# Security Assessment

# Defrost Finance

Nov 10th, 2021

# Table of Contents

# About

# Summary

This report has been prepared for Defrost Finance to discover issues and vulnerabilities in the source code of the Defrost Finance project 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;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

# Overview

## Project Summary

| | |
|---|---|
| Project Name | Defrost Finance |
| Platform | Avalanche C-chain |
| Language | Solidity |
| Codebase | https://github.com/DefrostFinance/defrost-finance-contract/tree/master/contracts/collateralVault<br>https://github.com/DefrostFinance/defrost-finance-contract/tree/master/contracts/systemCoin<br>https://github.com/DefrostFinance/defrost-finance-contract/blob/master/contracts/interestEngine/interestEngine.sol<br>https://github.com/DefrostFinance/defrost-finance-contract/blob/master/contracts/defrostFactory |
| Commit | 0b89aa6a7fd976145c2dee25c27d3f811acba9f5 |

## Audit Summary

| | |
|---|---|
| Delivery Date | Nov 10, 2021 |
| Audit Methodology | Static Analysis, Manual Review |
| Key Components | |

# Vulnerability Summary

| Vulnerability Level | Total | ⓘ Pending | ⊗ Declined | ⓘ Acknowledged | ⓘ Partially Resolved | ✓ Resolved |
|---|---|---|---|---|---|---|
| ● Critical | 1 | 0 | 0 | 0 | 0 | 1 |
| ● Major | 5 | 0 | 0 | 5 | 0 | 0 |
| ● Medium | 0 | 0 | 0 | 0 | 0 | 0 |
| ● Minor | 6 | 0 | 0 | 1 | 0 | 5 |
| ● Informational | 12 | 0 | 0 | 7 | 0 | 5 |
| ● Discussion | 0 | 0 | 0 | 0 | 0 | 0 |

## Audit Scope

| ID | File | SHA256 Checksum |
|---|---|---|
| VVD | collateralVault/collateralVault.sol | 031331248648ccd26aa50defd2f636243379f18660a5a0c226ef1d88f7e08360 |
| EVD | collateralVault/vaultEngine.sol | f6c310124f3038265b7543ab4d543bb576e4bea272203130e1d8826ad30f73ab |
| EDV | collateralVault/vaultEngineData.sol | f0b7c28c0b366b7a0dcf729c292e7be3ca2ea88d960cac82db2363b67e545708 |
| FFD | defrostFactory/defrostFactory.sol | 1d74bc60aff4d4d607c86a7a05685c342d63b885a357318cd2dea1596462221e |
| FDD | defrostFactory/defrostFactoryData.sol | 22f2d82799093e31c8be365a576ee33f339c539ed730f0d8d035ba885db8ab36 |
| EED | interestEngine/interestEngine.sol | 744521ba13fc15c502ba7ec20039fe49d494932e7f73f63b94d5d4a422c43e0e |
| CCD | systemCoin/Coin.sol | e169f22b8298eafc8b4fb76b7db7f6022209e59fe596665c4e4855c0a4aaee40 |

It should be noted that the system design includes a number of economic arguments and assumptions. These were explored to the extent that they clarified the intention of the code base, but we did not audit the mechanism design itself. Note that financial models of blockchain protocols need to be resilient to attacks. It needs to pass simulations and verifications to guarantee the security of the overall protocol. The correctness of the financial model is not in the scope of the audit.

Note that this audit only includes the contracts in the stated scope while the files outside the scope are treated as black boxes and are assumed to be functionally correct.

To bridge the trust gap between owner and users, the owner needs to express a sincere attitude with the consideration of the administrator team's anonymousness.

The owner has the responsibility to notify users with the following capability of the administrator:

- init contract through `initContract()`

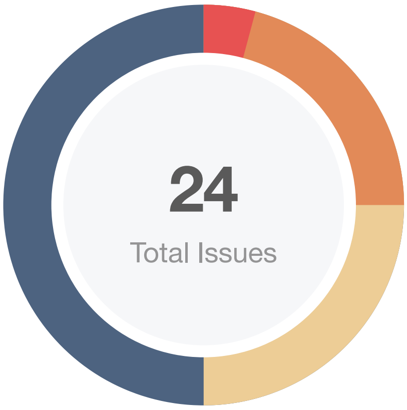
The origin has the responsibility to notify users with the following capability of the administrator:

- set emergency start time through `setEmergency()`
- set liquidation info through `setLiquidationInfo()`
- set pool asset limitation through `setPoolLimitation()`
- set stability fee through `setStabilityFee()`
- create vault through `createVault()`
- create system coins through `createSystemCoin()`

The authorized has the responsibility to notify users with the following capability of the administrator:

- maliciously add auth to an account through `addAuthorization()`
- maliciously remove auth from an account through `removeAuthorization()`
- maliciously mint new coins through `mint()`

# Findings



| | |
|---|---|
| 🟥 **Critical** | **1** (4.17%) |
| 🟧 **Major** | **5** (20.83%) |
| 🟨 **Medium** | **0** (0.00%) |
| 🟨 **Minor** | **6** (25.00%) |
| 🟦 **Informational** | **12** (50.00%) |
| 🟩 **Discussion** | **0** (0.00%) |

| ID | Title | Category | Severity | Status |
|---|---|---|---|---|
| GLOBAL-01 | Unlocked Compiler Version | Language Specific | ● Informational | ⓘ Acknowledged |
| CCD-01 | Lack of Zero Address Validation | Volatile Code | ● Minor | ✓ Resolved |
| CCD-02 | Function Visibility Optimization | Gas Optimization | ● Informational | ✓ Resolved |
| CCD-03 | Incorrect naming convention utilization | Coding Style | ● Informational | ⓘ Acknowledged |
| **CCD-04** | Centralization Risk | **Centralization / Privilege** | ● **Major** | ⓘ Acknowledged |
| CCD-05 | Set `constant` to Variables | Gas Optimization | ● Informational | ✓ Resolved |
| CCD-06 | Possibility of Replay Attack in `permit()` | Logical Issue | ● Minor | ✓ Resolved |
| CCD-07 | Susceptible to Signature Malleability | Volatile Code | ● Minor | ✓ Resolved |
| EED-01 | Function Visibility Optimization | Gas Optimization | ● Informational | ✓ Resolved |
| EED-02 | Incorrect naming convention utilization | Coding Style | ● Informational | ⓘ Acknowledged |
| EED-03 | Discussion For Function `getAssetBalance()` | Logical Issue | ● Informational | ⓘ Acknowledged |
| EED-04 | Visibility Specifiers Missing | Language Specific | ● Informational | ✓ Resolved |
| **EVD-01** | Centralization Risk | **Centralization / Privilege** | ● **Major** | ⓘ Acknowledged |
| FDD-01 | Incorrect naming convention utilization | Coding Style | ● Informational | ⓘ Acknowledged |

| ID | Title | Category | Severity | Status |
|---|---|---|---|---|
| FFD-01 | Lack of Zero Address Validation | Volatile Code | ● Minor | ✓ Resolved |
| **FFD-02** | Centralization Risk | **Centralization / Privilege** | ● **Major** | ⓘ Acknowledged |
| VVD-01 | Lack of Zero Address Validation | Volatile Code | ● Minor | ✓ Resolved |
| VVD-02 | Missing Emit Events | Gas Optimization | ● Informational | ✓ Resolved |
| **VVD-03** | Centralization Risk | **Centralization / Privilege** | ● **Major** | ⓘ Acknowledged |
| **VVD-04** | Centralization Risk | **Centralization / Privilege** | ● **Major** | ⓘ Acknowledged |
| VVD-05 | Incorrect `amount` Value | Logical Issue | ● Critical | ✓ Resolved |
| VVD-06 | Third Party Dependencies | Volatile Code | ● Minor | ⓘ Acknowledged |
| VVD-07 | Discussion For Function `setEmergency()` | Control Flow | ● Informational | ⓘ Acknowledged |
| VVD-08 | Discussion For Function `emergencyExit()` | Control Flow | ● Informational | ⓘ Acknowledged |

# GLOBAL-01 | Unlocked Compiler Version

| Category | Severity | Location | Status |
|---|---|---|---|
| Language Specific | ● Informational | Global | ⓘ Acknowledged |

## Description

The contract has an unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to ambiguity when debugging as compiler-specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

## Recommendation

It is a general practice to instead lock the compiler at a specific version rather than allow a range of compiler versions to be utilized to avoid compiler-specific bugs and be able to identify ones more easily. We recommend locking the compiler at the lowest possible version that supports all the capabilities wished by the codebase. This will ensure that the project utilizes a compiler version that has been in use for the longest time and as such is less likely to contain yet-undiscovered bugs.

## Alleviation

`[Client]`: Have set compiler version in range of 0.7.x.

# CCD-01 | Lack of Zero Address Validation

| Category | Severity | Location | Status |
|---|---|---|---|
| Volatile Code | ● Minor | systemCoin/Coin.sol: 128, 22, 14 | ✓ Resolved |

## Description

The zero address is not verified when calling the following function.

- `addAuthorization()`
- `removeAuthorization()`
- `mint()`
- `approve()`
- `constructor()`
- `exit()`
- `emergencyExit()`
- `_mintSystemCoin()`
- `_repaySystemCoin()`
- `constructor()`

## Recommendation

We advise the client to check that the aforementioned function's parameters are not zero address.

## Alleviation

`[Client]`: Have add modifier notZeroAddress

```
modifier notZeroAddress(address inputAddress) {
    require(inputAddress != address(0), "Coin : input zero address");
    _;
}
```

# CCD-02 | Function Visibility Optimization

| Category | Severity | Location | Status |
|---|---|---|---|
| Gas Optimization | ● Informational | systemCoin/Coin.sol: 138, 128, 99 | ✓ Resolved |

## Description

The following functions are declared as `public` and are not invoked in any of the contracts contained within the project's scope:

- `transfer()`
- `mint()`
- `burn()`
- `getInterestInfo()`

The functions that are never called internally within the contract should have external visibility.

## Recommendation

We advise that the functions' visibility specifiers are set to `external` and the array-based arguments change their data location from `memory` to `calldata`, optimizing the gas cost of the function.

## Alleviation

The client heeded our advice and resolved this issue.

# CCD-03 | Incorrect naming convention utilization

| Category | Severity | Location | Status |
|---|---|---|---|
| Coding Style | ● Informational | systemCoin/Coin.sol: 71 | ⓘ Acknowledged |

## Description

Naming conventions are powerful when adopted and used broadly. The use of different conventions can convey significant meta information that would otherwise not be immediately available.

Solidity defines a naming convention that should be followed.

- Contracts and libraries should be named using the CapWords style.
- Structs should be named using the CapWords style.
- Events should be named using the CapWords style.
- Functions should use mixedCase.
- Function arguments should use mixedCase.
- Local and State Variable Names should use mixedCase.
- Constants should be named with all capital letters with underscores separating words.
- Enums, in the style of simple type declarations, should be named using the CapWords style.

Reference: https://docs.soliditylang.org/en/latest/style-guide.html#naming-conventions

## Recommendation

We advise the client to follow the Solidity naming convention. The recommendations outlined here are intended to improve the readability, and thus they are not rules, but rather guidelines to try and help convey the most information through the names of things.

## Alleviation

`[Client]`: the contract's ABIs are already well used in many programs. It's hard to fix it.

## CCD-04 | Centralization Risk

| Category | Severity | Location | Status |
|---|---|---|---|
| **Centralization / Privilege** | ● **Major** | systemCoin/Coin.sol: 128, 22, 14 | ⓘ Acknowledged |

### Description

In the contract `Coin`, the role `authorized` has the authority over the following function:

- `addAuthorization()`
- `removeAuthorization()`
- `mint()`

Any compromise to the `authorized` account may allow the hacker to take advantage of this and:

- maliciously add auth to an account through `addAuthorization()`
- maliciously remove auth from an account through `removeAuthorization()`
- maliciously mint new coins through `mint()`

### Recommendation

We advise the client to carefully manage the `authorized` account's private key to 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 smart-contract-based accounts with enhanced security practices, e.g. Multisignature wallets.

Indicatively, here is some feasible suggestions that would also mitigate the potential risk at the different level in term of short-term and long-term:

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

### Alleviation

`[Client]`: `Coin.sol` must be deployed by `defrostFactory.sol`, all authorized roles must be `defrostFactory.sol` and `collateralVault.sol`. So, Personal account cannot become an authorized role.

# CCD-05 | Set `constant` to Variables

| Category | Severity | Location | Status |
|---|---|---|---|
| Gas Optimization | ● Informational | systemCoin/Coin.sol: 40 | ✓ Resolved |

## Description

The variable `version` is unchanged throughout the contract.

## Recommendation

We advise the client to set `version` as `constant` variables.

## Alleviation

The client heeded our advice and resolved this issue.

# CCD-06 | Possibility of Replay Attack in `permit()`

| Category | Severity | Location | Status |
|---|---|---|---|
| Logical Issue | ● Minor | systemCoin/Coin.sol: 178 | ✓ Resolved |

## Description

The `permit` function performs the operation of deriving signer address from the signature values of `v`, `r` and `s`. The state variable `DOMAIN_SEPARATOR` that is used to calculate hash has a value of `chainid` that is derived only once in the constructor, which does not change after contract deployment. The issue arises in the event of fork when the cross-chain replay attacks can be executed. The attack scenario can be thought of as if a fork of Ethereum happens and two different networks have id of for example `1` and `9`. The `chainid` coded in `DOMAIN_SEPARATOR` will be the same on contracts residing in both of the forks. If the chainid `1` is stored in the contract then the `permit` transaction signed for chainid `1` will be executable on both of the forks.

## Recommendation

We advise to construct the `DOMAIN_SEPARATOR` hash inside the permit function so the current `chainid` could be fetched and only the transactions signed for current network could succeed.

## Alleviation

`[Client]`: Have removed function `permit()`.

# CCD-07 | Susceptible to Signature Malleability

| Category | Severity | Location | Status |
|---|---|---|---|
| Volatile Code | ● Minor | systemCoin/Coin.sol: 188 | ✓ Resolved |

## Description

The signature malleability is possible within the Elliptic Curve cryptographic system. An Elliptic Curve is symmetric on the X-axis, meaning two points can exist with the same `X` value. In the `r`, `s` and `v` representation this permits us to carefully adjust `s` to produce a second valid signature for the same `r`, thus breaking the assumption that a signature cannot be replayed in what is known as a replay-attack.

## Recommendation

We advise the client to utilize a `recover()` function similar to that of the `ECDSA.sol` implementation of OpenZeppelin.

## Alleviation

`[Client]`: Have removed function `permit()`.

# EED-01 | Function Visibility Optimization

| Category | Severity | Location | Status |
|---|---|---|---|
| Gas Optimization | ● Informational | interestEngine/interestEngine.sol: 42 | ✓ Resolved |

## Description

The following functions are declared as `public` and are not invoked in any of the contracts contained within the project's scope:

- `transfer()`
- `mint()`
- `burn()`
- `getInterestInfo()`

The functions that are never called internally within the contract should have external visibility.

## Recommendation

We advise that the functions' visibility specifiers are set to `external` and the array-based arguments change their data location from `memory` to `calldata`, optimizing the gas cost of the function.

## Alleviation

The client heeded our advice and resolved this issue.

# EED-02 | Incorrect naming convention utilization

| Category | Severity | Location | Status |
|---|---|---|---|
| Coding Style | ● Informational | interestEngine/interestEngine.sol: 13 | ⓘ Acknowledged |

## Description

Naming conventions are powerful when adopted and used broadly. The use of different conventions can convey significant meta information that would otherwise not be immediately available.

Solidity defines a naming convention that should be followed.

- Contracts and libraries should be named using the CapWords style.
- Structs should be named using the CapWords style.
- Events should be named using the CapWords style.
- Functions should use mixedCase.
- Function arguments should use mixedCase.
- Local and State Variable Names should use mixedCase.
- Constants should be named with all capital letters with underscores separating words.
- Enums, in the style of simple type declarations, should be named using the CapWords style.

Reference: https://docs.soliditylang.org/en/latest/style-guide.html#naming-conventions

## Recommendation

We advise the client to follow the Solidity naming convention. The recommendations outlined here are intended to improve the readability, and thus they are not rules, but rather guidelines to try and help convey the most information through the names of things.

## Alleviation

`[Client]`: the contract’s ABIs are already well used in many programs. It’s hard to fix it.

# EED-03 | Discussion For Function `getAssetBalance()`

| Category | Severity | Location | Status |
|---|---|---|---|
| Logical Issue | ● Informational | interestEngine/interestEngine.sol: 65 | ⓘ Acknowledged |

## Description

In the function `getAssetBalance()`, zero would be returned when either `interestRateOrigin` or `interestInterval` is equal to 0, instead of the `assetAndInterest`. We would like to inquire why it is the case.

## Alleviation

`[Client]`: If the user's `interestRateOrigin` is zero, he will not interact with this contract, his asset balance will be zero. If `interestInterval` Is zero, the contract will not start to work, all users' asset balance will be zero.

# EED-04 | Visibility Specifiers Missing

| Category | Severity | Location | Status |
|---|---|---|---|
| Language Specific | ● Informational | interestEngine/interestEngine.sol: 13 | ✓ Resolved |

## Description

The linked variable declarations do not have a visibility specifier explicitly set.

## Recommendation

Inconsistencies in the default visibility the Solidity compilers impose can cause issues in the functionality of the codebase. We advise that visibility specifiers for the linked variables are explicitly set.

## Alleviation

`[Client]`: uint256 constant internal rayDecimals = 1e27;

# EVD-01 | Centralization Risk

| Category | Severity | Location | Status |
|---|---|---|---|
| **Centralization / Privilege** | ● **Major** | collateralVault/vaultEngine.sol: 19 | ⓘ Acknowledged |

## Description

In the contract `collateralVault`、 `vaultEngine` and `defrostFactory`, the role `origin` has the authority over the following function:

- `setEmergency()`
- `setLiquidationInfo()`
- `setPoolLimitation()`
- `setStabilityFee()`
- `createVault()`
- `createSystemCoin()`

Any compromise to the `origin` account may allow the hacker to take advantage of this and:

- set emergency start time through `setEmergency()`
- set liquidation info through `setLiquidationInfo()`
- set pool asset limitation through `setPoolLimitation()`
- set stability fee through `setStabilityFee()`
- create vault through `createVault()`
- create system coins through `createSystemCoin()`

## Recommendation

We advise the client to carefully manage the `origin` account's private key to 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 smart-contract-based accounts with enhanced security practices, e.g. Multisignature wallets.

Indicatively, here is some feasible suggestions that would also mitigate the potential risk at the different level in term of short-term and long-term:

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;

- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

## Alleviation

`[Client]`: onlyOrigin is a multiple signature authorized check. The accounts origin0 and origin1 have the property to create an invoked application, multiple signature will signature this application.

```
    modifier onlyOrigin() {
        require (isOrigin(),"proxyOwner: caller is not the tx origin!");
        checkMultiSignature();
        _;
    }
    function checkMultiSignature() internal {
        uint256 value;
        assembly {
            value := callvalue()
        }
        bytes32 msgHash = keccak256(abi.encodePacked(msg.sender,
address(this),value,msg.data));
        address multiSign = getMultiSignatureAddress();
        uint256 index = getValue(uint256(msgHash));
        uint256 newIndex = IMultiSignature(multiSign).getValidSignature(msgHash,index);
        require(newIndex > index, "multiSignatureClient : This tx is not aprroved");
        saveValue(uint256(msgHash),newIndex);
    }
```

# FDD-01 | Incorrect naming convention utilization

| Category | Severity | Location | Status |
|---|---|---|---|
| Coding Style | ● Informational | defrostFactory/defrostFactoryData.sol: 5 | ⓘ Acknowledged |

## Description

Naming conventions are powerful when adopted and used broadly. The use of different conventions can convey significant meta information that would otherwise not be immediately available.

Solidity defines a naming convention that should be followed.

- Contracts and libraries should be named using the CapWords style.
- Structs should be named using the CapWords style.
- Events should be named using the CapWords style.
- Functions should use mixedCase.
- Function arguments should use mixedCase.
- Local and State Variable Names should use mixedCase.
- Constants should be named with all capital letters with underscores separating words.
- Enums, in the style of simple type declarations, should be named using the CapWords style.

Reference: https://docs.soliditylang.org/en/latest/style-guide.html#naming-conventions

## Recommendation

We advise the client to follow the Solidity naming convention. The recommendations outlined here are intended to improve the readability, and thus they are not rules, but rather guidelines to try and help convey the most information through the names of things.

## Alleviation

`[Client]`: the contract’s ABIs are already well used in many programs. It’s hard to fix it.

# FFD-01 | Lack of Zero Address Validation

| Category | Severity | Location | Status |
|---|---|---|---|
| Volatile Code | ● Minor | defrostFactory/defrostFactory.sol: 13 | ✓ Resolved |

## Description

The zero address is not verified when calling the following function.

- `addAuthorization()`
- `removeAuthorization()`
- `mint()`
- `approve()`
- `constructor()`
- `exit()`
- `emergencyExit()`
- `_mintSystemCoin()`
- `_repaySystemCoin()`
- `constructor()`

## Recommendation

We advise the client to check that the aforementioned function's parameters are not zero address.

## Alleviation

`[Client]`: Have add modifier notZeroAddress

```
modifier notZeroAddress(address inputAddress) {
    require(inputAddress != address(0), "Coin : input zero address");
    _;
}
```

# FFD-02 | Centralization Risk

| Category | Severity | Location | Status |
|---|---|---|---|
| **Centralization / Privilege** | ● **Major** | defrostFactory/defrostFactory.sol: 43, 18 | ⓘ Acknowledged |

## Description

In the contract `collateralVault`、 `vaultEngine` and `defrostFactory`, the role `origin` has the authority over the following function:

- `setEmergency()`
- `setLiquidationInfo()`
- `setPoolLimitation()`
- `setStabilityFee()`
- `createVault()`
- `createSystemCoin()`

Any compromise to the `origin` account may allow the hacker to take advantage of this and:

- set emergency start time through `setEmergency()`
- set liquidation info through `setLiquidationInfo()`
- set pool asset limitation through `setPoolLimitation()`
- set stability fee through `setStabilityFee()`
- create vault through `createVault()`
- create system coins through `createSystemCoin()`

## Recommendation

We advise the client to carefully manage the `origin` account's private key to 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 smart-contract-based accounts with enhanced security practices, e.g. Multisignature wallets.

Indicatively, here is some feasible suggestions that would also mitigate the potential risk at the different level in term of short-term and long-term:

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;

- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

## Alleviation

`[Client]`: onlyOrigin is a multiple signature authorized check. The accounts origin0 and origin1 have the property to create an invoked application, multiple signature will signature this application.

```
    modifier onlyOrigin() {
        require (isOrigin(),"proxyOwner: caller is not the tx origin!");
        checkMultiSignature();
        _;
    }
    function checkMultiSignature() internal {
        uint256 value;
        assembly {
            value := callvalue()
        }
        bytes32 msgHash = keccak256(abi.encodePacked(msg.sender,
address(this),value,msg.data));
        address multiSign = getMultiSignatureAddress();
        uint256 index = getValue(uint256(msgHash));
        uint256 newIndex = IMultiSignature(multiSign).getValidSignature(msgHash,index);
        require(newIndex > index, "multiSignatureClient : This tx is not aprroved");
        saveValue(uint256(msgHash),newIndex);
    }
```

# VVD-01 | Lack of Zero Address Validation

| Category | Severity | Location | Status |
|---|---|---|---|
| Volatile Code | ● Minor | collateralVault/collateralVault.sol: 116, 97, 78, 72, 7 | ✓ Resolved |

## Description

The zero address is not verified when calling the following function.

- `addAuthorization()`
- `removeAuthorization()`
- `mint()`
- `approve()`
- `constructor()`
- `exit()`
- `emergencyExit()`
- `_mintSystemCoin()`
- `_repaySystemCoin()`
- `constructor()`

## Recommendation

We advise the client to check that the aforementioned function's parameters are not zero address.

## Alleviation

`[Client]`: Have add modifier notZeroAddress

```
modifier notZeroAddress(address inputAddress) {
    require(inputAddress != address(0), "Coin : input zero address");
    _;
}
```

# VVD-02 | Missing Emit Events

| Category | Severity | Location | Status |
|---|---|---|---|
| Gas Optimization | ● Informational | collateralVault/collateralVault.sol: 16 | ✓ Resolved |

## Description

Functions that affect the status of sensitive variables should be able to emit events as notifications to customers.

- `initContract()`

## Recommendation

We advise the client to add events for sensitive actions and emit them in the function.

## Alleviation

The client heeded our advice and resolved this issue.

# VVD-03 | Centralization Risk

| Category | Severity | Location | Status |
|---|---|---|---|
| **Centralization / Privilege** | ● **Major** | collateralVault/collateralVault.sol: 16 | ⓘ Acknowledged |

## Description

In the contract `collateralVault`, the role `owner` has the authority over the following function:

- `initContract()`

Any compromise to the `owner` account may allow the hacker to take advantage of this and:

- init contract through `initContract()`

## Recommendation

We advise the client to carefully manage the `owner` account's private key to 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 smart-contract-based accounts with enhanced security practices, e.g. Multisignature wallets.

Indicatively, here is some feasible suggestions that would also mitigate the potential risk at the different level in term of short-term and long-term:

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

## Alleviation

`[Client]`: Owner must be contract. In this case, owner is `defrostFactory.sol`.

```
function isOwner() public view returns (bool) {
    return msg.sender == owner() && isContract(msg.sender);
}
```

# VVD-04 | Centralization Risk

| Category | Severity | Location | Status |
|---|---|---|---|
| **Centralization / Privilege** | ● **Major** | collateralVault/collateralVault.sol: 42, 32, 27 | ⓘ Acknowledged |

## Description

In the contract `collateralVault`、`vaultEngine` and `defrostFactory`, the role `origin` has the authority over the following function:

- `setEmergency()`
- `setLiquidationInfo()`
- `setPoolLimitation()`
- `setStabilityFee()`
- `createVault()`
- `createSystemCoin()`

Any compromise to the `origin` account may allow the hacker to take advantage of this and:

- set emergency start time through `setEmergency()`
- set liquidation info through `setLiquidationInfo()`
- set pool asset limitation through `setPoolLimitation()`
- set stability fee through `setStabilityFee()`
- create vault through `createVault()`
- create system coins through `createSystemCoin()`

## Recommendation

We advise the client to carefully manage the `origin` account's private key to 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 smart-contract-based accounts with enhanced security practices, e.g. Multisignature wallets.

Indicatively, here is some feasible suggestions that would also mitigate the potential risk at the different level in term of short-term and long-term:

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;

- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

## Alleviation

`[Client]`: onlyOrigin is a multiple signature authorized check. The accounts origin0 and origin1 have the property to create an invoked application, multiple signature will signature this application.

```
    modifier onlyOrigin() {
        require (isOrigin(),"proxyOwner: caller is not the tx origin!");
        checkMultiSignature();
        _;
    }
    function checkMultiSignature() internal {
        uint256 value;
        assembly {
            value := callvalue()
        }
        bytes32 msgHash = keccak256(abi.encodePacked(msg.sender,
address(this),value,msg.data));
        address multiSign = getMultiSignatureAddress();
        uint256 index = getValue(uint256(msgHash));
        uint256 newIndex = IMultiSignature(multiSign).getValidSignature(msgHash,index);
        require(newIndex > index, "multiSignatureClient : This tx is not aprroved");
        saveValue(uint256(msgHash),newIndex);
    }
```

# VVD-05 | Incorrect `amount` Value

| Category | Severity | Location | Status |
|---|---|---|---|
| Logical Issue | ● Critical | collateralVault/collateralVault.sol: 60~61 | ✓ Resolved |

## Description

In the function `_join()`, the `getPayableAmount()` is to transfer the user's assets to the contract which can take the forms of either ETH or other erc20 tokens. If the user transfers ETH, the variable `collateralBalances[account]` needs to increase by `msg.value`, not `amount`. The aforementioned line of code is inconsistent with the actual underlying asset transfer.

## Recommendation

We advise the client to recheck the functions.

## Alleviation

The client heeded our advice and resolved this issue in commit : 07b70dc35bbd876768dbc98faae6e4530a75a9cf.

# VVD-06 | Third Party Dependencies

| Category | Severity | Location | Status |
|---|---|---|---|
| Volatile Code | ● Minor | collateralVault/collateralVault.sol: 14 | ⓘ Acknowledged |

## Description

The contract is serving as the underlying entity to interact with third-party protocols. The scope of the audit would treat those 3rd party entities as black boxes and assume their functional correctness. However, in the real world, 3rd parties may be compromised and lead to assets being lost or stolen.

## Recommendation

We understand that the implementation of `collateralVault.sol` requires interaction with `oraclePrice()`. We encourage the team to carefully review this function for any security vulnerabilities, and constantly monitor the statuses of 3rd parties to mitigate the side effects when unexpected activities are observed.

## Alleviation

`[Client]`: We will use a bridge contract which will be connected with ChainLink oracle.

# VVD-07 | Discussion For Function `setEmergency()`

| Category | Severity | Location | Status |
|---|---|---|---|
| Control Flow | ● Informational | collateralVault/collateralVault.sol: 27 | ⓘ Acknowledged |

## Description

When the contract is halted, the function `setEmergency()` can only be called once. We would like to know if it Is consistent with project design?

## Alleviation

`[Client]`: The definition of the emergency time: When havoc will be happened, this economic system will be stopped, Users will emergency exit from the system.

# VVD-08 | Discussion For Function `emergencyExit()`

| Category | Severity | Location | Status |
|---|---|---|---|
| Control Flow | ● Informational | collateralVault/collateralVault.sol: 78 | ⓘ Acknowledged |

## Description

In the function `emergencyExit()`, all the collateral of the user can be refunded. We would like to know why the user's loan is not settled along with the collateral and if the code logic is consistent with project design?

## Alleviation

`[Client]`: The definition of the emergency time: When havoc will be happened, this economic system will be stopped, Users will emergency exit from the system.

# 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.

### 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.

### 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 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 you (“Customer” or 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, nor may copies be delivered to any other person other than the Company, without CertiK’s prior written consent in each instance.

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.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED “AS IS” AND “AS

AVAILABLE” AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER’S OR ANY OTHER PERSON’S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER’S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK’S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER’S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED “AS IS” AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK’S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING

MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

# 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.

CERTIK