CERTIK

Security Assessment

1inch - Limit Order

Nov 25th, 2021

Table of Contents

Summary

This report has been prepared for 1inch to discover issues and vulnerabilities in the source code of the 1inch - Limit Order 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	1inch - Limit Order
Platform	ethereum
Language	Solidity
Codebase	https://github.com/1inch/limit-order-protocol
Commit	• fc3f3d6af6c03603df7ec149afcc7bb86a627646 • bf190982639e0408d0f8b1bbf4a874ff0810096c • 9b5b4cfa174c2ca3e4d4b2f6b08eb73d13f2eefd

Audit Summary

Delivery Date	Nov 25, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	

Vulnerability Summary

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

Audit Scope

ID	File	SHA256 Checksum
ACC	helpers/AmountCalculator.sol	193815b0f73cf2e4fc109897f587f9a22399ae829b1a3aaf48d81a46bddf4d63
CCC	helpers/ChainlinkCalculator.sol	ca034717629da3b53a5db377cd13722696fa967cc60d49d9ed023f8389f3227e
ERC	helpers/ERC1155Proxy.sol	87599a0aeb6bde8bc9fa4c93184fc7f2f89f89719342cac256c42167f77374f2
ERP	helpers/ERC721Proxy.sol	b8fd60b0d0e3e33b48d00c1c5a37de771844c71b7235639958464836fafe066f
ERS	helpers/ERC721ProxySafe.sol	6dc8954e3ef892f3e8a9e5480db7b1b74c294f520e0f86b91a5fa5dafffc4581
IOC	helpers/ImmutableOwner.sol	4dd0eefcca2b4c029467afce8081e016e2e1ea4d802df247c4158250489d9827
NMC	helpers/NonceManager.sol	8a095c998af1df18a25ea1f4e1a152842d22d691b49618fe0812af83106c7b48
PHC	helpers/PredicateHelper.sol	db314568a0d9757a884a1ff3f76481f05fc47714e40f25e094e67287e3075176
ADC	libraries/ArgumentsDecoder.sol	7f3276bde678376da813d0d7efe70daf059c78d76dc6f6c23303afcee9dbf06c
PCK	libraries/Permitable.sol	7b65219c91eee0c043a08169a28dfab2483660e15b6ee6719402f14aa287f008
RRP	libraries/RevertReasonParser.sol	91311b20562ffccdef5459a345671a9be55b28bfafd07c6d9b1257afdf3fc791
LOP	LimitOrderProtocol.sol	86f81cefee9df58e2fb94cfb5f635fe2dde8db10f6c1d00ca9b7faf83e0b23ec
OMC	OrderMixin.sol	05c51bd187ba3af0a70bd8c1e1ec76e23f05587c588ee96df86911e78b730f47
ORF	OrderRFQMixin.sol	5ab37879b3fda9e37dbb78b36525e5a5b5c016c1078ef5f6b0ba08dfa603e08e

Understandings

Overview

1inch limit order protocol is a set of smart contracts that can work on any EVM-based blockchain (Ethereum, Binance Smart Chain, Polygon, etc.). Key features of the protocol are extreme flexibility and high gas efficiency that is achieved by using two different order types - regular Limit Orders and RFQ Orders.

Dependencies

There are a few depending injection contracts or addresses in the current project:

- `targets[i]`, `order.maker` , `order.makerAsset`, `order.takerAsset`, `order.receiver`, `order.allowedSender`, `token` decoded from `permit`, `interactionTarget` decoded from `order.interaction`, and `target` for the contract `OrderMixin`;
- `order.maker` , `order.makerAsset`, `order.takerAsset`, `order.allowedSender`, and `target` for the contract `OrderRFQMixin`;
- `target` for the contract `AmountCalculator`;
- `oracle`, `oracle1`, and `oracle2` for the contract `ChainlinkCalculator`;
- `IERC721 token` for the contract `ERC721Proxy`;
- `IERC721 token` for the contract `ERC721ProxySafe`;
- `IERC1155 token` for the contract `ERC1155Proxy`;
- `targets[i]` and `target` for the contract `PredicateHelper`;
- `token` for the contract `Permitable`.

We assume these contracts or addresses are valid and non-vulnerable actors and implement proper logic to collaborate with the current project.

Privileged Functions

In the contract `ERC721Proxy`, the role `immutableOwner` has the authority over the following function:

- `func_602HzuS()`, which transfers NFT tokens in an ERC721 contract.

In the contract `ERC721ProxySafe`, the role `immutableOwner` has the authority over the following function:

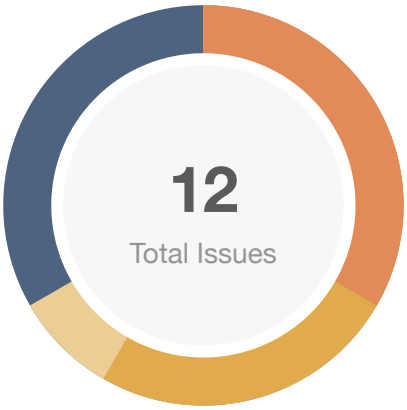
- `func_602HzuS()`, which transfers NFT tokens in an ERC721 contract.

In the contract `ERC1155Proxy`, the role `immutableOwner` has the authority over the following function:

- `func_301JL5R()`, which transfers tokens in an ERC1155 contract.

To improve the trustworthiness of the project, dynamic runtime updates in the project should be notified to the community. Any plan to invoke the aforementioned functions should be also considered to move to the execution queue of the `Timelock` contract.

Findings



Critical	0 (0.00%)
Major	4 (33.33%)
Medium	3 (25.00%)
Minor	1 (8.33%)
Informational	4 (33.33%)
Discussion	0 (0.00%)

ID	Title	Category	Severity	Status
ACC-01	Users May Lose Money on Partial Order Fills	Logical Issue	Informational	Resolved
ADC-01	Functionality of `decodeBool()`	Logical Issue	Minor	Resolved
ERC-01	Centralization Risk	**Centralization / Privilege**	**Major**	Acknowledged
ERP-01	Centralization Risk	**Centralization / Privilege**	**Major**	Acknowledged
ERS-01	Centralization Risk	**Centralization / Privilege**	**Major**	Acknowledged
OMC-01	The `_callGetter()` Function Can Return Incorrect Values	Logical Issue	Major	Resolved
OMC-02	Incorrect Variable Used	Logical Issue	Medium	Resolved
OMC-03	Potential Reentrancy Attack	Logical Issue	Medium	Acknowledged
OMC-04	Misspelled Error Message	Coding Style	Informational	Resolved
ORF-01	Potential Reentrancy Attack	Logical Issue	Medium	Acknowledged
ORF-02	Missing Emit Events	Coding Style	Informational	Resolved
ORF-03	Gas Inefficiency of Validation	Gas Optimization	Informational	Resolved

ACC-01 | Users May Lose Money on Partial Order Fills

Category	Severity	Location	Status
Logical Issue	Informational	projects/1inch_limit-order-protocol/contracts/helpers/AmountCalculator.sol (b478c1b): 14, 20	Resolved

Description

The `getMakerAmount()` function gives the floor of `swapTakerAmount * orderMakerAmount / orderTakerAmount`, while `getTakerAmount()` gives the ceiling of `swapMakerAmount * orderTakerAmount / orderMakerAmount`. For users that partially fill orders with small amounts, they may potentially lose money.

For example, suppose `orderMakerAmount = 10` and `orderTakerAmount = 3`, so the ratio between maker tokens and taker tokens is 10:3.

If a user planned to exchange 1 taker token, then they would receive `getMakerAmount(10,3,1) = 3` maker tokens, due to a floor rounding, resulting in a loss of 0.33 maker tokens.

If a user instead wished to obtain 1 maker token, then they would have to pay `getTakerAmount(10,3,1) = 1` taker token, due to the ceiling rounding, resulting in an overpay of 0.7 taker tokens.

Alleviation

[1inch Team]: That’s intended behavior. Taker can control the rounding.

ADC-01 | Functionality of `decodeBool()`

Category	Severity	Location	Status
Logical Issue	Minor	projects/1inch_limit-order-protocol/contracts/libraries/ArgumentsDecoder.sol (b478c1b): 14	Resolved

Description

The `decodeBool()` function does not check if the bytes input is 0 or 1, but rather if it is non-zero. Any non-zero data will be decoded into `true`.

Alleviation

The 1inch Team resolved this issue by changing the `decodeBool()` function to check whether or not the decoded value is 1 in commit 908926a004af4da8c7f6966f652424500f2ac046.

ERC-01 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	**Major**	projects/1inch_limit-order-protocol/contracts/helpers/ERC1155Proxy.sol (b478c1b): 21	Acknowledged

Description

In the contract `ERC1155Proxy`, the role `immutableOwner` has the authority over the following function:

- `func_301JL5R()`, which transfers tokens in an ERC1155 contract.

Any compromise to the `immutableOwner` account may allow the hacker to take advantage of this and gain unauthorized access to token transfers.

Recommendation

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

Here are some feasible suggestions that would also mitigate this risk in the short-term and long-term:

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

Alleviation

[1inch Team]: Proxy owner will be the `LimitOrderProtocol` itself.

[CertiK]: The auditors agree that if the `immutableOwner` is the `LimitOrderProtocol` contract, there will not be risks on the `immutableOwner` account's private key. However, considering the auditors do not know if the deployment will proceed correctly, the status of this issue will be updated after contract deployment upon request.

ERP-01 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	**Major**	projects/1inch_limit-order-protocol/contracts/helpers/ERC721Proxy.sol (b478c1b): 22	Acknowledged

Description

In the contract `ERC721Proxy`, the role `immutableOwner` has the authority over the following function:

- `func_602HzuS()`, which transfers NFT tokens in an ERC721 contract.

Any compromise to the `immutableOwner` account may allow the hacker to take advantage of this and gain unauthorized access to token transfers.

Recommendation

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

Here are some feasible suggestions that would also mitigate this risk in the short-term and long-term:

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

Alleviation

[1inch Team]: Proxy owner will be the `LimitOrderProtocol` itself.

[CertiK]: The auditors agree that if the `immutableOwner` is the `LimitOrderProtocol` contract, there will not be risks on the `immutableOwner` account's private key. However, considering the auditors do not know if the deployment will proceed correctly, the status of this issue will be updated after contract deployment upon request.

ERS-01 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	**Major**	projects/1inch_limit-order-protocol/contracts/helpers/ERC721ProxySafe.sol (b478c1b): 22	Acknowledged

Description

In the contract `ERC721ProxySafe`, the role `immutableOwner` has the authority over the following function:

- `func_602HzuS()`, which transfers NFT tokens in an ERC721 contract.

Any compromise to the `immutableOwner` account may allow the hacker to take advantage of this and gain unauthorized access to token transfers.

Recommendation

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

Here are some feasible suggestions that would also mitigate this risk in the short-term and long-term:

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

Alleviation

[1inch Team]: Proxy owner will be the `LimitOrderProtocol` itself.

[CertiK]: The auditors agree that if the `immutableOwner` is the `LimitOrderProtocol` contract, there will not be risks on the `immutableOwner` account's private key. However, considering the auditors do not know if the deployment will proceed correctly, the status of this issue will be updated after contract deployment upon request.

OMC-01 | The `_callGetter()` Function Can Return Incorrect Values

Category	Severity	Location	Status
Logical Issue	Major	projects/1inch_limit-order-protocol/contracts/OrderMixin.sol (b478c1b)	Resolved

Description

The `_callGetter()` function on line 330 is meant to return a `makerAmount` when given `takerAmount` data and similarly return a `takerAmount` when given `makerAmount` data.

However, if `getter.length == 0`, it actually returns an amount that is of the same type as the input.

```
 330     function _callGetter(bytes memory getter, uint256 orderAmount, uint256 amount)
private view returns(uint256) {
 331         if (getter.length == 0) {
 332             // On empty getter calldata only exact amount is allowed
 333             require(amount == orderAmount, "LOP: wrong amount");
 334             return orderAmount;
```

Recommendation

We recommend revisiting the logic of this function.

Alleviation

The 1inch Team heeded our advice and resolved this issue by using the correct variable in commit 9b5b4cfa174c2ca3e4d4b2f6b08eb73d13f2eefd.

OMC-02 | Incorrect Variable Used

Category	Severity	Location	Status
Logical Issue	Medium	projects/1inch_limit-order-protocol/contracts/OrderMixin.sol (b478c1b): 280	Resolved

Description

In the function `fillOrderTo()`, the function makes a call to `order.makerAsset`.

```
273         _makeCall(
274             order.makerAsset,
275             abi.encodePacked(
276                 IERC20.transferFrom.selector,
277                 uint256(uint160(order.maker)),
278                 uint256(uint160(target)),
279                 makingAmount,
280                 order.makerAsset
281             )
282         );
```

However, if we understood the `Order` struct correctly, the `order.makerAsset` at the end of the `abi.encodePacked` should instead be `order.makerAssetData`.

Recommendation

We recommend fixing the typo by changing `order.makerAsset` in the `abi.encodePacked` to `order.makerAssetData`.

Alleviation

The 1inch Team heeded our advice and resolved this issue by using the correct variable in commit cace3ec8f1cb58d2938417bdcfa47a4eea41d8a0.

OMC-03 | Potential Reentrancy Attack

Category	Severity	Location	Status
Logical Issue	Medium	projects/1inch_limit-order-protocol/contracts/OrderMixin.sol (b478c1b): 190~197	Acknowledged

Description

A reentrancy attack can occur when the contract creates a function that makes an external call to another untrusted contract before resolving any effects. If the attacker can control the untrusted contract, they can make a recursive call back to the original function, repeating interactions that would have otherwise not run after the external call resolved the effects.

Recommendation

We recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts or applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

Alleviation

[1inch Team]: The only place with reentrancy is the permit call that happens when `order.permit` is not empty. We have reentrancy checks there. Other places follow the checks-effects-interactions pattern.

[Certik]: We agree that there is a reentrancy check on `order.permit`. However, the contract performs other calls, such as to `order.takerAsset` (line 253), `order.makerAsset` (line 274), and `interactionTarget` (line 267). If these are malicious, they can reenter and bypass the reentrancy check on `order.permit`.

OMC-04 | Misspelled Error Message

Category	Severity	Location	Status
Coding Style	Informational	projects/1inch_limit-order-protocol/contracts/OrderMixin.sol (b478c1b): 202	Resolved

Description

One of the error messages in the function `fillOrderTo()` is misspelled.

```
202             require(remainingMakerAmount != 1, "LOP: remaining amoint is 0");
```

The error message should instead be "LOP: remaining amount is 0".

Recommendation

We recommend correcting the spelling mistake.

Alleviation

The 1inch Team heeded our advice and resolved this issue by fixing the spelling mistake in commit f6dad7b99c6864acb2236abe954159d37ce9eba3.

ORF-01 | Potential Reentrancy Attack

Category	Severity	Location	Status
Logical Issue	Medium	projects/1inch_limit-order-protocol/contracts/OrderRFQMixin.sol (b478c1b): 91~97	Acknowledged

Description

A reentrancy attack can occur when the contract creates a function that makes an external call to another untrusted contract before resolving any effects. If the attacker can control the untrusted contract, they can make a recursive call back to the original function, repeating interactions that would have otherwise not run after the external call resolved the effects.

Recommendation

We recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts or applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

Alleviation

[1inch Team]: The code already follows the checks-effects-interactions pattern.

[Certik Team]: We agree the code follows the checks-effects-interactions pattern for variable states in the contract. However, two transfers are performed on lines 144 and 145 and it is possible for the first transfer to re-enter before changes to the chain state from the second transfer.

ORF-02 | Missing Emit Events

Category	Severity	Location	Status
Coding Style	Informational	projects/1inch_limit-order-protocol/contracts/OrderRFQMixin.sol (b478c1b): 45	Resolved

Description

The following function affects the status of sensitive variables and should be able to emit events as notifications:

- `OrderRFQMixin.cancelOrderRFQ()` to cancel an RFQ order.

Recommendation

We recommend emitting events for all the essential state variables that are possible to be changed during the runtime.

Alleviation

[1inch Team]: RFQ orders are intended to be used by market makers who probably have their own sophisticated order tracking tools set up. And in that case they don't need specific event to track cancelation of their own order. Moreover we've seen no single use of RFQ canceling so far. Thus we think that there is no need in emitting cancel event for RFQ orders.

ORF-03 | Gas Inefficiency of Validation

Category	Severity	Location	Status
Gas Optimization	Informational	projects/1inch_limit-order-protocol/contracts/OrderRFQMixin.sol (b478c1b): 139~141	Resolved

Description

In the function `fillOrderRFQTo()`, the function checks whether the order is valid near the end of the function call.

```
139         require(order.allowedSender == address(0) || order.allowedSender ==
msg.sender, "LOP: private order");
140         bytes32 orderHash =
_hashTypedDataV4(keccak256(abi.encode(LIMIT_ORDER_RFQ_TYPEHASH, order)));
141         require(SignatureChecker.isValidSignatureNow(maker, orderHash, signature),
"LOP: bad signature");
```

It might be more optimal to locate these checks at the start of the function call so extraneous calculations are not performed.

Recommendation

We recommend moving these checks to the beginning of the function call.

Alleviation

The 1inch Team heeded our advice and resolved this issue by and have moved the validation checks to the beginning of the function in commit bf190982639e0408d0f8b1bbf4a874ff0810096c.

Appendix

Finding Categories

Centralization / Privilege

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

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Logical Issue

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

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.