whitelisted
    ✔ should fail if amount is invalid
    ✔ should fail if try to unstake over deposited amount
    valid case
      ✔ burn wrapped token of msg.sender
      ✔ transfer origin token to msg.sender
  #unstakeETH()
    ✔ should fail if WETH is not whitelisted
    ✔ should fail if amount is invalid
    valid case
      ✔ burn wrapped token of msg.sender
      ✔ transfer ETH to msg.sender
  #borrow()
    ✔ should fail if msg.sender is not strategy
    ✔ should fail if token is locked
    ✔ should fail if token is not whitelisted
    valid case
      ✔ transfer token to Strategy contract
      ✔ set variables properly
  #repay()
    ✔ should fail if msg.sender is not strategy
    ✔ should fail if token is not whitelisted
    valid case
      ✔ should repay and set variables properly
  View Functions
    #checkWhitelisted()
      ✔ should fail if token is not supported
    #getMinimumMargin()
      ✔ should return minimum margin properly
    #balance()
      ✔ should return balance of vaults
    #claimable()
      ✔ should fail if token is not supported
      valid case
        ✔ should return claimable value
  Ithil
    WrappedToken spec
      #constructor()
        ✔ set nativeToken properly
      #decimals()
        ✔ should return decimals of nativeToken
      #mint()
        ✔ should fail if msg.sender is not owner
        valid case
          ✔ mint token to user
      #burn()
        ✔ should fail if msg.sender is not owner
        valid case
          ✔ burn token of user
scenario test
    position open
      MarginTradingStrategy
        collateral == spent token
          ✔ check invalid parameters
          ✔ successfully opened
        collateral == obtained token
          ✔ check invalid parameters
          ✔ successfully opened
      YearnStrategy
        collateral == spent token
          ✔ check invalid parameters
          ✔ successfully opened
        collateral == obtained token
          ✔ check invalid parameters
          ✔ successfully opened
      LidoStrategy
        collateral == spent token
          ✔ check invalid parameters
          ✔ successfully opened
        collateral == obtained token
          ✔ check invalid parameters
          ✔ successfully opened
    position edit
      MarginTradingStrategy
        ✔ check blocked operation
        ✔ successfully edited
      YearnStrategy
        ✔ check blocked operation
        ✔ successfully edited
      LidoStrategy
        ✔ check blocked operation
        ✔ successfully edited
    position close
      MarginTradingStrategy
    ✔ check blocked operation
    ✔ successfully closed
  YearnStrategy
    ✔ check blocked operation
    ✔ successfully closed
  LidoStrategy
    ✔ check blocked operation
    ✔ successfully closed
liquidation / margin call
  MarginTradingStrategy
    ✔ liquidation fail
    ✔ liquidation success
    ✔ margincall success
  YearnStrategy
    ✔ liquidation fail
    ✔ liquidation success
    ✔ margincall success
  LidoStrategy
    ✔ liquidation fail
    ✔ liquidation success
    ✔ margincall success
```

End of Document