Security Assessment

Maverick Protocol

Apr 21st, 2022

Table of Contents

Summary

This report has been prepared for Maverick Protocol to discover issues and vulnerabilities in the source code of the Maverick Protocol 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:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests and apply fuzzing tests to cover the happy-path use cases and potential edge 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.

Note that in Solidity files Pool.sol, Bin.sol, Estimator.sol, etc., some of the mathematical functions related to the ALP (Automated Liquidity Placement) algorithm are treated as blackboxes. Confirmed in the meeting with the Maverick team.

Overview

Project Summary

Project Name	Maverick Protocol
Platform	Ethereum
Language	Solidity
Codebase	https://github.com/MaverickProtocol/contracts/
Commit	e87ccdee097a912856d392643936365977a645fc

Audit Summary

Delivery Date	Apr 21, 2022 UTC
Audit Methodology	Manual Review, Static Analysis

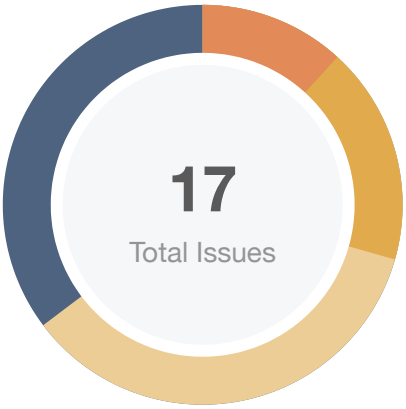
Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Mitigated	Partially Resolved	Resolved
Critical	0	0	0	0	0	0	0
Major	2	0	0	2	0	0	0
Medium	3	0	0	1	0	0	2
Minor	6	0	0	4	0	0	2
Informational	6	0	0	2	0	0	4
Discussion	0	0	0	0	0	0	0

Audit Scope

ID	File	SHA256 Checksum
FCK	models/Factory.sol	06c501ae74c9aa4329d5bfe8584a1171714193ccb896038ed7c840e01d3661a6
ZER	models/ZeroExRouter.sol	547a69a035a33b5de0e45ad334bc518ad188a8ba37adb7b54b511d5ca0e0f972
PCK	models/Pool.sol	4360cd405a4864a8533621b07227f83751a822273674e7434a96bb437ee72407
BCK	libraries/Bin.sol	9e1837f423362e385cd3755a5f86018d8babc7d1d733d1f6855dda3f1af27d21
CCK	libraries/Cast.sol	ba8e98124498a476bb6586ebdf7f0cb98339b2f05915632f7bc5340e88b149fb
BMC	libraries/BasicMath.sol	89263089be612116bfc5a86fda11bd615d7c93b254faddce364b6e070969ee9a
LPT	models/LPToken.sol	8389dbf16b95e6a39e87d5f8ce4c4a4b0aa83362cfcb74409788e7cb6d06637b
AMC	libraries/AdvancedMath.sol	04597098a0288ee52bca121f411c277d59c189f1dea52ed2136636579bf85938
CCP	libraries/Checkpoint.sol	f97a9635af4783d6279ccbdfd977a795b7fe228c2fdbf1da0b31be8438a9145f
STC	libraries/SafeTransfer.sol	fb4be3c39b0ae0efeec2f9cbce508d9b1b078299dfeb831502c01caa667c5f94
RCK	models/Router.sol	e965d33b3ac0eed1156afd899f845a99792f51cbfed4f4e322242f2dbd5eb9e4

Findings



Critical	**0** (0.00%)
Major	**2** (11.76%)
Medium	**3** (17.65%)
Minor	**6** (35.29%)
Informational	**6** (35.29%)
Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
GLOBAL-01	Third Party Dependencies	Volatile Code	Minor	Acknowledged
GLOBAL-02	Lack Of Zero Address Checks	Volatile Code	Minor	Resolved
GLOBAL-03	Gas Optimization On `uint` And `int` Values	Language Specific, Gas Optimization	Informational	Acknowledged
CCP-01	`Checkpoint.initialize()` Can Be Called Multiple Times	Logical Issue	Medium	Resolved
CCP-02	Potential Error In Function `Checkpoint.append()`	Volatile Code	Minor	Resolved
CCP-03	Division Before Multiplication	Mathematical Operations	Informational	Resolved
CCP-04	`public` Function Could Be Declared As `external`	Gas Optimization	Informational	Resolved
CKP-01	Risks Of Using `balance` Of An Address For Swapping	Volatile Code	Medium	Acknowledged
CKP-02	Incompatibility With Deflationary Tokens(Swap)	Volatile Code	Minor	Acknowledged
CKP-03	Missing Proper Handling Of `msg.value`	Volatile Code	Minor	Acknowledged
CKP-04	Lack Of Pool Validity Checks	Volatile Code	Minor	Acknowledged
CKP-05	Variables Could Be Declared Immutable	Gas Optimization	Informational	Acknowledged

ID	Title	Category	Severity	Status
CON-01	Unused Struct Field / Function Parameter	Gas Optimization	Informational	Resolved
FCK-01	Typos In Comments And Variable Names	Coding Style	Informational	Resolved
PCK-01	Centralization Related Risks	**Centralization / Privilege**	**Major**	Acknowledged
PCK-02	Missing Router Restrictions On `Pool.addLiquidity()` And `Pool.removeLiquidity()`	Control Flow	Major	Acknowledged
PCK-03	`Pool.initialize()` Can Be Called Multiple Times	Logical Issue	Medium	Resolved

GLOBAL-01 | Third Party Dependencies

Category	Severity	Location	Status
Volatile Code	Minor	Global	Acknowledged

Description

The contract is serving as the underlying entity to interact with third party 0x protocol and prb-math library. 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. Note that ZeroExRouter.sol is relied on the liquidity function interface of 0x protocol.

Recommendation

We understand that the business logic of Maverick Protocol requires interaction with 0x and the libraries of prb-math and OpenZeppelin. We encourage the team to constantly monitor the statuses of 3rd parties to mitigate the side effects when unexpected activities are observed.

GLOBAL-02 | Lack Of Zero Address Checks

Category	Severity	Location	Status
Volatile Code	Minor	Global	Resolved

Description

Addresses should be checked before assignment or external call to make sure they are not zero addresses.

Recommendation

Recommend adding some non-zero checks for the passed-in address value to prevent unexpected errors.

Alleviation

Fixed in commit hash `a1e0bcd6dc2ef0b41c0987f92a155790cc4a50dc`.

GLOBAL-03 | Gas Optimization On `uint` And `int` Values

Category	Severity	Location	Status
Language Specific, Gas Optimization	Informational	Global	Acknowledged

Description

This finding is not a security issue.

We noticed that `uint8` ,`uint16`, `uint32`, `int128`, `uint224`, etc. are used for variable type declaration. We would like to mention that all other `int` types except `int256` would be implicitly converted to `int256` during code execution, since EVM is designed and works with `256bit/32byte`. The implicit conversion would lead to extra gas cost.

Here attached an example showing the gas cost:

```
pragma solidity 0.8.4;
contract A { //69729 gas
 uint8 a = 0;
}

contract B { //69324 gas
 uint256 a = 0;
}
```

Recommendation

Recommend reviewing the design docs and the codes to determine the importance of gas or readability. Also recommend adding fuzzing tests on the calculations are correct on both signed and unsigned integers.

CCP-01 | `Checkpoint.initialize()` Can Be Called Multiple Times

Category	Severity	Location	Status
Logical Issue	Medium	libraries/Checkpoint.sol: 18	Resolved

Description

In library Checkpoint, the function `initialize()` is used to initialize `buffer.maxSize` and to push the first Checkpoint instance to `buffer.list`. Currently this function can be called after the first valid call, which would still push new instances but bypass the updating logics in `append()`.

Recommendation

Recommend adding some `require` checks to make sure the function cannot be called after the first valid call.

Alleviation

Checkpoint.sol is removed from the repository.

CCP-02 | Potential Error In Function `Checkpoint.append()`

Category	Severity	Location	Status
Volatile Code	Minor	libraries/Checkpoint.sol: 48~49	Resolved

Description

In function `append()`, it seems possible that when `append()` is called, and when the function wants to immediately append a new buffer, the `buffer.end - 2` operation would lead to error. E.g. when `buffer.end` is `1`.

```
if (previous.timestamp == timestamp) {
   current = buffer.list[buffer.end - 1];
   previous = buffer.list[buffer.end - 2];
...
```

Also, the `diff` would be `0` when the condition of `previous.timestamp == timestamp` fulfills.

Recommendation

Recommend properly handling the edge case and adding unit tests to cover this and more edge cases.

Alleviation

Checkpoint.sol is removed from the repository.

CCP-03 | Division Before Multiplication

Category	Severity	Location	Status
Mathematical Operations	Informational	libraries/Checkpoint.sol: 151~153	Resolved

Description

Performing integer division before multiplication truncates the low bits, which could potentially lead to the precision loss.

Checkpoint.sol: `twap()`

```
uint256 previousU = (start.cumulative - previous.cumulative) / (start.timestamp - previous.timestamp);
```

```
cumulativeDiff += (start.timestamp - startTimestamp) * previousU;
```

Recommendation

Recommend applying multiplication before division to avoid precision loss.

Alleviation

Checkpoint.sol is removed.

CCP-04 | `public` Function Could Be Declared As `external`

Category	Severity	Location	Status
Gas Optimization	Informational	libraries/Checkpoint.sol: 119	Resolved

Description

The function(s) which are never called internally within the contract could have external visibility for gas optimization.

e.g. Checkpoint.sol: `twap()`

Recommendation

Recommend changing the visibility from `public` to `external` for gas saving.

Alleviation

Checkpoint.sol is removed.

CKP-01 | Risks Of Using `balance` Of An Address For Swapping

Category	Severity	Location	Status
Volatile Code	Medium	models/ZeroExRouter.sol: 48, 92, 132 models/Router.sol: 175	Acknowledged

Description

Using `address(this).balance` or `token.balanceOf(address(this))` as the `inputAmount` is risky.

Function `swapEthForToken()` in contract Router and functions `sellTokenForToken()`, `sellEthForToken()` and `sellTokenForETH()` in ZeroExRouter are functions that swaps between tokens and tokens (or between ETH and tokens). When the direction is swapping ETH to token, the amount to be swapped should be the attached `msg.value` in the specific transaction. When the direction is swapping token to ETH or token to token, the amount to be swapped should be the `msg.sender`'s input amount, which need to be pointed out by the function caller as a function parameter.

Take `ZeroExRouter.sellTokenForToken()` as an example. A userA want to sell tokenA for tokenB, and the recipient is userA itself. The userA should first transfer amountA of tokenA to the contract (`tx-a1`), and then call `sellTokenForToken(tokenA, tokenB, userA, minAmount, )` (`tx-a2`). A hackerZ can monitor the pattern of the two transactions. The hackerC can wait for the execution of userA's `tx-a1`, and then front run userA's `tx-a2`, with `sellTokenForToken(tokenA, tokenB, hackerZ, minAmount, )` (`tx-z1`) to take advantage of userA.

Recommendation

Recommend using the received amount in calculation instead of the current balance.

Note that if these functions are not external callable, i.e. if these functions are internal or private functions used together with prior token/ETH transfers, the current implementation might be reusable. Just remember to properly handle the deflationary token and reentrancy cases.

Alleviation

The current design is to apply the interface of 0x liquidity functions.

CKP-02 | Incompatibility With Deflationary Tokens(Swap)

Category	Severity	Location	Status
Volatile Code	Minor	models/Router.sol: 67, 73, 100, 107, 160, 207 models/ZeroExRouter.sol: 61, 143 models/Pool.sol: 173, 179, 310~311, 334, 342, 543, 551	Acknowledged

Description

When transferring standard ERC20 deflationary tokens, the input amount may not be equal to the received amount due to the charged transaction fee. As a result, an inconsistency in the amount will occur and the transaction may fail due to the validation checks.

Recommendation

One possible approach is to regulate the set of LP tokens supported and add necessary mitigation mechanisms to keep track of accurate balances if there is a need to support deflationary tokens. Another approach is to setup a token whitelist, and only approved tokens can be used to create pools. Note that for the whitelist approach, please also see PCK-1 Centralization Related Risks for more information.

Alleviation

The Maverick Team accepts the risk.

CKP-03 | Missing Proper Handling Of `msg.value`

Category	Severity	Location	Status
Volatile Code	Minor	models/ZeroExRouter.sol: 97 models/Router.sol: 177	Acknowledged

Description

Functions `Router.swap` and `ZeroExRouter.sellEthForToken()` are declared with the `payable` keyword. However, the `msg.value` is never handled if the function is invoked in a transaction.

(This finding is related to CKP-01 Risks of Using `balance` of an Address for Swapping)

Recommendation

Recommend adding checks to make sure the received `msg.value` is equal to the current Ether balance, if there are no other use cases of transferring Ether to these two contracts

Alleviation

Same issue to CKP-01

CKP-04 | Lack Of Pool Validity Checks

Category	Severity	Location	Status
Volatile Code	Minor	models/Factory.sol: 117 models/Router.sol: 60, 127, 152, 185, 205, 232 models/ZeroExRouter.sol: 58, 102, 141, 190	Acknowledged

Description

In Router and ZeroExRouter, when the pool address is about to be used, it would be get from `factory.lookup()`. The return value of `factory.lookup()` is from the mapping of `pools[poolKey]`.

However, there's no sanity check to validate if a pool exists. It is possible that the pool is never created, and the `pools[poolKey]` is the default `address(0)`.

Recommendation

Recommend adding validation checks to make sure the pools exist.

Alleviation

It is intended design for gas saving

CKP-05 | Variables Could Be Declared Immutable

Category	Severity	Location	Status
Gas Optimization	Informational	models/Factory.sol: 29, 30, 33, 34, 35, 36, 37 models/ZeroExRouter.sol: 18, 19	Acknowledged

Description

The linked variables assigned in the constructor can be declared as `immutable`. Immutable state variables can be assigned during contract creation but will remain constant throughout the lifetime of a deployed contract. A big advantage of immutable variables is that reading them is significantly cheaper than reading from regular state variables since they will not be stored in storage.

Recommendation

We recommend declaring these variables as immutable. Please note that the `immutable` keyword only works in Solidity version `v0.6.5` and up.

CON-01 | Unused Struct Field / Function Parameter

Category	Severity	Location	Status
Gas Optimization	Informational	libraries/Checkpoint.sol: 12 models/ZeroExRouter.sol: 53, 96, 136	Resolved

Description

Checkpoint.sol: The `uint16 array` field in struct `Buffer` is never used.

ZeroExRouter.sol: The `bytes calldata auxiliaryData` parameter in functions `sellTokenForToken()`, `sellEthForToken()` and `sellTokenForEth()` is never userd.

Recommendation

Recommend removing the unused code snippets.

Alleviation

Checkpoint.sol is removed. The parameter of `sellTokenForToken()` function is to apply the 0x interface.

FCK-01 | Typos In Comments And Variable Names

Category	Severity	Location	Status
Coding Style	Informational	models/Factory.sol: 162~188	Resolved

Description

`boundries` <> `boundaries`

Recommendation

Recommend fixing these typos for better readability and open-source purposes.

Alleviation

Fixed in commit hash `a1e0bcd6dc2ef0b41c0987f92a155790cc4a50dc`.

PCK-01 | Centralization Related Risks

Category	Severity	Location	Status
Centralization / Privilege	**Major**	models/Pool.sol: 166	Acknowledged

Description

In the contract Pool.sol the role `factory.owner` has authority over the function `claimProtocolFees()`.

Any compromise to the `owner` account 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.

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.

Alleviation

[Maverick Team]:

The owner will be a DAO contract.

PCK-02 | Missing Router Restrictions On `Pool.addLiquidity()` And `Pool.removeLiquidity()`

Category	Severity	Location	Status
Control Flow	Major	models/Pool.sol: 232, 275	Acknowledged

Description

Our current understanding is that the `addLiquidity()` and `removeLiquidity()` functions in Pool.sol should be only called by the Router contract, such that the related token transferring logics are successfully triggered in Router. The `addLiquidity()` and `removeLiquidity()` functions in Router.sol are the actual accessible points for external users to interact with.

If `Pool.removeLiquidity()` is meant to be called by external users, then the balance of base and quote tokens should be that of a `msg.sender` instead of that of the contract address (`address(this)`).

In addition, there are some possible math simplifications, like in `addLiquidity()`, the `quoteFeeEscrow` is added to `totalQuoteBalance`, subtracted when calculating `quoteIn`, and finally used added with `quoteIn` again in the function call of `_setLiquidity()`

Recommendation

Recommend adding proper access control of these two functions in Pool.sol, to restrict the direct access to external users. Also recommend properly documenting a safe method of interaction with the contracts.

Alleviation

The Maverick Team assumes the users interacting with the contract directly know what they are doing, which implies advance users would use scripts for interaction and normal users would use front end for interaction. Thus no one would separate two transactions and give the hackers a chance to front run their transactions.

PCK-03 | `Pool.initialize()` Can Be Called Multiple Times

Category	Severity	Location	Status
Logical Issue	Medium	models/Pool.sol: 121	Resolved

Description

In contract Pool, the function `initialize()` is possible to be called multiple times, if the input `_quote` is zero address.

Recommendation

Recommend adding some `require` checks to make sure the function cannot be called after the first valid call. For example, adding a check to make sure the input argument `_quote` is a non-zero address might be helpful.

Alleviation

Fixed in commit hash `a1e0bcd6dc2ef0b41c0987f92a155790cc4a50dc`.

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.

Mathematical Operations

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

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.