# Code Security Assessment

# FORT

Jan 25th, 2022

# Table of Contents

# Summary

This report has been prepared for FORT to discover issues and vulnerabilities in the source code of the FORT 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 | FORT |
| Description | A DeFi Development and Application System with Unlimited Liquidity. |
| Platform | other |
| Language | Solidity |
| Codebase | https://github.com/FORT-Protocol/Hedge/tree/nest4.0/contracts |
| Commit | 1351c2c604eb8d639868be2bcf039114932990bd |

## Audit Summary

| | |
|---|---|
| Delivery Date | Jan 25, 2022 |
| Audit Methodology | Static Analysis, Manual Review |

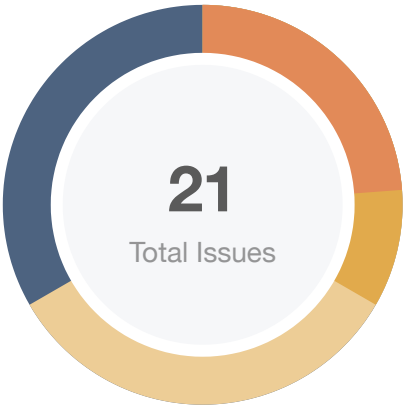
## Vulnerability Summary

| Vulnerability Level | Total | Pending | Declined | Acknowledged | Partially Resolved | Mitigated | Resolved |
|---|---|---|---|---|---|---|---|
| Critical | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Major | 5 | 0 | 0 | 4 | 0 | 0 | 1 |
| Medium | 2 | 0 | 0 | 2 | 0 | 0 | 0 |
| Minor | 7 | 0 | 0 | 5 | 0 | 0 | 2 |
| Informational | 7 | 0 | 0 | 3 | 0 | 0 | 4 |
| Discussion | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## Audit Scope

| ID | File | SHA256 Checksum |
|---|---|---|
| CPF | custom/ChainParameter.sol | 5c9ba6cc296ae2b957b1c08ac0cfb444a069da62966ddd5b007ebd63a9b7f4bd |
| CPO | custom/CommonParameter.sol | 01f84dbd81500a87f67b87821bb1f636c4ff2bb99150098affe2f88debee25be |
| HFU | custom/HedgeFrequentlyUsed.sol | 81eb22f682d0caddbbac55999084ed471fb350dfceb6c1f44b0f227dcf11371c |
| NPA | custom/NestPriceAdapter.sol | b314bb87addf27990aa41e34346f8e982aa728ac23dd3be85c2059822be518b8 |
| DCU | DCU.sol | 9dff73a4f7dcc77e1ecea6b957ede05b99a3cbc0eefc1840358bd047a305bb67 |
| HBF | HedgeBase.sol | 6193637e57b69fdd00245438b77e4ee5d961fb9bd7f0e92ba284ab4d1f0651f6 |
| HDA | HedgeDAO.sol | b1100d7f2cf6a9b489213e46e1309cc2b328db148be327cf685b8044016dfb7b |
| HFF | HedgeFutures.sol | da4e104e529b8ad31fb59d88f2cb18ce1ad41e26c7b688a5f47d8880e4fa8fc7 |
| HGF | HedgeGovernance.sol | 782362f7a233e9e738177504b9757baeced54d565e7069a120c2a994e47046a1 |
| HMF | HedgeMapping.sol | b5a530f635179370cf5291e5be116b1e187f5a18600ff6c23292710fcae523d2 |
| HOF | HedgeOptions.sol | 85dcd6fc9fc474886c19a83a45beb5803911ae98da3ce2d4038424be002836ee |
| HSF | HedgeSwap.sol | e8a357427acde9b941512f277b4d3fc7af8beec8af5a68ce34a90fa8910cdd9f |
| HVF | HedgeVaultForStaking.sol | 4fc0eb38979499b1466fab33835a3a11e33d3793c27e5636456faab80b3f462f |

# Findings




| | |
|---|---|
| **Critical** | **0** (0.00%) |
| **Major** | **5** (23.81%) |
| **Medium** | **2** (9.52%) |
| **Minor** | **7** (33.33%) |
| **Informational** | **7** (33.33%) |
| **Discussion** | **0** (0.00%) |

| ID | Title | Category | Severity | Status |
|---|---|---|---|---|
| Fort-01 | Unlocked compiler version | Language Specific | Informational | Acknowledged |
| Fort-02 | Financial Models | Logical Issue | Informational | Acknowledged |
| **Fort-03** | Centralization Related Risks | **Centralization / Privilege** | **Major** | Acknowledged |
| **DCU-01** | Privileged ownership | **Centralization / Privilege** | **Major** | Acknowledged |
| DCU-02 | Function Naming Convention | Coding Style | Informational | Resolved |
| FOR-01 | Missing emit events | Control Flow | Minor | Resolved |
| FOR-02 | Missing error messages | Coding Style | Informational | Resolved |
| FOR-03 | Unused Import File | Coding Style | Informational | Resolved |
| FOR-04 | Inconsistent Comments and Code on `HedgeOptions.sol` | Inconsistency | Minor | Resolved |
| FOR-05 | Inconsistent Comments and Code in `HedgeOptions.sol` | Inconsistency | Minor | Acknowledged |
| FOR-06 | Inconsistency with White Paper in `HedgeOptions.sol` | Volatile Code | Minor | Acknowledged |
| **HBF-01** | Centralization Risk in HedgeBase.sol | **Centralization / Privilege** | **Major** | Acknowledged |

| ID | Title | Category | Severity | Status |
|---|---|---|---|---|
| HBF-02 | Unnecessary Condition | Logical Issue | Informational | Acknowledged |
| HFU-01 | TODO comments | Coding Style | Informational | Resolved |
| HGF-01 | Unnecessary `require` Statement | Logical Issue | Minor | Acknowledged |
| **HGF-03** | Privileged Ownership | **Centralization / Privilege** | **Major** | Acknowledged |
| HSF-01 | Inconsistent Comments and Code in `HedgeSwap.sol` | Inconsistency | Medium | Acknowledged |
| HSF-03 | Potential Sandwich Attacks in `HedgeSwap.sol` | Logical Issue | Minor | Acknowledged |
| HSF-04 | Inconsistency with White Paper in `HedgeSwap.sol` | Volatile Code | Major | Resolved |
| NPA-01 | Third Party Dependencies | Volatile Code | Minor | Acknowledged |
| NPA-03 | Constant Token Price | Volatile Code | Medium | Acknowledged |

## Fort-01 | Unlocked compiler version

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

### Description

The contract has 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 different compiler versions. This can lead to an 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

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at.

For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

### Alleviation

FORT team acknowledged this finding.

# Fort-02 | Financial Models

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Informational | Global | Acknowledged |

## Description

Referring to the official white paper(https://docs.hedge.red/), FORT is a DeFi development and application system with unlimited liquidity.

## Recommendation

Financial models of blockchain protocols need to be resilient to attacks. They need to pass simulations and verifications to guarantee the security of the overall protocol.

The financial model of this protocol is not in the scope of this audit.

Users should understand the financial models of this protocol before use.

## Alleviation

FORT team acknowledged this finding.

## Fort-03 | Centralization Related Risks

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| **Centralization / Privilege** | **Major** | **Global** | Acknowledged |

## Description

In the contract `DCU.sol`, the roles in `_governanceMapping` have authority over the following functions:

- function `setMinter()`

Any compromise to the accounts in `_governanceMapping` with `flag`(permission weight) greater than `0` may allow a hacker to take advantage of this authority.

---

In the contract `HedgeDAO.sol`, the roles in `_governanceMapping` have authority over the following functions:

- function `setApplication()`

Any compromise to the accounts in `_governanceMapping` with `flag`(permission weight) greater than `0` may allow a hacker to take advantage of this authority.

---

In the contract `HedgeGovernance.sol`, the roles in `_governanceMapping` have authority over the following functions:

- function `setGovernance()`

Any compromise to the accounts in `_governanceMapping` with `flag`(permission weight) greater than `0` may allow a hacker to take advantage of this authority.

---

In the contract `HedgeMapping.sol`, the roles in `_governanceMapping` have authority over the following functions:

- function `setBuiltinAddress()`
- function `registerAddress()`

Any compromise to the accounts in `_governanceMapping` with `flag`(permission weight) greater than `0` may allow a hacker to take advantage of this authority.

---

In the contract `HedgeFutures.sol`, the roles in `_governanceMapping` have authority over the following functions:

- function `create()`

Any compromise to the accounts in `_governanceMapping` with `flag`(permission weight) greater than `0` may allow a hacker to take advantage of this authority.

---

In the contract `HedgeVaultForStaking.sol`, the roles in `_governanceMapping` have authority over the following functions:

- function `setClaimed()`
- function `setConfig()`
- function `batchSetPoolWeight()`

Any compromise to the accounts in `_governanceMapping` with `flag`(permission weight) greater than `0` may allow a hacker to take advantage of this authority.

## Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multi-signature wallets.

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

## Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

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

- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

## Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

## Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles. OR
- Remove the risky functionality.

## Alleviation

FORT team acknowledged this finding and would transfer the ownership to DAO in the future.

# DCU-01 | Privileged ownership

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| **Centralization / Privilege** | **Major** | DCU.sol: 18~21, 40~42, 47~49 | Acknowledged |

## Description

The `_minters` of contract `DCU` has the permission to:

- Executing `mint()` to mint `DCU` token to any account
- Executing `burn()` to destroy `DCU` token from any account without obtaining the consensus of the community.

## Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multi-signature wallets.

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

## Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

## Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

## Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles. OR
- Remove the risky functionality.

## Alleviation

FORT team acknowledged this finding and would transfer the ownership to DAO in the future.

# DCU-02 | Function Naming Convention

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Coding Style | Informational | DCU.sol: 47 | Resolved |

## Description

```
47  function burn(address from, uint value) external onlyMinter {
48    _burn(from, value);
49  }
```

Since function `burn` is used to destroy tokens from the `from` address which is declared as the first parameter, the name of function can be renamed as `burnFrom` to avoid the confusion with burn DCU tokens from address `msg.sender`.

## Recommendation

We recommend renaming the function as `burnFrom()` to improve readability.

## Alleviation

FORT team heeded our advice and renamed the function as `burnFrom()`. The change was supplied in commit `0a39d0ac5885a36733827774ea1c8f69f139313e`.

## FOR-01 | Missing emit events

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Control Flow | Minor | HedgeVaultForStaking.sol: 111~129, 93~97, 85~87<br>HedgeMapping.sol: 121~123, 40~67<br>DCU.sol: 26~28<br>HedgeGovernance.sol: 41~48<br>HedgeBase.sol: 20~23, 28~33 | Resolved |

### Description

There should always be events emitted in the sensitive functions that are controlled by centralization roles.

### Recommendation

It is recommended emitting events for the sensitive functions that are controlled by centralization roles.

### Alleviation

FORT team heeded our advice and added events for the sensitive functions. The change was supplied in commit `0a39d0ac5885a36733827774ea1c8f69f139313e`.

## FOR-02 | Missing error messages

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Coding Style | Informational | HedgeFutures.sol: 479<br>test/NestPriceFacade.sol: 278, 279, 306, 391, 327 | Resolved |

### Description

The **require** can be used to check for conditions and throw an exception if the condition is not met. It is better to provide a string message containing details about the error that will be passed back to the caller.

### Recommendation

We advise adding error messages to the linked **require** statements.

### Alleviation

FORT team heeded our advice and added error messages for the necessary functions. The change was supplied in commit `0a39d0ac5885a36733827774ea1c8f69f139313e`.

## FOR-03 | Unused Import File

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Coding Style | Informational | HedgeOptions.sol: 8, 7<br>HedgeFutures.sol: 8, 7, 5<br>HedgeDAO.sol: 5<br>HedgeBase.sol: 9, 7, 5<br>custom/HedgeFrequentlyUsed.sol: 5 | Resolved |

## Description

The following import file is not used in the audit files:

- ./custom/HedgeFrequentlyUsed.sol

```
5 import "../interfaces/IHedgeGovernance.sol";
```

- HedgeBase.sol

```
5 import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
6
7 import "./libs/TransferHelper.sol";
8
9 import "./interfaces/IHedgeDAO.sol";
```

- HedgeDAO.sol

```
5 import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
```

- HedgeFutures.sol

```
5 import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
6
7 import "./libs/TransferHelper.sol";
8 import "./libs/ABDKMath64x64.sol";
```

- HedgeOptions.sol

```
7 import "./libs/TransferHelper.sol";
8 import "./libs/ABDKMath64x64.sol";
```

## Recommendation

We recommend commenting this import out or removing it.

## Alleviation

FORT team removed the unused import files. The change was supplied in commit `0a39d0ac5885a36733827774ea1c8f69f139313e`.

## FOR-04 | Inconsistent Comments and Code on `HedgeOptions.sol`

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Inconsistency | Minor | custom/ChainParameter.sol: 9<br>HedgeOptions.sol: 343 | Resolved |

### Description

According to the comment, the average time to create a block is 14 seconds, but in contract `ChainParameter`, the value of constant variable `BLOCK_TIME` is 3.

### Recommendation

Please ensure the correct value of this constant variable.

### Alleviation

FORT team corrected the comments. The change was supplied in commit `0a39d0ac5885a36733827774ea1c8f69f139313e`.

## FOR-05 | Inconsistent Comments and Code in `HedgeOptions.sol`

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Inconsistency | Minor | HedgeOptions.sol: 170, 156<br>custom/NestPriceAdapter.sol: 28 | Acknowledged |

### Description

According to the comment, parameter `tokenAddress` can be set to `address(0)` when the target token is ETH. But in function `_latestPrice()`, it will revert if parameter `tokenAddress` is `address(0)`

```
    /// @param tokenAddress 目标代币地址，0表示eth
    ...
        function open(
        address tokenAddress,
        uint strikePrice,
        bool orientation,
        uint exerciseBlock,
        uint dcuAmount
    ) external payable override {


        uint oraclePrice = _latestPrice(tokenAddress, msg.value, msg.sender);
    ...
    }

    function _latestPrice(address tokenAddress, uint fee, address payback) internal
returns (uint oraclePrice) {
        require(tokenAddress == address(0), "HO:not allowed!");
        ...
    }
```

### Recommendation

We recommend correcting the description in comments, or removing the `require` statement from function `_latestPrice()`.

### Alleviation

**[FORT]**: The `tokenAddress` should be `ETH` here, the price is calculated by USD, and the 0 address is only supported `ETH`. In the current stage, only `ETH` options are supported. So only the price of `ETH` could be queried now.

## FOR-06 | Inconsistency with White Paper in `HedgeOptions.sol`

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Volatile Code | Minor | custom/CommonParameter.sol: 15, 12, 9<br>HedgeOptions.sol: 511~530, 487~509 | Acknowledged |

## Description

## Formula and Related Code

According to white paper(https://docs.hedge.red/Hedge/Options.html), the cost per option is obtained according to the following formula:

$$V_c = S_0 e^{\mu T}(1-\phi(\frac{d_1}{\sqrt{T}}-\sigma\sqrt{T}))-K(1-\phi\frac{d_1}{\sqrt{T}}) \quad (1)$$

$$V_p = K\phi(\frac{d_1}{\sqrt{T}})-S_0 e^{\mu T}\phi(\frac{d_1}{\sqrt{T}}-\sigma\sqrt{T}) \quad (2)$$

- $V_p$ is the cost of one put option;
- $\phi(X)$ is the standard normal distribution function;
- $d_1 = \frac{1}{\sigma}[ln\frac{K}{S0}+(\frac{\sigma^2}{2}-\mu)T]$
- $K$ is the strike price;
- $\sigma$ is the volatility, obtained from the NEST oracle;
- $S_0$ is the current price;
- $\mu$ is the underlying return, an arithmetic average based on historical data statistics;
- $T$ is the strike time;

In this finding, we will describe `call option` as example. And in the contract `HedgeOptions`, the $V_c$ will be calculated in function `_calcVc` as blew :

```
487 function _calcVc(uint S0, uint T, uint K) private pure returns (uint vc) {
488
489   int128 sigmaSQ_T = _d18TOb64(SIGMA_SQ * T);
490   int128 miu_T = _toInt128(MIU_LONG * T);
491   int128 sigma_t = ABDKMath64x64.sqrt(sigmaSQ_T);
492   int128 D1 = _D1(S0, K, sigmaSQ_T, miu_T);
493   int128 d = ABDKMath64x64.div(D1, sigma_t);
494
495   uint left = _toUInt(ABDKMath64x64.mul(
496     //ABDKMath64x64.exp(miu_T),
497     // appr equal to x*(1+rt)
```

```
498       // by chenf 2021-12-28 15:27
499       miu_T + ONE,
500       ABDKMath64x64.sub(
501         ONE,
502         _snd(ABDKMath64x64.sub(d, sigma_t))
503       )
504     )) * S0;
505     uint right = _toUInt(ABDKMath64x64.sub(ONE, _snd(d))) * K;
506
507     vc = left > right ? left - right : 0;
508   }
```

## About $d_1 = \frac{1}{\sigma}[ln\frac{K}{S_0} + (\frac{\sigma^2}{2} - \mu)T]$

In the contract `HedgeOptions`, the right part in $d_1$ will be calculated in function `_D1()` as blew :

```
532   function _D1(uint S0, uint K, int128 sigmaSQ_T, int128 miu_T) private pure returns
(int128) {
533
534     //require(K < 0x1000000000000000000000000000000000000000000000000, "FEO:K can't
ROL 64bits");
535     return
536       ABDKMath64x64.sub(
537         ABDKMath64x64.add(
538           ABDKMath64x64.ln(_toInt128(K * 0x10000000000000000 / S0)),
539           sigmaSQ_T >> 1
540         ),
541         miu_T
542       );
543   }
```

Formula $ln\frac{K}{S_0} + (\frac{\sigma^2}{2} - \mu)T$ could be transferred to $ln\frac{K}{S_0} + \frac{\sigma^2}{2}T - \mu T$, and according to the code in `_D1()`, parameter `sigmaSQ_T` represents the formula $\sigma^2 T$, and parameter `miu_T` represents the formula $\mu T$.

## About $\sigma$

Parameter `sigmaSQ_T` is set by following code :

```
uint constant SIGMA_SQ = 45659142400;
```

```
int128 sigmaSQ_T = _d18TOb64(SIGMA_SQ * T);
...
int128 D1 = _D1(S0, K, sigmaSQ_T, miu_T);
```

$\sigma^2$ is set as a constant variable in this protocol, which means $\sigma$ is a constant value, this is different from the description " $\sigma$ is the volatility, obtained from the NEST oracle " in the white paper.

## About $\mu$

In the `call` option, the parameter `miu_T` is set by the following code:

```
uint constant MIU_LONG = 64051194700;
```

```
int128 miu_T = _toInt128(MIU_LONG * T);
...
int128 D1 = _D1(S0, K, sigmaSQ_T, miu_T);
```

In the `put` option, the parameter `miu_T` is set by the following code:

```
uint constant MIU_SHORT= 0;
```

```
int128 miu_T = _toInt128(MIU_SHORT * T);
...
int128 D1 = _D1(S0, K, sigmaSQ_T, miu_T);
```

No matter in call option or in the `put` option, $\mu$ is set as a constant variable in this protocol, which is different from the description " $\mu$ is the underlying return, an arithmetic average based on historical data statistics " in the white paper.

## About Left Part of $V_c$ : $S_0e^{\mu T}(1-\phi(\frac{d_1}{\sqrt{T}}))$

In the contract `HedgeOptions`, $S_0e^{\mu T}(1-\phi(\frac{d_1}{\sqrt{T}}))$ will be calculated in function `_calcVc()` as blew :

```
495 uint left = _toUInt(ABDKMath64x64.mul(
496   //ABDKMath64x64.exp(miu_T),
497   // appr equal to x*(1+rt)
498   // by chenf 2021-12-28 15:27
499   miu_T + ONE,
500   ABDKMath64x64.sub(
501     ONE,
502     _snd(ABDKMath64x64.sub(d, sigma_t))
503   )
504 )) * S0;
```

$e^{\mu T}$ is calculated as $\mu T+$ `0x10000000000000000` in line 499, which is different from the formula $V_c$.

## Recommendation

We recommend enhancing the code or white paper to ensure logical consistency.

## Alleviation

**[FORT]**:

1. The NEST provides real-time volatility, but FORT requires long-term volatility. Long-term volatility is a statistical figure based on the price performance of the corresponding asset over a relatively long period of time. Since currently, only ETH is supported, this should be the ETH price, which is 120% annualized. We will review the white paper.
2. The $e^{\mu}T$ is used to calculate the compound interest, and since the $\mu$ is extremely tiny, the $e^{\mu}T$ can be approximated by $T$. This point has been communicated with the financial product manager and as an optimized solution.

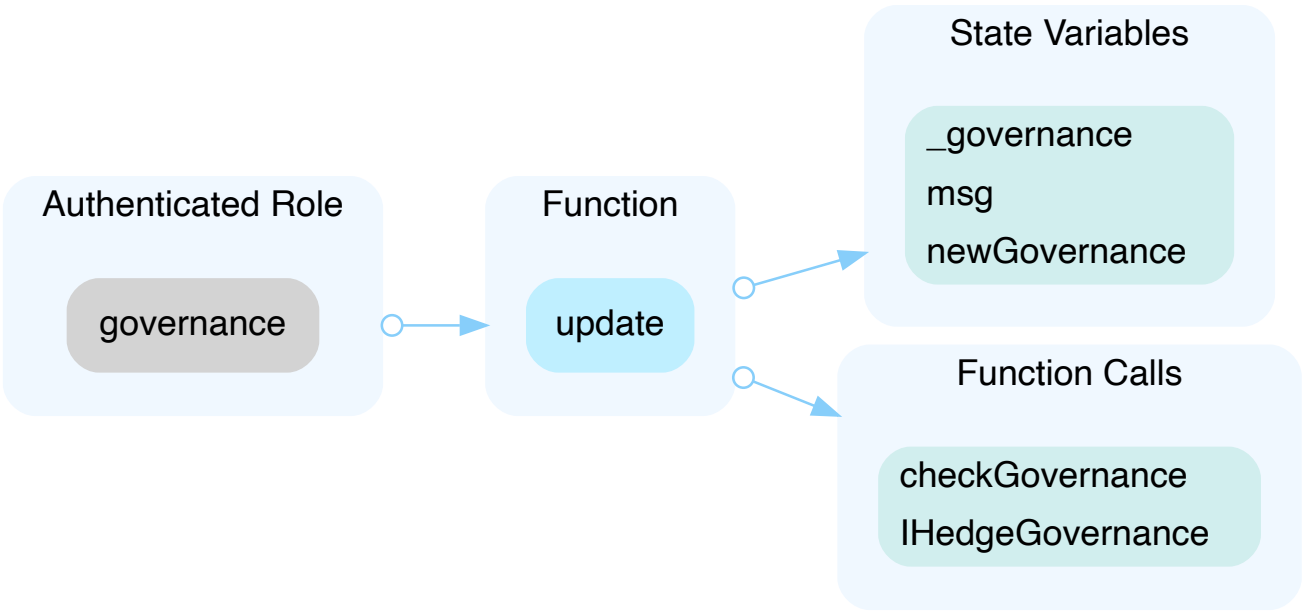
# HBF-01 | Centralization Risk in HedgeBase.sol

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Centralization / Privilege | Major | HedgeBase.sol: 28~33 | Acknowledged |

## Description

In the contract, `HedgeBase`, the role, `governance`, has authority over the functions shown in the diagram below.

Any compromise to the `governance` account may allow the hacker to take advantage of this authority.




## Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets.

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

## Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

## Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

## Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

## Alleviation

FORT team acknowledged this finding and would transfer the ownership to DAO in the future.

## HBF-02 | Unnecessary Condition

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Informational | HedgeBase.sol: 31 | Acknowledged |

### Description

According to the comment of state variable `_governance` in this contract, `_governance` is a contract address and used to declare the interface `IHedgeGovernance`.

```
15    /// @dev IHedgeGovernance implementation contract address
16    address public _governance;
```

According to the code of interface `IHedgeGovernance`, function `HedgeBase.update()` will not be called, which means the `IHedgeGovernance` implementation contract address will never be `msg.sender` of `HedgeBase.update()`.

```
28  function update(address newGovernance) public virtual {
29
30    address governance = _governance;
31    require(governance == msg.sender ||
IHedgeGovernance(governance).checkGovernance(msg.sender, 0), "Hedge:!gov");
32    _governance = newGovernance;
33  }
```

Therefore, the condition `governance == msg.sender` in this require statement is necessary.

### Recommendation

We recommend validating the `governance == msg.sender` in the function `update()`.

### Alleviation

FORT team acknowledged this finding.

## HFU-01 | TODO comments

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Coding Style | Informational | custom/HedgeFrequentlyUsed.sol: 25~62 | Resolved |

### Description

The aforementioned lines contain TODO comments which can be removed to increase the quality of codebase for production environment.

### Recommendation

We advise to remove the TODO comments from the aforementioned lines.

### Alleviation

FORT will remove the TODO comments in the main net version.

## HGF-01 | Unnecessary `require` Statement

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Minor | HedgeGovernance.sol: 18 | Acknowledged |

### Description

No matter what the value of parameter `governance` is, it will not affect the execution of the function.

### Recommendation

We recommend changing the `require` condition and deploying the contract on test net to review the role of function `HedgeGovernance.initialize()`.

### Alleviation

**[FORT]**: The `initialize()` is an `override` function, and it will be used in place of the `constructor` function.

# HGF-03 | Privileged Ownership

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Centralization / Privilege | Major | HedgeGovernance.sol: 41~48 | Acknowledged |

## Description

```
41 function setGovernance(address addr, uint flag) external override onlyGovernance {
42
43   if (flag > 0) {
44     _governanceMapping[addr] = GovernanceInfo(addr, uint96(flag));
45   } else {
46     _governanceMapping[addr] = GovernanceInfo(address(0), uint96(0));
47   }
48 }
```

Any governance could modify the other governances' authorized.

## Recommendation

We recommend avoiding any governance to modify the other governances' authorized.

## Alleviation

FORT team acknowledged this finding.

## HSF-01 | Inconsistent Comments and Code in `HedgeSwap.sol`

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Inconsistency | Medium | HedgeSwap.sol: 42~45, 28~29 | Acknowledged |

### Description

According to the comment, the parameter `payback` is declared to receive the excess fees, but according to the code, fee is not needed in this contract.

Also, if there is the excess fees, it will be better to payback to `msg.sender`.

Meanwhile, if `payback` is the `address(0)`, the caller will lost funds forever.

### Recommendation

We recommend removing the parameter `payback` and codes from line 42 to 45 if fee is not needed in this contract.

### Alleviation

**[FORT]**: The `swap()` interface is defined in another project, `CoFiX`. By implementing this interface, the fund pool can be connected to `CoFiX` for routing exchange. The `payback` parameter is used to return excess fees to `CoFiX`. In order to save gas, it is necessary to read the balance and transfer as little as possible. The payback address provides flexibility so that complex exchange and calling logic can be realized. If it is returned directly to `msg.sender`, it will be more difficult and spend more gas to finish this flow.

**[CertiK]**: We encourage the team to constantly monitor the statuses of `CoFiX.swap()` to mitigate the side effects when unexpected activities are observed.

## HSF-03 | Potential Sandwich Attacks in `HedgeSwap.sol`

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Minor | HedgeSwap.sol: 1 | Acknowledged |

## Description

A sandwich attack might happen when an attacker observes a transaction swapping tokens or adding liquidity without setting restrictions on slippage or minimum output amount. The attacker can manipulate the exchange rate by frontrunning (before the transaction being attacked) a transaction to purchase one of the assets and make profits by backrunning (after the transaction being attacked) a transaction to sell the asset.

The following functions are called without setting minimum output amount or maximum input amount, so transactions triggering these functions are vulnerable to sandwich attacks, especially when the input amount or exact output amount is large:

- `_swap()`
- `_swapExact()`

## Recommendation

We recommend setting reasonable minimum output amounts or maximum input amount, based on token prices when calling the aforementioned functions.

## Alleviation

**[FORT]**: In general, users exchange by `CoFiX`, and there is a limit to the minimum amount obtained in the CoFiX exchange.

## HSF-04 | Inconsistency with White Paper in `HedgeSwap.sol`

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Volatile Code | Major | HedgeSwap.sol: 19 | Resolved |

### Description

The following is described in the official white paper (https://docs.hedge.red/Hedge/SystemDataModel.html)：

- DAO finances 30 million NEST from early developers, community KOLs, etc. for SWAP initial liquidity with a 1:1 consideration, while DAO injects 30 million DCU into SWAP, forming a pool of 30 million NEST: 30 million DCU. Users get DCU Token through SWAP transactions, which currently supports ETH<=>DCU and NEST<=>DCU.

### About Initial Fund

According to code in `HedgeSwap.sol`, there is not 30 million but 15 million `NEST` and 15 million `DCU` will be provided as initial liquidity, which is inconsistent with white paper.

```
19    uint constant K = 15000000 ether * 15000000 ether;
```

Also, there is no token transferred to this contract from `DAO` to initialize this contract, only a constant variable `K` is declared. But without a real token, a number will make no sense.

### About Transactions

Only `NEST<=>DCU` transaction is supported in this contract, `ETH<=>DCU` is not supported.

### Recommendation

We recommend coding as documented in the white paper.

### Alleviation

**[FORT]**: This protocol will be deployed on Ethereum and BSC chains. So the sum is 30 million.

## NPA-01 | Third Party Dependencies

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Volatile Code | Minor | custom/NestPriceAdapter.sol: 5 | Acknowledged |

### Description

The contract is serving as the underlying entity to interact with third party `NEST` protocol. The scope of the audit treats 3rd party entities as black boxes and assume their functional correctness. However, in the real world, 3rd parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of 3rd parties can possibly create severe impacts, such as increasing fees of 3rd parties, migrating to new LP pools, etc.

### Recommendation

We understand that the business logic of `NestPriceAdapter` requires interaction with `NEST`, etc. We encourage the team to constantly monitor the statuses of 3rd parties to mitigate the side effects when unexpected activities are observed.

### Alleviation

**[FORT]**: The NEST and FORT are closely related, and we will continuously monitoring the operation of NEST.

## NPA-03 | Constant Token Price

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Volatile Code | Medium | custom/NestPriceAdapter.sol: 56 | Acknowledged |

## Description

Referring to the code, `DCU` would be a constant price with `ETH`.

For example, the price of `DCU` is calculated as following code :

```
function exercise(uint index, uint amount) external payable override {
  ...
  uint oraclePrice = _findPrice(address(0), exerciseBlock, msg.value, msg.sender);


  uint gain = 0;


  if (orientation) {

    if (oraclePrice > strikePrice) {
      gain = amount * (oraclePrice - strikePrice) / USDT_BASE;
    }
  }
  ...
  if (gain > 0) {
    DCU(DCU_TOKEN_ADDRESS).mint(msg.sender, gain);
  }
  ...
}
```

In the function `HedgeOptions.exercise()`, the latest price of `DCU` will be `oraclePrice / USDT_BASE`.

The dividend `oraclePrice` is set by function `NestPriceAdapter._findPrice()`, which value will be:

```
  42  function _findPrice(address tokenAddress, uint blockNumber, uint fee, address
payback) internal returns (uint oraclePrice) {
  43    require(tokenAddress == address(0), "HO:not allowed!");
  44
  45
  46    (, uint rawPrice) = INestOpenPrice(NEST_OPEN_PRICE).findPrice {
  47      value: fee
  48    } (ETH_USDT_CHANNEL_ID, blockNumber, payback);
  49
  50
```

```
51    oraclePrice = _toUSDTPrice(rawPrice);
52  }
53
54
55  function _toUSDTPrice(uint rawPrice) internal pure returns (uint) {
56    return 2000 ether * 1 ether / rawPrice;
57  }
```

The divisor `USDT_BASE` is a constant variable in `HedgeFrequentlyUsed.sol`

```
22    uint constant USDT_BASE = 1 ether;
```

Therefore, the price of `DCU` will be `2000 ether / rawPrice` can be described as "1 ETH will be 2000 ether DCU". This is a constant price.

But there are no related description records on the white paper.

## Recommendation

We recommend development team could document whether the exchange rate between `DCU` and `ETH` will be constant.

If it will be a constant rate, providing guarantees and services that can exchange 2000 ether `DCU` for 1 `ETH` at any time is needed.

If it will not be a constant rate, correcting code in function `NestPriceAdapter._toUSDTPrice()` is needed.

## Alleviation

**[FORT]**: This is how to use NEST. It is used to query how many ETH is equivalent to 2000 USD.

# 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.

### 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.

### 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.

## 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.