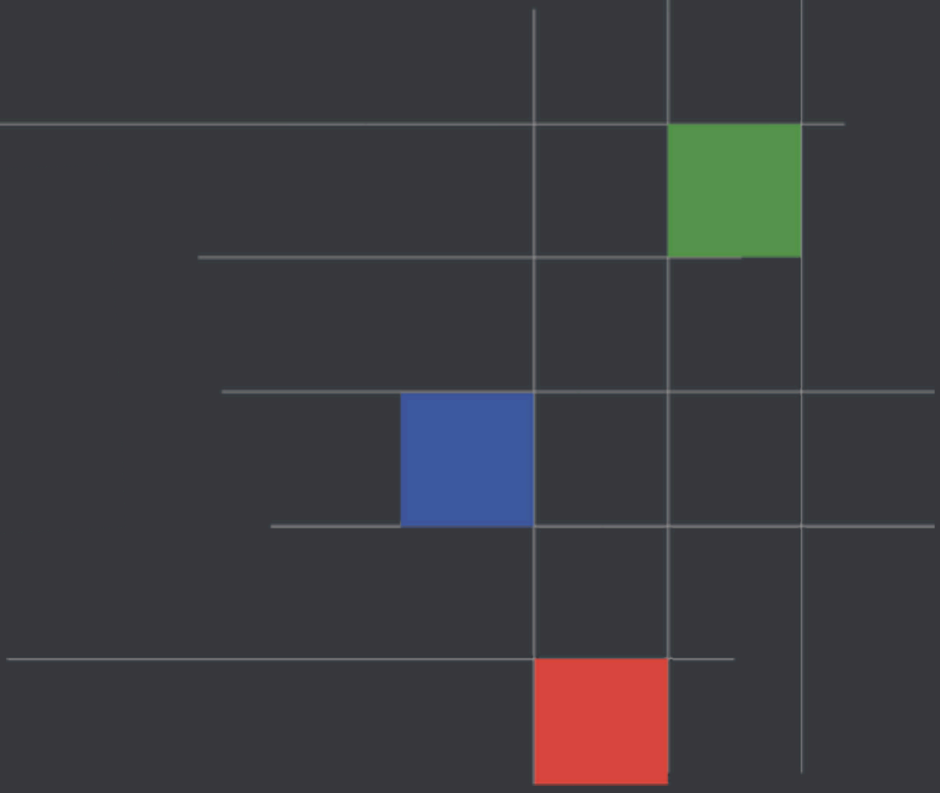
Wault Protocol

Security Assessment

March 19th, 2021

For :
WAULT Protocol



Disclaimer

CertiK reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts CertiK to perform a security review.

CertiK Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK’s position is that each company and individual are responsible for their own due diligence and continuous security. CertiK’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has indeed completed a round of auditing with the intention to increase the quality of the company/product’s IT infrastructure and or source code.

Overview

Project Summary

Project Name	Wault Finance
Description	Decentralized Finance Protocol
Platform	Ethereum; Solidity
Codebase	GitHub Repository
Commits	a03495d1dbff9f5708b738ac8ebfae1fe001998e e41853d137867149acb547b1369dad73c94423fa

Audit Summary

Delivery Date	Mar. 15th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Feb. 24, 2021 - Mar. 12, 2021, Mar. 15, 2021, Mar. 19, 2021

Vulnerability Summary

Total Issues	4
Total Critical	0
Total Major	1
Total Minor	0
Total Informational	3

Executive Summary

This report has been prepared for **Wault Finance** smart contract to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Dynamic Analysis, Static Analysis, and Manual Review techniques.

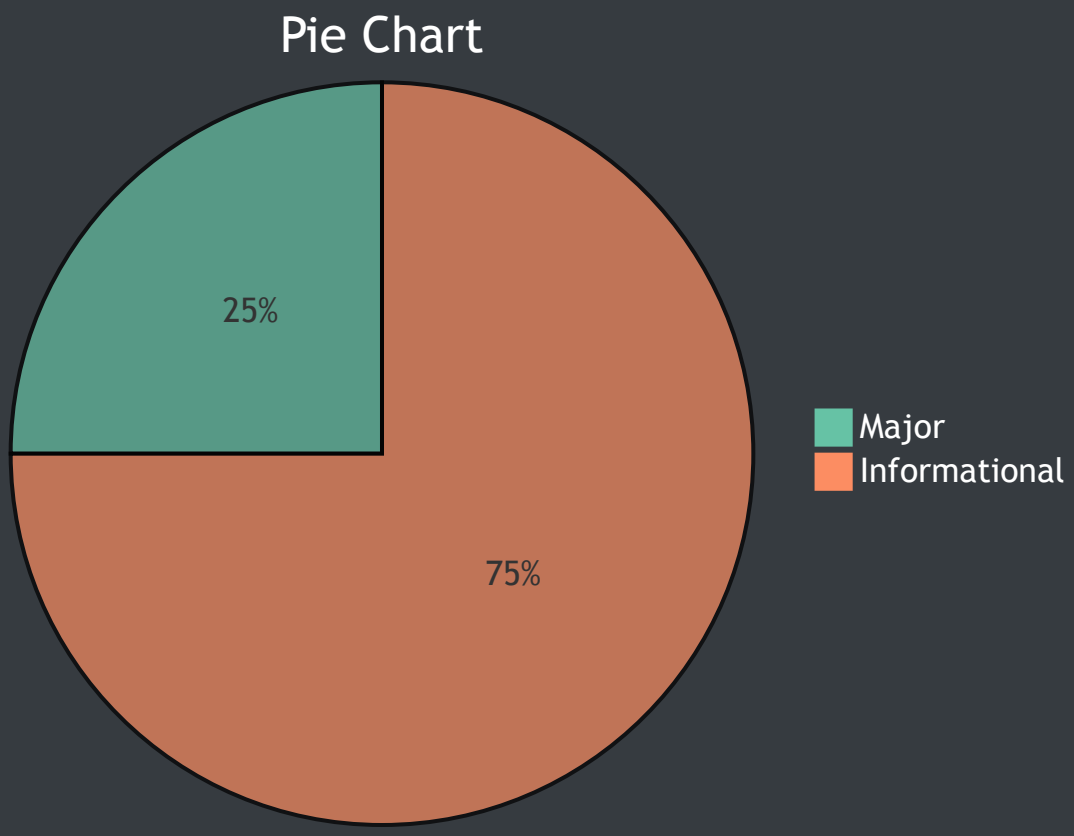
The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

File in Scope

ID	Contract	SHA256-Checksum
WL	**WaultLocker.sol**	2c1750b3e474674a7a9011cee528bdf7e8346119814b892784040fcf10ee698b

Findings



ID	Title	Type	Severity	Resolved
WL-01	Incorrect Token Transfer Strategy	Logical Issue	Major	✓
WL-02	Rounding Error	Mathematical Operations	Informational	!
WL-03	Variable Naming Convention	Coding Style	Informational	!
WL-04	Optimization on Sensitive Action	Optimization	Informational	!

WL-01: Variable

Type	Severity	Location
Logical Issue	Major	WaultLocker.sol : L42, L45, L83, L86,L107

Description:

For some specific tokens, such as USDT Token, the transfer or transferFrom implementation does not meet the EIP20 standard. The `IERC20(token).transferFrom()` and `IERC20(token).transfer()` could fail under certain conditions.

Check the `lockTokens()` function:

```
function lockTokens(IERC20 _token, address _withdrawer, uint256 _amount, uint256 _unlockTimestamp) external returns (uint256 _id) {
    ......
    require(_token.transferFrom(msg.sender, address(this), _amount), 'Transfer of tokens failed!');
    ......
    require(_token.transfer(waultMarkingAddress, tax), 'Taxing failed!');
    ......
}

```

The `customLockTokens()` function:

```
function customLockTokens(uint256 _id) external {
    ......
    require(lockedToken[_id].token.transferFrom(msg.sender, address(this), lockedToken[_id].amount), 'Transfer of tokens failed!');
    ......
    require(lockedToken[_id].token.transfer(waultMarkingAddress, tax), 'Taxing failed!');
    ......
    }
```

And the `withdrawTokens()` function:

```
function withdrawTokens(uint256 _id) external {
    ......
    require(lockedToken[_id].token.transfer(msg.sender, lockedToken[_id].amount), 'Transfer of tokens failed!');
    }
```

Recommendation:

Using the `SafeERC20` library instead of ERC20, check SafeERC20.sol in OpenZeppelin library.

Examples:

```
pragma solidity 0.7.6;

import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v3.4/contracts/token/ERC20/SafeERC20.sol";

contract WaultLocker is Ownable{
    using SafeERC20 for IERC20;

    ......

    function lockTokens(IERC20 _token, address _withdrawer, uint256 _amount, uint256 _unlockTimestamp) external returns (uint256 _id) {
        ......
        require(_token.safeTransferFrom(msg.sender, address(this), _amount), 'Transfer of tokens failed!');
        ......
        require(_token.safeTransfer(waultMarkingAddress, tax), 'Taxing failed!');
        ......
    }

    ......

    function customLockTokens(uint256 _id) external {
        ......
        require(lockedToken[_id].token.safeTransferFrom(msg.sender, address(this), lockedToken[_id].amount), 'Transfer of tokens failed!');
        ......
        require(lockedToken[_id].token.safeTransfer(waultMarkingAddress, tax), 'Taxing failed!');
        ......
    }

    ......

    function withdrawTokens(uint256 _id) external {
        ......
        require(lockedToken[_id].token.safeTransfer(msg.sender, lockedToken[_id].amount), 'Transfer of tokens failed!');
```

```
33      }
34
35      ......
36  }
```

Alleviation:

[Wault Finance]: The team addressed the issue in the commit e41853d137867149acb547b1369dad73c94423fa

WL-02: Rounding Error

Type	Severity	Location
Rounding Error	Informational	WaultLocker.sol : L38

Description:

When the `_amount` value is too small, the following code will have a large error due to the rounding error.

```
function lockTokens(IERC20 _token, address _withdrawer, uint256 _amount, uint256 _unlockTimestamp) external returns (uint256 _id) {
    require(_amount > 500, 'Token amount too low!');
    .......
    uint256 tax = _amount.mul(taxPermille).div(1000);
    ......
    }
```

This issue could be ignored in general application scenarios. However, it is still recommended to add some simple checks to avoid this problem.

Recommendation:

Considering rounding error, an example of the improved code is shown below.:

```
function lockTokens(IERC20 _token, address _withdrawer, uint256 _amount, uint256 _unlockTimestamp) external returns (uint256 _id) {
    require(_amount >= 500 && _amount % 500 == 0, 'Token amount too low!');
    ......

    uint256 tax = _amount.mul(taxPermille).div(1000);
    ......
}
```

Alleviation:

N/A

WL-03: Variable Naming Convention

Type	Severity	Location
Coding Style	Informational	WaultLocker.sol L28

Description:

The following variable do not conform to the standard naming convention of Solidity whereby constant variable names utilize the UPPER_CASE format.

```
1      string  public symbol       = "TEST";
2      string  public name         = "Test Token";
3      uint8   public decimals     = 18;
4      uint    public _totalSupply = 150e18;
```

Alleviation:

N/A

WL-04: Optimization on Sensitive Action

Type	Severity	Location
Optimization	Informational	WaultLocker.sol L110:112

Description:

Function `setWaultMarkingAddress` is missing a require for input parameter. And the parameter do not conform to the naming convention of this contract. And we advise that add `emit event` for this sensitive action.

```
1  function setWaultMarkingAddress(address _waultMarkingAddress) external onlyOwner {
2          waultMarkingAddress = _waultMarkingAddress;
3      }
```

Recommendation:

We advise that the naming conventions utilized by the statements are adjusted to reflect the correct type of declaration according to the solidity style guide.

Alleviation:

N/A

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how `block.timestamp` works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a `struct` assignment operation affecting an in-memory `struct` rather than an instorage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete` .

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a `constructor` assignment imposing different `require` statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as `constant` contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified

version of the project.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.

Icons explanation

 : Issue resolved

 : Issue not resolved / Acknowledged. The team will be fixing the issues in the own timeframe.

 : Issue partially resolved. Not all instances of an issue was resolved.