HAECHI AUDIT

Tadpole Finance

Smart Contract Security Analysis

Published on : Apr 25, 2022

Version v3.0

HAECHI AUDIT

Smart Contract Audit Certificate

Tadpole Finance

Security Report Published by HAECHI AUDIT
v1.0 Apr 01, 2022
v2.0 Apr 11, 2022
v3.0 Apr 25, 2022

Auditor : Felix Kim

Executive Summary

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

TABLE OF CONTENTS

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

TABLE OF CONTENTS

ABOUT US

INTRODUCTION

SUMMARY

OVERVIEW

FINDINGS

- The initial value of the owner of UniversalBridgeImplementation cannot be set. (Found - v.1.0) (Acknowledgement - v.2.0) (Resolved -v.3.0)
- Compile errors occur. (Found - v.1.0) (Resolved - v.2.0)

DISCLAIMER

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 in 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 Tadpole Finance team. HAECHI AUDIT conducted the audit focusing on whether the smart contract created by Tadpole Finance 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 Tadpole Finance 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 at GitHub (https://github.com/Tadpole-finance/tadpole-protocol/tree/master/contracts). The last commit of the code used in this Audit is “da2cab5f107accbedc695c8eff80be18310c2c80”.

Issues HAECHI AUDIT found 1 critical issue, 0 major issues, and 0 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 “e53e00774078efd33b37e727e6edd9ce9c5e4636”, 1 critical issue has been acknowledged and 1 minor issue has been resolved.

[v3.0] - From new UniversalBridgeImplementation.sol, 1 critical issue has been resolved.

Severity	Issue	Status
CRITICAL	The initial value of the owner of UniversalBridgeImplementation cannot be set.	(Found - v1.0) (Acknowledged - v2.0) (Resolved - v3.0)
MINOR	Compile errors occur.	(Found - v1.0) (Resolved - v2.0)

OVERVIEW

Contracts subject to audit

- CDaiDelegate
- CErc20
- CErc20Delegate
- CErc20Delegator
- CErc20Immutable
- CEther
- CToken
- CTokenFactory
- CTokenFactoryInterface
- CTokenInterfaces
- CarefulMath
- Comptroller
- ComptrollerG1
- ComptrollerG2
- ComptrollerG3
- ComptrollerInterface
- UnitrollerAdminStorage
- ComptrollerV1Storage
- ComptrollerV2Storage
- ComptrollerV3Storage
- ComptrollerTadpoleStorage
- DAIInterestRateModelV3
- EIP20Interface
- EIP20NonStandardInterface
- ComptrollerErrorReporter
- TokenErrorReporter
- Exponential
- InterestRateModelStorage
- InterestRateModel
- JumpRateModel
- JumpRateModelV2
- JumpRateModelV3
- InterestRateProxy
- JumpRateModelV3Storage
- Maximillion

- PriceOracle
- PriceOracleV1
- Reservoir
- SafeMath
- SimplePriceOracle
- Timelock
- Unitroller
- WhitePaperInterestRateModel
- ComptrollerLensInterface
- CompoundLens
- GovernorAlpha
- contracts/Governance/Tad.sol:Tad
- contracts/Governance/TadBsc.sol:Tad
- contracts/Governance/TadInterface.sol:Tad
- BEP20Factory
- Context
- IERC20
- Ownable
- UniversalBridgeImplementation
- UniversalBridgeProxy
- UniversalBridgeStorage

FINDINGS

CRITICAL

The initial value of the owner of UniversalBridgeImplementation cannot be set. (Found - v.1.0) (Acknowledgement - v.2.0) (Resolved -v.3.0)

```
   constructor() {}

   function initiate(address _signer, address _tadAddress, uint256 _tadBurnFee) external
onlyOwner {
       require(!initiated, "contract is already initiated");
       initiated = true;

       require(keccak256(bytes(IERC20(_tadAddress).name())) ==
keccak256(bytes('Tadpole')), "invalid Tadpole Address");
       signer     = _signer;
       tadAddress = _tadAddress;
       tadBurnFee = _tadBurnFee;
   }
```

Issue

The UniversalBridgeImplementation contract sets the initial information of the contract by the *UniversalBridgeImplementation#initiate()* function after being deployed. The *UniversalBridgeImplementation#initiate()* function has the onlyOwner modifier, so only the address set as the owner can call this function. However, because the constructor has no syntax that sets the owner, the owner variable is set as the value of address(0). Thus, there is no case where the *UniversalBridgeImplementation#initiate()* function can be called, causing the contract deployed not to be used normally.

Recommendation

We recommend adding a syntax in the constructor that sets msg.sender as the owner.

Acknowledgement

The team is aware of the problem. Also, the contract will not be called alone because it will be called using proxy.

Update

[v3.0] - The issue is resolved by adding the logic to initialize the owner in *UniversalBridgeImplementation#initiate()*.

MINOR

Compile errors occur. (Found - v.1.0) (Resolved - v.2.0)

```
   function setUnderlyingPrices(CToken[] memory cTokens, uint[] memory
underlyingPricesMantissa) public {

       senderMustBeReporter();

       require(cTokens.length == underlyingPricesMantissa.length, "cTokens and
underlyingPricesMantissa must be the same length");
       for(uint index=0; index<cTokens.length; index++){
           setUnderlyingPrice(cTokens[index], underlyingPricesMantissa[index]);
       }
   }

   function setDirectPrices(address[] memory assets, uint[] memory prices) public {

       senderMustBeReporter();

       require(cTokens.length == underlyingPricesMantissa.length, "cTokens and
underlyingPricesMantissa must be the same length");
       require(assets.length == prices.length, "cTokens and underlyingPricesMantissa
must be the same length");
       for(uint index=0; index<assets.length; index++){
           setDirectPrice(assets[index], prices[index]);
       }
   }
```

[https://github.com/Tadpole-finance/tadpole-protocol/blob/master/contracts/PriceOracleV1.sol#L42-L60]

Issue

In Solidity 0.5.x version, the reference type variable must have an explicitly specified data location (storage, memory, calldata). However, even though the *PriceOracleV1#setUnderlyingPrices()* and *PriceOracleV1#setDirectPrices()* functions take

an array, which is a reference type, as a parameter, the data location is not specified. Thus, a compile error occurs.
A compile error also occurs as it tries to import a Comp.sol contract that does not exist in the CompoundLens contract and the ComptrollerG3 contract.

Recommendation

We advise specifying the data location in the function where an error occurs. If you do not plan to use the contract where an import error occurs, please delete it. If you do intend to use it, we recommend changing Comp.sol to Tad.sol.

Update

[v2.0] - The issue has been resolved by correcting all parts where compilation errors occur.

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 Mainnet. In order to write secure smart contracts, correction of discovered problems and sufficient testing thereof are required.

End of Document