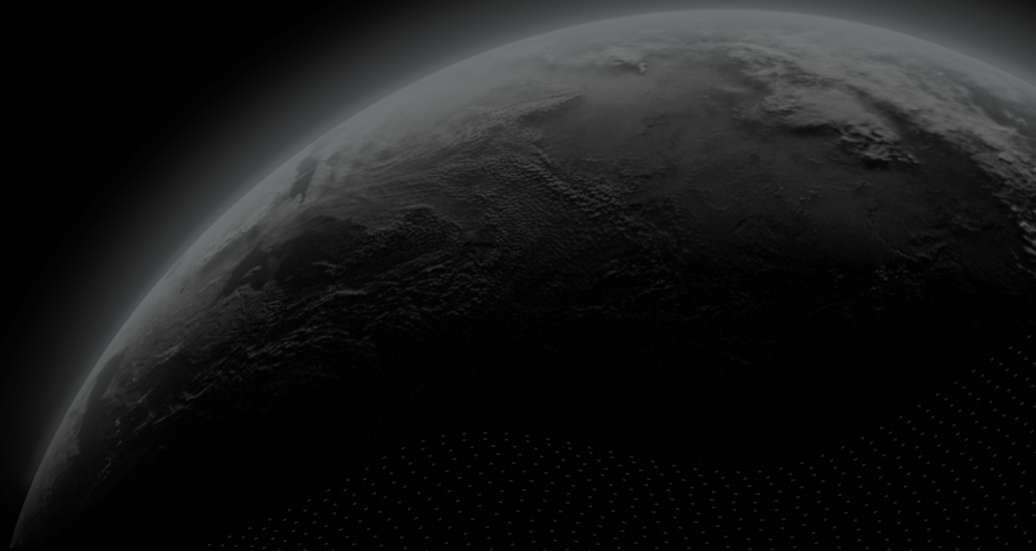




Security Assessment

# Defrost Finance - IV

CertiK Verified on Sept 13th, 2022





Certik Verified on Sept 13th, 2022

## Defrost Finance - IV

The security assessment was prepared by Certik, the leader in Web3.0 security.

### Executive Summary

#### TYPES

DeFi

#### ECOSYSTEM

Avalanche

#### METHODS

Manual Review, Static Analysis

#### LANGUAGE

Solidity

#### TIMELINE

Delivered on 09/13/2022

#### KEY COMPONENTS

N/A

#### CODEBASE

<https://github.com/DefrostFinance/defrost-finance-leverage>  
...View All

#### COMMITTS

[89489759e8a97f3a3b99150d24dbc965097cf2db](#)  
[82111a15936e75762c4fc6a432bc3caf16507519](#)  
...View All

### Vulnerability Summary



9

Total Findings

3

Resolved

1

Mitigated

0

Partially Resolved

5

Acknowledged

0

Declined

0

Unresolved

■ 0 Critical

Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.

■ 1 Major

1 Mitigated



Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.

■ 2 Medium

1 Resolved, 1 Acknowledged



Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.

---

**■ 3 Minor**

2 Resolved, 1 Acknowledged



Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.

---

**■ 3 Informational**

3 Acknowledged



Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

---

# TABLE OF CONTENTS | DEFROST FINANCE - IV

## I **Summary**

Executive Summary

Vulnerability Summary

Codebase

Audit Scope

Approach & Methods

## I **Findings**

GLOBAL-01 : Centralization Related Risks

PPD-01 : Incompatibility with Deflationary Tokens

PPD-02 : Incorrect Implementation of `AVAXDeposit` Modifier

PDF-01 : Third Party Dependency

PPD-03 : Unused Return Value

PPD-04 : Incorrect Validation for ERC20 Token Contract

DFB-01 : Unlocked Compiler Version

PPD-05 : Incorrect Assumption

PPD-06 : Divide by zero

## I **Appendix**

## I **Disclaimer**

# CODEBASE | DEFROST FINANCE - IV

## I Repository

<https://github.com/DefrostFinance/defrost-finance-leverage>





## I Commit

[89489759e8a97f3a3b99150d24dbc965097cf2db](#)

[82111a15936e75762c4fc6a432bc3caf16507519](#)

## AUDIT SCOPE | DEFROST FINANCE - IV

4 files audited ● 4 files with Acknowledged findings

ID	File	SHA256 Checksum
● FFD	 leverageFactory/leverageFactory.sol	100570b78ddd98647c6af9d52e65aa93cad9e60a551628dc2edfd825f61b0334
● FDF	 leverageFactory/leverageFactoryData.sol	bd3a1bb995f6952878858af40bce31319b9551dcef548bd2d9a6c81ce7a0ddfc
● DPD	 leveragePool/leverageData.sol	f1d55c4b6d9e0c6acc8508eca8da70af92d1c11d52a61de68ba5d8c7d965bd61
● PPD	 leveragePool/leveragePool.sol	8549e78c87fc6e4553d472f19d252fd68e3b23767c3224ee5f0cf5f494a262fc

## APPROACH & METHODS | DEFROST FINANCE - IV

This report has been prepared for Defrost Finance to discover issues and vulnerabilities in the source code of the Defrost Finance - IV project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Manual Review and Static Analysis 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:

- Testing the smart contracts against both common and uncommon attack vectors;
- 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.

## FINDINGS | DEFROST FINANCE - IV



9

Total Findings

0

Critical

1

Major

2

Medium

3

Minor

3

Informational

This report has been prepared to discover issues and vulnerabilities for Defrost Finance - IV. Through this audit, we have uncovered 9 issues ranging from different severity levels. Utilizing Static Analysis techniques to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
<b>GLOBAL-01</b>	<b>Centralization Related Risks</b>	<b>Centralization / Privilege</b>	<b>Major</b>	<b>Mitigated</b>
PPD-01	Incompatibility With Deflationary Tokens	Logical Issue	Medium	Acknowledged
PPD-02	Incorrect Implementation Of <code>AVAXDeposit</code> Modifier	Logical Issue	Medium	Resolved
PDF-01	Third Party Dependency	Volatile Code	Minor	Acknowledged
PPD-03	Unused Return Value	Volatile Code	Minor	Resolved
PPD-04	Incorrect Validation For ERC20 Token Contract	Volatile Code	Minor	Resolved
DFB-01	Unlocked Compiler Version	Language Specific	Informational	Acknowledged
PPD-05	Incorrect Assumption	Logical Issue	Informational	Acknowledged
PPD-06	Divide By Zero	Logical Issue	Informational	Acknowledged



## GLOBAL-01 | FINDING DETAILS

### Finding Title

Centralization Related Risks

Category	Severity	Location	Status
Centralization / Privilege	● Major		● Mitigated

### Description

In the contract `leveragePool.sol` the role `Origin` has authority over the functions shown below.

- `setSwapHelper()`
- `acceptSwapHelper()`
- `setOracle()`
- `acceptOracle()`
- `setLiquidationInfo()`

Meanwhile, the role `Owner` and `Origin` has authority over the function `setSwapFee()`

`OwnerOrOrigin`

Any compromise to the privilege role account may allow a hacker to take advantage of this authority and update the sensitive settings and execute sensitive functionalities of the project.

### 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 ( $\frac{2}{3}$ ,  $\frac{3}{5}$ ) 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.

*Noted: Recommend considering the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.*

## I Alleviation

[Defrost Finance]:

Like Defrost V1. OnlyOrigin is a MultiSignature modifier. The MultiSignature Contract Address On avalanche is

<https://snowtrace.io/address/0x7f08ba62fadaf3b4b70a8ddc22b3c63669bddeb2>

timeLockSetting is a time-lock which lock period is 2 days. When `setSwapHelper(address _swapHelper)` Or `setOracle(address _oracle)` is invoked, `acceptSwapHelper()` or `acceptOracle()` will be invoked 2 days later.

```
uint256 public constant timeSpan = 2 days;
settingMap[key] = settingInfo(_value,block.timestamp+timeSpan);
require(settingMap[key].acceptTime>0 && settingMap[key].acceptTime < block.timestamp ,
"timeLock error!");
```

## PPD-01 | FINDING DETAILS

### Finding Title

Incompatibility With Deflationary Tokens

Category	Severity	Location	Status
Logical Issue	● Medium	leveragePool/leveragePool.sol: 85, 104, 124, 139, 140, 149, 151, 159, 160, 163, 165, 182, 184, 204, 205	● Acknowledged

### Description

When transferring deflationary ERC20 tokens, the input amount may not be equal to the received amount due to the charged transaction fee. For example, if a user sends 100 deflationary tokens (with a 10% transaction fee), only 90 tokens actually arrived to the contract. However, a failure to discount such fees may allow the same user to withdraw 100 tokens from the contract, which causes the contract to lose 10 tokens in such a transaction.

Reference: <https://thoreum-finance.medium.com/what-exploit-happened-today-for-gocerberus-and-garuda-also-for-lokum-ybear-piggy-caramelswap-3943ee23a39f>

```
139         lendingToken.safeTransferFrom(msg.sender, address(this), amount);
```

- Transferring tokens by `amount`.

```
140         _leverage(getUserVaultID(msg.sender), amount, leverageRate, slipRate);
```

- This function call executes the following operation.
- In `leveragePool._leverage`,
  - `lendingPool.repay(vaultID, amount);`
  - Note: `repay` is an external function and its behavior wasn't evaluated.
- The `amount` appears to be used for bookkeeping purposes without compensating the potential transfer fees.

```
149         IERC20(underlying).safeTransferFrom(msg.sender, address(this), amount);
```

- Transferring tokens by `amount`.

```
151         _addUnderlying(msg.sender,vaultID,amount);
```

- This function call executes the following operation.
- In `leveragePool._addUnderlying` ,
  - `userVault[userID] = userVault[userID].add(amount);`
- The `amount` appears to be used for bookkeeping purposes without compensating the potential transfer fees.

```
159         lendingToken.safeTransferFrom(msg.sender, address(this),amount);
```

- Transferring tokens by `amount` .

```
160         _buyLeverage(account,getUserVaultID(account),amount,amountLending,slipRate);
```

- This function call executes the following operation.
- In `leveragePool._buyLeverage` ,
  - `uint256 amountAll = amountLending.add(amount);`
- The `amount` appears to be used for bookkeeping purposes without compensating the potential transfer fees.

```
163         IERC20(underlying).safeTransferFrom(msg.sender, address(this),amount);
```

- Transferring tokens by `amount` .

```
165         _addUnderlying(account,vaultID,amount);
```

- This function call executes the following operation.
- In `leveragePool._addUnderlying` ,
  - `userVault[userID] = userVault[userID].add(amount);`
- The `amount` appears to be used for bookkeeping purposes without compensating the potential transfer fees.

```
182         IERC20(underlying).safeTransferFrom(msg.sender, address(this), amount);
```

- Transferring tokens by `amount` .

```
184         _addUnderlying(account, vaultID, amount);
```

- This function call executes the following operation.
- In `leveragePool._addUnderlying` ,
  - `userVault[userID] = userVault[userID].add(amount);`
- The `amount` appears to be used for bookkeeping purposes without compensating the potential transfer fees.

```
204         lendingToken.safeTransferFrom(msg.sender, address(this), amount);
```

- Transferring tokens by `amount` .

```
205         lendingPool.repay(getUserVaultID(account), amount);
```

- The `amount` appears to be used for bookkeeping purposes without compensating the potential transfer fees.
- Note: `repay` is an external function and its behavior wasn't evaluated.

## Recommendation

We advise the client to regulate the set of tokens supported and add necessary mitigation mechanisms to keep track of accurate balances if there is a need to support deflationary tokens.

## Alleviation

[Defrost Finance] : Deflationary tokens are not supported

## PPD-02 | FINDING DETAILS

### Finding Title

Incorrect Implementation Of `AVAXDeposit` Modifier

Category	Severity	Location	Status
Logical Issue	● Medium	leveragePool/leveragePool.sol: 18, 20, 135, 143, 155, 167, 175, 200, 309, 406	● Resolved

### Description

In the contract `leveragePool.sol`, the modifier `AVAXDeposit` checked any given address `asset` and see if its the exact address of `WAVAX` AVAX ERC20 wrapper. This modifier has been used in multiple locations in the `leveragePool.sol` contract to check either `underlying` address (e.g. L143) or `lendingToken` address (e.g. L155) is `WAVAX` address. This potentially suggests that the `underlying` address and `lendingToken` must be the same token address in order to make all the functions in the listed locations work.

However, from the constructor and/or document, both of `underlying` and `lendingToken` addresses are not explicitly confirmed as the `WAVAX` address. Meanwhile, the current implementation of modifier `AVAXDeposit` and usage of it prevents the extension of the leveragePool to other underlying/lendingToken systems.

### Recommendation

Consider revisiting the modifier `AVAXDeposit` and also consider implementing setter functions to control/update the address of `underlying` and the address of `lendingToken`.

### Alleviation

[Defrost Finance]: Some functions with ETH suffix will be used only if lending token is wavax or underlying token is wavax. Otherwise, those functions are unavailable.

## PDF-01 | FINDING DETAILS

### Finding Title

Third Party Dependency

Category	Severity	Location	Status
Volatile Code	● Minor	leveragePool/leverageData.sol: 12, 13, 14, 16, 22, 23; leveragePool/leveragePool.sol: 13	● Acknowledged

### Description

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

```
12      IWAVAX public WAVAX;
```

- The contract `leverageData` interacts with third party contract with `IWAVAX` interface via `WAVAX`.

```
13      IERC20 public lendingToken;
```

- The contract `leverageData` interacts with third party contract with `IERC20` interface via `lendingToken`.

```
14      address public underlying;
```

- The contract `leverageData` interacts with third party contract with `IERC20` interface via `underlying`.

```
16      ILendingPool public lendingPool;
```

- The contract `leverageData` interacts with third party contract with `ILendingPool` interface via `lendingPool`.



```
22     ISwapHelper public swapHelper;
```

- The contract `leverageData` interacts with third party contract with `ISwapHelper` interface via `swapHelper`.

```
23     IDSOracle public oracle;
```

- The contract `leverageData` interacts with third party contract with `IDSOracle` interface via `oracle`.

```
13     address payable _feeAddress,address _lendingPool,address _underlying,
```

- The function `leveragePool.constructor` interacts with third party contract with `IERC20` interface via `_underlying`.

## Recommendation

We understand that the business logic requires interaction with the third parties. We encourage the team to constantly monitor the statuses of third parties to mitigate the side effects when unexpected activities are observed.

## PPD-03 | FINDING DETAILS

### Finding Title

Unused Return Value

Category	Severity	Location	Status
Volatile Code	● Minor	leveragePool/leveragePool.sol: 102	● Resolved

### Description

The return value of an external call is not stored in a local or state variable.

```
102          lendingPool.borrow(vaultID,amountLending);
```

### Recommendation

We recommend checking or using the return values of all external function calls.

### Alleviation

[Defrost Finance]:

```
function borrow(bytes32 account,uint256 amount) external;
```

Remove return value.

## PPD-04 | FINDING DETAILS

### Finding Title

Incorrect Validation For ERC20 Token Contract

Category	Severity	Location	Status
Volatile Code	● Minor	leveragePool/leveragePool.sol: 17	● Resolved

### Description

In the constructor of the contract `leveragePool.sol`, the `_underlying` address is validated to guarantee its a ERC20 token contract address per the log message in the `require()` statement. However this `require()` statement is not adequate to check if a address is pointing to a ERC20 token contract.

### Recommendation

Consider revisiting the `require()` check in the constructor to make sure:

- if the log message is accurate in `require()`
- or follow ERC165 to check if the given address is supporting ERC20 interfaces. Reference: <https://github.com/binodnp/openzeppelin-solidity/blob/master/docs/ERC165.md>

### Alleviation

[Defrost Finance]:

Only want to eliminate the case `underlying == avax`.

```
require(_underlying != address(0), "Please use WAVAX instand");
```

## DFB-01 | FINDING DETAILS

### Finding Title

Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	● Informational	leverageFactory/leverageFactory.sol: 2; leverageFactory/leverageFactoryData.sol: 2; leveragePool/leverageData.sol: 2; leveragePool/leveragePool.sol: 2	● 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;
```

## PPD-05 | FINDING DETAILS

### Finding Title

Incorrect Assumption

Category	Severity	Location	Status
Logical Issue	● Informational	leveragePool/leveragePool.sol: 83	● Acknowledged

### Description

In the function `_addUnderlying()`, there's a assumption that the `userVault` will always be tight to `WAVAX` ERC20 token and `AVAX` native coin with a 1:1 exchange ratio. However current assumptions and implementation restrict the pool's extension and therefore do not allow any other ERC20 token/`AVAX` pool.

### Recommendation

Due to the cost of refactoring, the team could consider monitoring the addresses of the pools, specifically the address of `underlying` by adding checks in the constructor when necessary to guarantee that the ERC20 token could stick to a 1:1 exchange ratio with `AVAX` native coin.

### Alleviation

[Defrost Finance]:

`function addUnderlying(address account, uint256 amount)` is available when underlying is ERC20 token, include wavax.

`function addUnderlyingETH(address account)` is available when underlying is wavax.

## PPD-06 | FINDING DETAILS

### Finding Title

Divide By Zero

Category	Severity	Location	Status
Logical Issue	● Informational	leveragePool/leveragePool.sol: 79, 237, 239, 284, 285, 393	● Acknowledged

### Description

In the following statements, the divisor may be zero, which violates the math operation standard of solidity language.

```
79 return (allUnderlying - loadUSD)/underlyingPrice;
```

```
237 borrow = (allUnderlying-allLoan)/lendingPrice;
```

```
239 payment = (allLoan - allUnderlying)/lendingPrice;
```

```
284 amount =  
amount.mul(lendingPrice).mul(calDecimals.add(liquidationReward))/underlyingPrice/calDecimals  
;
```

```
285 allUnderlying = allUnderlying.mul(repayLoan)/loan;
```

```
393 uint256 amountAll = amountIn.mul(calDecimals)/(calDecimals.sub(swapFee));
```

### Recommendation

We recommend adding sanity checks to skip any scenario in which the divisor is 0

## APPENDIX | DEFROST FINANCE - IV

### Finding Categories

Categories	Description
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.
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.

### 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 | CERTIK

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.

# CertiK | Securing the Web3 World

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.

