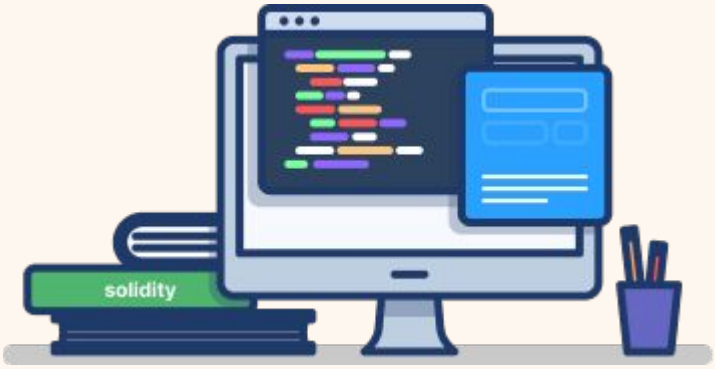
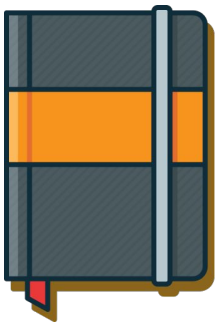
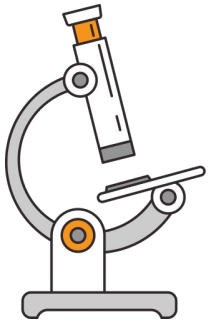
Coinbae Audit

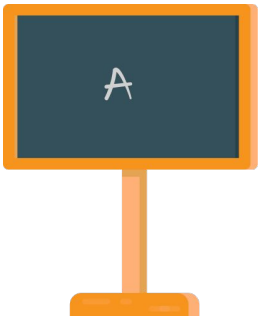


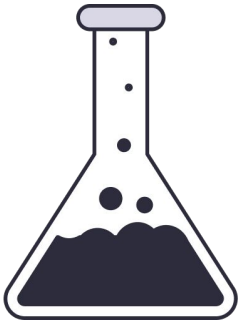
Hakka Finance Vault contract wHakka audit April 2021

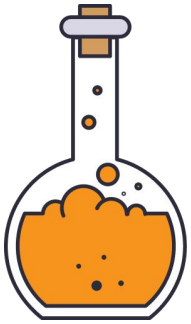
Contents











Introduction

Audit:

In April 2021 Coinbae's audit division performed an audit for the Hakka Finance Vault contract wHakka.sol.

https://github.com/artistic709/HakkaFinance/blob/master/vault/wHakka.sol

Hakka Finance

Hakka Finance is a set of decentralized finance (DeFi) applications, which form an all-inclusive ecosystem of tools that allow users to pursue financial sovereignty. The current product offering consists of a stablecoin automated market maker (AMM) DEX, a gamified insurance product, and a general DeFi handbook. All of this is administered and governed by a homonymous governance token, which goes by the ticker of HAKKA.

Scope of the audit:

The following Coinbae audit will cover assertions and property checking, ERC Standards, solidity coding best practices and conformity to the solidity style guide.

Overview:

Name: Hakka Finance wHakka.sol
Website: https://hakka.finance/

Audit Report Scope

Assertions and Property Checking:

1. Solidity assert violation.
2. Solidity AssertionFailed event.

ERC Standards:

1. Incorrect ERC20 implementation.

Solidity Coding Best Practices:

1. Outdated compiler version.
2. No or floating compiler version set.
3. Use of right-to-left-override control character.
4. Shadowing of built-in symbol.
5. Incorrect constructor name.
6. State variable shadows another state variable.
7. Local variable shadows a state variable.
8. Function parameter shadows a state variable.
9. Named return value shadows a state variable.
10. Unary operation without effect Solidity code analysis.
11. Unary operation directly after assignment.
12. Unused state variable.
13. Unused local variable.
14. Function visibility is not set.
15. State variable visibility is not set.

Solidity Coding Best Practices (Continued):

16. Use of deprecated functions: call code(), sha3(), ...
17. Use of deprecated global variables (msg.gas, ...).
18. Use of deprecated keywords (throw, var).
19. Incorrect function state mutability.
20. Does the code conform to the Solidity styleguide.

Convert code to conform Solidity style guide:

1. Convert all code so that it is structured accordingly the Solidity style guide.

Audit Report Scope

Categories:

High Severity:

High severity issues opens the contract up for exploitation from malicious actors. We do not recommend deploying contracts with high severity issues.

Medium Severity Issues:

Medium severity issues are errors found in contracts that hampers the effectiveness of the contract and may cause outcomes when interacting with the contract. It is still recommended to fix these issues.

Low Severity Issues:

Low severity issues are warning of minor impact on the overall integrity of the contract. These can be fixed with less urgency.

Optimization Issues:

Optimization issues are issues that pose no security risk or expose any underlying vulnerabilities, but instead make the contract more efficient.

Informational Issues:

Informational issues are issues that point to smart contract coding best practises.

Audit Report

19	19	0
Identified	Confirmed	Critical

0	1	6
High	Medium	Low

Optimization issues identified: 12

Analysis:
https://github.com/artistic709/HakkaFinance/blob/master/vault/wHakka.sol

Risk:
Low

Optimization issues identified:

External Function

renounceOwnership() should be declared external:
- Ownable.renounceOwnership() (wHakka.sol#203-206)

transferOwnership(address) should be declared external:
- Ownable.transferOwnership(address) (wHakka.sol#212-214)

totalSupply() should be declared external:
- ERC20.totalSupply() (wHakka.sol#433-435)

balanceOf(address) should be declared external:
- ERC20.balanceOf(address) (wHakka.sol#437-439)

allowance(address,address) should be declared external:
- ERC20.allowance(address,address) (wHakka.sol#441-443)

transfer(address,uint256) should be declared external:
- ERC20.transfer(address,uint256) (wHakka.sol#445-448)

approve(address,uint256) should be declared external:
- ERC20.approve(address,uint256) (wHakka.sol#450-454)

transferFrom(address,address,uint256) should be declared external:
- ERC20.transferFrom(address,address,uint256) (wHakka.sol#456-460)

stake(address,uint256,uint256) should be declared external:
- wHakka.stake(address,uint256,uint256) (wHakka.sol#555-572)

unstake(address,uint256,uint256) should be declared external:
- wHakka.unstake(address,uint256,uint256) (wHakka.sol#574-590)

Optimization issues identified:

External Function

inCaseTokenGetsStuckPartial(IERC20,uint256) should be declared external:
- wHakka.inCaseTokenGetsStuckPartial(IERC20,uint256) (wHakka.sol#592-595)

Low Issues

Reentrancy Events:

Reentrancy in wHakka.stake(address,uint256,uint256)
(wHakka.sol#555-572):
External calls:
- wAmount = getStakingRate(time).mul(amount).div(1e18)
(wHakka.sol#557)
- currentModel.stakingRate(time) (wHakka.sol#548)
State variables written after the call(s):
- _mint(to,wAmount) (wHakka.sol#568)
- _balances[to] = _balances[to].add(amount) (wHakka.sol#476)
- _mint(to,wAmount) (wHakka.sol#568)
- _totalSupply = _totalSupply.add(amount) (wHakka.sol#477)
- stakedHakka[to] = stakedHakka[to].add(amount) (wHakka.sol#564)
- votingPower[to] = votingPower[to].add(wAmount) (wHakka.sol#565)

Reentrancy in wHakka.stake(address,uint256,uint256)
(wHakka.sol#555-572):
External calls:
- wAmount = getStakingRate(time).mul(amount).div(1e18)
(wHakka.sol#557)
- currentModel.stakingRate(time) (wHakka.sol#548)
Event emitted after the call(s):
- Transfer(address(0),to,amount) (wHakka.sol#478)
- _mint(to,wAmount) (wHakka.sol#568)

Reentrancy in wHakka.stake(address,uint256,uint256)
(wHakka.sol#555-572):
External calls:
- wAmount = getStakingRate(time).mul(amount).div(1e18)
(wHakka.sol#557)
- currentModel.stakingRate(time) (wHakka.sol#548)
- Hakka.safeTransferFrom(msg.sender,address(this),amount)
(wHakka.sol#569)
Event emitted after the call(s):
- Stake(to,msg.sender,amount,wAmount,time) (wHakka.sol#571)

Low Issues

Reentrancy Events:

Reentrancy in wHakka.unstake(address,uint256,uint256) (wHakka.sol#574-590):
External calls:
- Hakka.safeTransfer(to,amount) (wHakka.sol#587)
Event emitted after the call(s):
- Unstake(msg.sender,to,amount,wAmount) (wHakka.sol#589)

Timestamp:

stakingRateModel.stakingRateMax() (wHakka.sol#506-514) uses timestamp for comparisons
Dangerous comparisons:
- timeElapsed > 0 (wHakka.sol#508)

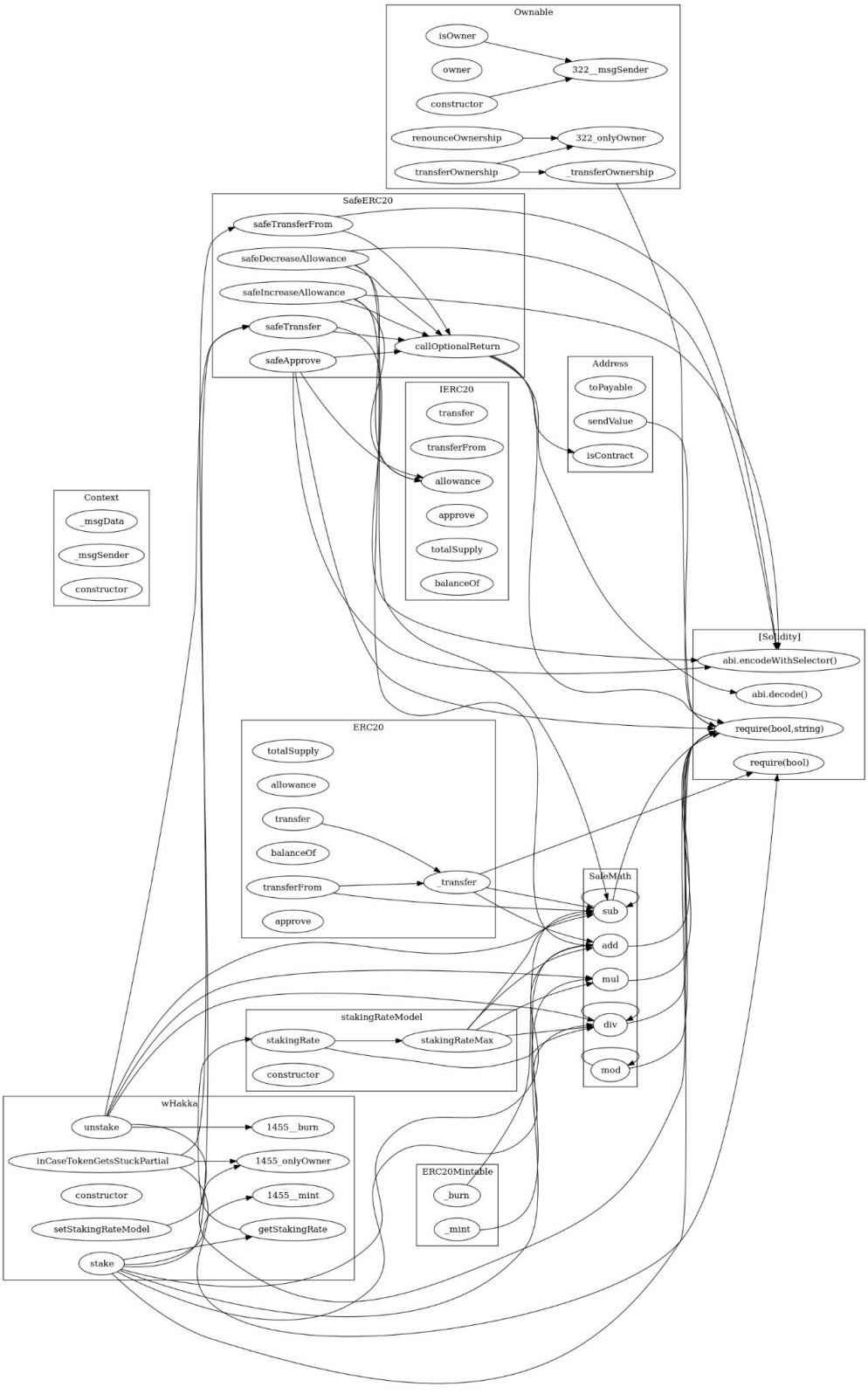
wHakka.unstake(address,uint256,uint256) (wHakka.sol#574-590) uses timestamp for comparisons
Dangerous comparisons:
- require(bool,string)(block.timestamp >= v.unlockTime,locked) (wHakka.sol#576)

Medium

Reentrancy no ETH:

Reentrancy in wHakka.stake(address,uint256,uint256) (wHakka.sol#555-572):
External calls:
- wAmount = getStakingRate(time).mul(amount).div(1e18) (wHakka.sol#557)
- currentModel.stakingRate(time) (wHakka.sol#548)
State variables written after the call(s):
- vaultCount[to] ++ (wHakka.sol#566)
- v.hakkaAmount = amount (wHakka.sol#560)
- v.wAmount = wAmount (wHakka.sol#561)
- v.unlockTime = block.timestamp.add(time) (wHakka.sol#562)

Contract Flow



Audit Team

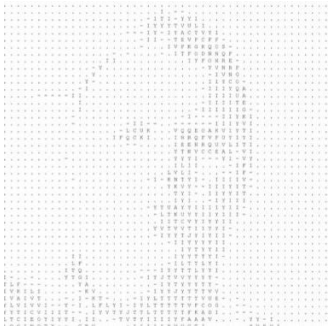
Team Lead: Eelko Neven

Eelko has been in the it/security space since 1991. His passion started when he was confronted with a formatted hard drive and no tools to undo it. At that point he started reading a lot of material on how computers work and how to make them work for others. After struggling for a few weeks he finally wrote his first HD data recovery program. Ever since then when he was faced with a challenge he just persisted until he had a solution.

This mindset helped him tremendously in the security space. He found several vulnerabilities in large corporation servers and notified these corporations in a responsible manner. Among those are Google, Twitter, General Electrics etc.

For the last 12 years he has been working as a professional security /code auditor and performed over 1500 security audits / code reviews, he also wrote a similar amount of reports.

He has extensive knowledge of the Solidity programming language and this is why he loves to do Defi and other smartcontract reviews.



Email:
info@coinbae.com

Disclaimer

Coinbae audit is not a security warranty, investment advice, or an endorsement of Hakka Finance. This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the Hakka Finance team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.