CERTIK

Security Assessment

Onekey

Jun 21st, 2021

Table of Contents

Summary

This report has been prepared for Onekey smart contracts, 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 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.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases given they are currently missing in the repository;
- Provide more comments per each function for readability, especially contracts are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	Onekey
Platform	BSC
Language	Solidity
Codebase	https://github.com/OneKeyHQ/onekey-nft
Commit	a3978f392eee447a44105db99bfa28d7b775ffdf 4f75fabd14112d18ac734c2e0e5c0d1f5e5da217

Audit Summary

Delivery Date	Jun 21, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	

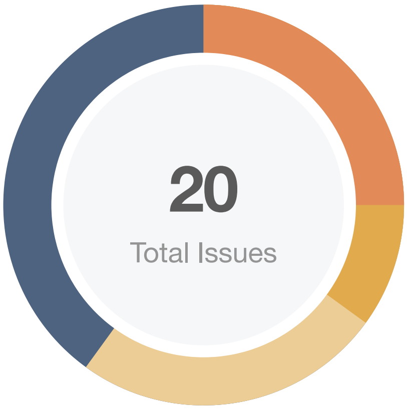
Vulnerability Summary

Total Issues	20
● Critical	0
● Major	5
● Medium	2
● Minor	5
● Informational	8
● Discussion	0

Audit Scope

ID	file	SHA256 Checksum
AVC	AirdropVault.sol	39618e436b2550764d62fd7f9ad0c8c38c971392bfe1fd16a259b0ea8ccd3238
CCK	Crowdfunding.sol	aa97d7a5ab64a3757d8d034a387d5d841f9c9d294106bfd53f0a76dc46b770f0
HVC	HolderVault.sol	2be2f31561decab538c2dcf117f9308ce7c2e43b48bee75f307016ff1287421e
OTC	OnekeyToken.sol	84f78aa800da0365fa8d8976cb046ff40b14d476193cbad2a92fd0a7558291e1
RMC	RoundManager.sol	532d277b47fb2aab5a756b2dbaf85f1f1f9409fc3dbaa81472c96aae3189e42c
OCK	libraries/Ownable.sol	b857e3276c046f6769a05e6acb84d14b696f63d0a99f43fd4696967f39511cb4
SMC	libraries/SafeMath.sol	036fcff7adc78867dbc757758c2dea7b71a5a10f1aca069a1e833e2f016133bb
THC	libraries/TransferHelper.sol	369a92ec54d78eb988b726e3a8d814267806d707bbe1aa7c1dff49d295279a80

Findings



🟥 **Critical**	**0** (0.00%)
🟧 **Major**	**5** (25.00%)
🟨 **Medium**	**2** (10.00%)
▨ **Minor**	**5** (25.00%)
🟦 **Informational**	**8** (40.00%)
🟩 **Discussion**	**0** (0.00%)

ID	Title	Category	Severity	Status
AVC-01	Lack of Input Validation	Volatile Code	● Informational	✓ Resolved
AVC-02	Unused Variable	Gas Optimization	● Informational	✓ Resolved
AVC-03	Check-effect-interaction Pattern Violation	Logical Issue	● Medium	✓ Resolved
AVC-04	Centralized Risk	**Centralization / Privilege**	● **Major**	✓ **Resolved**
AVC-05	Potentially Manipulated Lucky Numbers	**Centralization / Privilege**	● **Major**	✓ **Resolved**
CCK-01	Centralized Risk	**Centralization / Privilege**	● **Medium**	✓ **Resolved**
CCK-02	Unknown Implementation of `balanceOf` Function	**Centralization / Privilege**	● **Minor**	✓ **Resolved**
CCK-03	Unknown Implementation of `addOrder` Function	**Centralization / Privilege**	● **Minor**	✓ **Resolved**
CCK-04	Proper Usage of `require` and `assert` Functions	Coding Style	● Informational	✓ Resolved
CCK-05	Lack of Input Validation	Volatile Code	● Informational	✓ Resolved
CCK-06	Centralized Risk	**Centralization / Privilege**	● **Major**	✓ **Resolved**
CCK-07	Typo `refferal`	Coding Style	● Informational	✓ Resolved

ID	Title	Category	Severity	Status
CCK-08	Lack of Input Validation	Logical Issue	● Informational	✓ Resolved
CCK-09	Lack of Input Validation	Logical Issue	● Informational	✓ Resolved
HVC-01	Claiming Rewards On Behalf Of Another User	Logical Issue	● Minor	✓ Resolved
HVC-02	Lack of Input Validation	Volatile Code	● Informational	✓ Resolved
OTC-01	Costly Loop	Gas Optimization	● Minor	✓ Resolved
OTC-02	Centralized Risk	**Centralization / Privilege**	● **Major**	✓ **Resolved**
RMC-01	Default Value Used For Target Token	Volatile Code	● Major	✓ Resolved
RMC-02	`finalRoundEndAt` Not Used	Logical Issue	● Minor	✓ Resolved

AVC-01 | Lack of Input Validation

Category	Severity	Location	Status
Volatile Code	● Informational	AirdropVault.sol: 44~50	✓ Resolved

Description

The assigned values to `foundingContract` and `targetToken` in the constructor of `AirdropVault.sol` should be verified as non-zero values to prevent errors.

Recommendation

Check that the passed-in values are non-zero. Example:

```
require(_foundingContract != address(0), "_foundingContract is a zero address");
require(_targetToken != address(0), "_targetToken is a zero address");
```

Alleviation

`[Onekey]` The client heeded our advice and added checks that the passed-in values are non-zero in the latest commit:4f75fabd14112d18ac734c2e0e5c0d1f5e5da217

AVC-02 | Unused Variable

Category	Severity	Location	Status
Gas Optimization	● Informational	AirdropVault.sol: 16	✓ Resolved

Description

The state variable `ROLL_IN_PROGRESS` in `AirdropVault.sol` is not used.

Recommendation

We advise the client to consider removing the variable `ROLL_IN_PROGRESS`.

Alleviation

`[Onekey]` The client heeded our advice and removed unused variable `ROLL_IN_PROGRESS` in the latest commit:4f75fabd14112d18ac734c2e0e5c0d1f5e5da217

AVC-03 | Check-effect-interaction Pattern Violation

Category	Severity	Location	Status
Logical Issue	● Medium	AirdropVault.sol: 124~130	✓ Resolved

Description

`rewardClaimed[_round]` is updated after `TransferHelper.safeTransfer`, which violates the check-effect-interaction pattern.

Recommendation

We advise the client to revise the function `claimAirdrop` by rewriting the statements from L124 to L130 as follows:

```
rewardClaimed[_round] = false;

TransferHelper.safeTransfer(
        targetToken,
        msg.sender,
        rewardAmount[_round]
);
```

Alleviation

`[Onekey]` The client heeded our advice and changed claimed statue before token transfer to avoid the check-effect-interaction in the latest commit:4f75fabd14112d18ac734c2e0e5c0d1f5e5da217

AVC-04 | Centralized Risk

Category	Severity	Location	Status
Centralization / Privilege	● **Major**	AirdropVault.sol: 101	✓ **Resolved**

Description

In function `withdrawLINK`, the owner of the contract `owner` could transfer `_value` amount of token to an arbitrary address `_to`.

Recommendation

We advise the client to carefully manage the `owner` account's private key and avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or via smart-contract based accounts with enhanced security practices, f.e. Multisignature wallets.

Indicatively, here are some feasible solutions that would also mitigate the potential risk:

- Time-lock with reasonable latency, i.e. 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent single point of failure due to the private key;
- Introduction of a DAO / governance / voting module to increase transparency and user involvement.

Alleviation

`[Onekey]` `withdrawLINK` function used to claim unused LINK token. Cause request random numbers from Chainlink, and the contract will spend some LINK token. So that contract needs have some LINK tokens. But when the crowdfunding ends, we can claim unused LINK tokens back.

AVC-05 | Potentially Manipulated Lucky Numbers

Category	Severity	Location	Status
Centralization / Privilege	● Major	AirdropVault.sol: 86, 105, 144, 152	✓ Resolved

Description

The function `claimAirdrop` on L105 check if a user should be rewarded by referring to the current round's lucky number derived from `luckyNumberList` and registered numbers for the user derived from `userInfo`. While on L86, the contract has the privilege to add a new lucky number to `luckyNumberList` by invoking the function `fulfillRandomness`. And this lucky number could be manipulated by setting the variable `_randomness`. Also, the function `getLuckyNumbers` on L144 returns registered numbers for a user derived from `userInfo`, and the function `getRoundLuckyNumbers` on L152 returns the current round's lucky number derived from `luckyNumberList`.

Recommendation

We advise the client to check if the contract should have the privilege to append to `luckyNumberList` in the way described in the function `fulfillRandomness` and if the accesses for these aforementioned functions are configured correctly.

Alleviation

`[Onekey]` Based on the Chainlink VRFConsumerBase contract, only VRFCoordinator can fulfill the random number. And fulfillRandomness is an internal function, only rawFulfillRandomness function in VRFConsumerBase used. We assume the Chainlink project is reliable, and we have got in touch with the Chainlink team to make sure this function work properly.

CCK-01 | Centralized Risk

Category	Severity	Location	Status
Centralization / Privilege	● Medium	Crowdfunding.sol: 82~100	✓ Resolved

Description

The owner of the contract `owner` has the privilege to change the values of `holderContract`, `airdropContract`, and `roundContract`. And these variables are used to decide the target addresses of transferring in function `_deliverReward`.

Recommendation

We advise the client to carefully manage the `owner` account's private key and avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or via smart-contract based accounts with enhanced security practices, f.e. Multisignature wallets.

Indicatively, here are some feasible solutions that would also mitigate the potential risk:

- Time-lock with reasonable latency, i.e. 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent single point of failure due to the private key;
- Introduction of a DAO / governance / voting module to increase transparency and user involvement.

Alleviation

`[Onekey]` Add Time-lock with reasonable latency. Use openzeppeline `TimelockController` contracts.

`[Onekey]` `Crowdfunding` contract has been deployed at 0x98DeafE487DcD6DEd695B1bFBCA907B7ef66367f and its's ownership has been transferred to Timelock deployment with 12 hours delay at 0x9Be2fF9aD9aB148E9A0c9FC42A49753D430f7b8F through transaction 0xe5d8823a7c5440635d33dd4cc92353db0e89aff046c0fb7348166a90480c2ae2

CCK-02 | Unknown Implementation of `balanceOf` Function

Category	Severity	Location	Status
Centralization / Privilege	● Minor	Crowdfunding.sol: 157, 201	✓ Resolved

Description

On L157 and L201, `IERC20(targetAssest)` can be any contract address where the `IERC20` interface is implemented. As a result, the invocations of `IERC20(targetAssest).balanceOf(address(this));` in function `buyWallet` may bring dangerous effects as the implementation is unknown to the user.

Recommendation

We advise the client to restrict the group of users who can access to `buyWallet` function and check and ensure the contract specified by `IERC20(targetAssest)` is a standard smart contract that follows the `IERC20` interface with correct logic implementation as designed in the project repository.

Alleviation

`[Onekey]` Crowdfunding will set USDT as `targetAssest`, so we assume USDT contract is safe. And `targetAssest` has the immutable attribute so it will never be changed.

CCK-03 | Unknown Implementation of `addOrder` Function

Category	Severity	Location	Status
Centralization / Privilege	● Minor	Crowdfunding.sol: 240	✓ Resolved

Description

On L240, `IHolderVault(holderContract)` can be any contract address where the `IHolderVault` interface is implemented. As a result, the invocation of `IHolderVault(holderContract).addOrder` in function `_deliverReward` may bring dangerous effects as the implementation is unknown to the user.

Recommendation

We advise the client to restrict the group of users who can access to `_deliverReward` function and check and ensure the contract specified by `IHolderVault(holderContract)` is a standard smart contract that follows the `IHolderVault` interface with correct logic implementation as designed in the project repository.

Alleviation

`[Onekey]` `IHolderVault` is the interface of the HolderVault contract, it will deploy by ourselves, and we will guarantee the logic implementation are correct. Also, OneKey's contracts will be open source. In the meantime will be verified on bscscan.

CCK-04 | Proper Usage of `require` and `assert` Functions

Category	Severity	Location	Status
Coding Style	● Informational	Crowdfunding.sol: 78	✓ Resolved

Description

The `assert` function should only be used to test for internal errors, and to check invariants. The `require` function should be used to ensure valid conditions, such as validation of inputs, state variables, and return values.

Recommendation

Consider using the `require` function, along with a custom error message when the condition fails, instead of the `assert` function.

Alleviation

`[Onekey]` The client heeded our advice and replace the `assert` with `require` in the latest commit:4f75fabd14112d18ac734c2e0e5c0d1f5e5da217

CCK-05 | Lack of Input Validation

Category	Severity	Location	Status
Volatile Code	● Informational	Crowdfunding.sol: 67~75	✓ Resolved

Description

The assigned values to `onekeyToken`, `WETH`, and `targetAssest` in the constructor of the contract `Crowdfunding` should be verified as non-zero values to prevent errors.

Recommendation

Check that the passed-in values are non-zero. Example:

```
require(_onekeyToken != address(0), "_onekeyToken is a zero address");
require(_WETH != address(0), "_WETH is a zero address");
require(_targetAssest != address(0), "_targetAssest is a zero address");
```

Alleviation

`[Onekey]` The client heeded our advice and added the input validators in the constructor of the contract in the latest commit:4f75fabd14112d18ac734c2e0e5c0d1f5e5da217

CCK-06 | Centralized Risk

Category	Severity	Location	Status
Centralization / Privilege	● **Major**	Crowdfunding.sol: 113~118	✓ **Resolved**

Description

In function `updateWallets`, the owner of the contract `owner` has the privilege to update the state variable `wallets`. And `wallets` is used in buying wallets in the function `buyWallet` on L121 and delivering rewards in the function `_deliverReward` on L219.

Recommendation

We advise the client to carefully manage the `owner` account's private key and avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or via smart-contract based accounts with enhanced security practices, f.e. Multisignature wallets.

Indicatively, here are some feasible solutions that would also mitigate the potential risk:

- Time-lock with reasonable latency, i.e. 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent single point of failure due to the private key;
- Introduction of a DAO / governance / voting module to increase transparency and user involvement.

Alleviation

`[Onekey]` Add Time-lock with reasonable latency. Use openzeppeline `TimelockController` contracts.

`[Onekey]` `Crowdfunding` contract has been deployed at 0x98DeafE487DcD6DEd695B1bFBCA907B7ef66367f and its's ownership has been transferred to Timelock deployment with 12 hours delay at 0x9Be2fF9aD9aB148E9A0c9FC42A49753D430f7b8F through transaction 0xe5d8823a7c5440635d33dd4cc92353db0e89aff046c0fb7348166a90480c2ae2

CCK-07 | Typo `refferal`

Category	Severity	Location	Status
Coding Style	● Informational	Crowdfunding.sol: 1	✓ Resolved

Description

The word `refferal` is used across the file `Crowdfunding.sol`.

Recommendation

We advise the client to consider renaming `refferal` to `referral` to avoid confusion.

Alleviation

`[Onekey]` The client heeded our advice and correct the typo in the latest commit:4f75fabd14112d18ac734c2e0e5c0d1f5e5da217

CCK-08 | Lack of Input Validation

Category	Severity	Location	Status
Logical Issue	● Informational	Crowdfunding.sol: 139~144	✓ Resolved

Description

In function `buyWallet`, the user will fail to buy wallets if `_sellToken` is ether. Because the contract calls `safeTransferFrom` directly without checking `_sellToken` is ether or not.

Recommendation

We advise the client to handle the case when `_sellToken` is ether separately.

```
if (_sellToken == WETH) {
    ...
} else {
    TransferHelper.safeTransferFrom(
        _sellToken,
        msg.sender,
        address(this),
        _sellAmount
   );
}
```

Alleviation

`[Onekey]` The client fixed this issue by updating the function `buyWallet` with following snippet in the latest commit:4f75fabd14112d18ac734c2e0e5c0d1f5e5da217

```
if (_sellToken == ETH) {
    ....
} else {
    ....
    TransferHelper.safeTransferFrom(
        _sellToken,
        msg.sender,
        address(this),
        _sellAmount
    );
}
```

CCK-09 | Lack of Input Validation

Category	Severity	Location	Status
Logical Issue	● Informational	Crowdfunding.sol: 212	✓ Resolved

Description

In function `_fillQuote`, the call to `safeApprove` will fail if `_sellToken` is ether.

Recommendation

We advise the client to add a check for `_sellToken`.

```
if (_sellToken == WETH) {
    ...
} else {
    TransferHelper.safeApprove(_sellToken, _spender, _sellAmount);
}
```

Alleviation

`[Onekey]` The client fixed this issue by updating the function `_fillQuote()` with the following snippet in the latest commit:4f75fabd14112d18ac734c2e0e5c0d1f5e5da217

```
if (_sellToken != ETH)
    TransferHelper.safeApprove(_sellToken, _spender, _sellAmount);
```

HVC-01 | Claiming Rewards On Behalf Of Another User

Category	Severity	Location	Status
Logical Issue	● Minor	HolderVault.sol: 47~59	✓ Resolved

Description

In function `claim`, the rewards is sent to the address `_user`, and this address could be different from `msg.sender`.

Recommendation

We advise the client to consider adding a requirement ensures that any user should only claim his/her own reward. Example:

```
require(_user == msg.sender, "claiming rewards for a user other than msg.sender");
```

Alleviation

`[Onekey]` The client heeded our advice and added the user check in function `claim()` in the latest commit:4f75fabd14112d18ac734c2e0e5c0d1f5e5da217

```
require(_user == msg.sender, "SHOULD_CLAIM_BY_THEMSELVES");
```

HVC-02 | Lack of Input Validation

Category	Severity	Location	Status
Volatile Code	● Informational	HolderVault.sol: 40~44	✓ Resolved

Description

The assigned values to `targetToken` and `foundingContract` in the constructor of the contract `HolderVault` should be verified as non-zero values to prevent errors.

Recommendation

Check that the passed-in values are non-zero. Example:

```
require(_targetToken != address(0), "_targetToken is a zero address");
require(_foundingContract != address(0), "_foundingContract is a zero address");
```

Alleviation

`[Onekey]` The client heeded our advice and added the input validators in the constructor of the contract in the latest commit:4f75fabd14112d18ac734c2e0e5c0d1f5e5da217

OTC-01 | Costly Loop

Category	Severity	Location	Status
Gas Optimization	● Minor	OnekeyToken.sol: 82~88	✓ Resolved

Description

The storage variable `totalMinted` is accessed in each iteration of the loop from L82 to L88. This operation could be costly in terms of gas consumption.

Recommendation

We advise the client to consider using a local variable to hold the intermediate result. Example:

```
uint256 tmp = totalMinted;
for (uint256 i = 0; i < _amount; i++) {
    if (_id == 0) user.mini.push(tmp);
    else if (_id == 1) user.touch.push(tmp);
    else if (_id == 2) user.pro.push(tmp);
    tmp += 1;
}
totalMinted = tmp;
```

As the cost is largely dependent on storage accesses, the original implementation should have 4 storage reads and 1 storage write in each iteration. In the fixed version shown above, there should be 1 storage read and 1 storage write in the above code snippet.

Alleviation

`[Onekey]` The client heeded our advice and used memory variable temp to reduce gas consumption in the latest commit:4f75fabd14112d18ac734c2e0e5c0d1f5e5da217

OTC-02 | Centralized Risk

Category	Severity	Location	Status
Centralization / Privilege	● **Major**	OnekeyToken.sol: 37, 78	✓ **Resolved**

Description

In function `mint`, the minter of the contract `MINTER_ROLE` could mint `_amount` amount of token to an arbitrary address `_account`.

Recommendation

We advise the client to carefully manage the `MINTER_ROLE` account's private key and avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or via smart-contract based accounts with enhanced security practices, f.e. Multisignature wallets.

Indicatively, here are some feasible solutions that would also mitigate the potential risk:

- Time-lock with reasonable latency, i.e. 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent single point of failure due to the private key;
- Introduction of a DAO / governance / voting module to increase transparency and user involvement.

Alleviation

`[Onekey]` Add Time-lock with reasonable latency. Use openzeppeline `TimelockController` contracts.

`[Onekey]` `OnekeyToken` contract has been deployed at 0xAa25850bb317dA4B5d1CC2B45C0a9F6263faB4db and deployer's `MINTER_ROLE` has been revoked through transaction 0x0a04b75acfa7deda6e9a6e4460dd8ddd93243df96808464db5d988f823786aae

Moreover, `DEFAULT_ADMIN_ROLE` has been granted to Timelock deployment with 12 hours delay at 0x9Be2fF9aD9aB148E9A0c9FC42A49753D430f7b8F through transaction 0xceba99146ebaf2379fce905ab94b60fd5fa475e8cf77e1d314131ee85c6da3e3, and deployer's `DEFAULT_ADMIN_ROLE` has been revoked through transaction 0xb820aa7c7ee378e857a67510b88385d6cfbf84c255c4fe0975662156f4a81868

RMC-01 | Default Value Used For Target Token

Category	Severity	Location	Status
Volatile Code	● Major	RoundManager.sol: 37, 150	✓ Resolved

Description

The state variable `targetToken` is declared on L37, and it will have an all-zero byte-representation as its default value. Since there is no write to `targetToken` in the contract, this default value will be used for transferring on L150, which may lead to unexpected results.

Recommendation

We advise the client to check if the usage of `targetToken` on L150 is correct.

Alleviation

`[Onekey]` Set `targetAssest` value in the constructor. And change `targetToken` to `targetAsset`, the same name in other contacts.

`[Onekey]` The client heeded the advice and fixed the issue in the latest commit:4f75fabd14112d18ac734c2e0e5c0d1f5e5da217

RMC-02 | `finalRoundEndAt` Not Used

Category	Severity	Location	Status
Logical Issue	● Minor	RoundManager.sol: 31	✓ Resolved

Description

In `RoundManager.sol`, the state variable `finalRoundEndAt` is initialized but not used.

Recommendation

We advise the client to check if the following require statement is needed at the beginning of the function `updateRoundTime`.

```
require(block.number <= finalRoundEndAt, "ALL_ROUND_IS_OVER");
```

Alleviation

`[Onekey]` The client fixed the bug by adding following check in the latest commit:4f75fabd14112d18ac734c2e0e5c0d1f5e5da217

```
require(block.number <= finalRoundEndAt, "ALL_ROUND_IS_OVER");
```

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Gas Optimization

Gas Optimization findings 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.

Logical Issue

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

Volatile Code

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

Coding Style

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

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK's prior written consent.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is 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 assessment. This report does 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.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing 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.

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we’re able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.