CERTIK

Security Assessment

Empire Token

Nov 4th, 2021

Table of Contents

Summary

This report has been prepared for Empire Token to discover issues and vulnerabilities in the source code of the Empire Token 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.

Additionally, this audit is based on a premise that all external contracts were implemented safely.

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	Empire Token
Platform	BSC
Language	Solidity
Codebase	https://bscscan.com/address/0x293c3ee9abacb08bb8ced107987f00efd1539288#code
Commit	

Audit Summary

Delivery Date	Nov 04, 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	3	0	0	3	0	0
Medium	0	0	0	0	0	0
Minor	5	0	0	5	0	0
Informational	10	0	0	10	0	0
Discussion	0	0	0	0	0	0

Audit Scope

ID	File	SHA256 Checksum
EET	Empire.sol	95bfa03c79a74e510e81d89971e084b977ea778dc62d364a4d135b164c34efd3

Understandings

Overview

The Empire Protocol is a decentralized finance (DeFi) token deployed on the Binance smart chain(BSC). Empire employs two novel features in its protocol; static rewards for each user as well as an LP acquisition mechanism. The static reward (also known as reflection) and LP acquisition mechanisms function as follows: Each Empire transaction is taxed 4.5% and 5.5% fees totaling 10% of the transaction amount. The part of the first fee is redistributed to all existing holders using a form of rebasing mechanism whilst the rest will be distributed to a devWallet. Besides, a 4.5% fee is accumulated internally until a sufficient amount of capital has been amassed to perform an LP acquisition. When this number is reached, the total tokens accumulated are split with half being converted to ETH and the total being supplied to the dex contract as liquidity. The fees mentioned above can be set in the contract.

Privileged Functions

The contract contains the following privileged functions that are restricted by some modifiers. They are used to modify the contract configurations and address attributes. We grouped these functions below:

The `onlyOwner` modifier:

Contract `Ownable`:

- renounceOwnership()
- transferOwnership(address newOwner)
- lock(uint256 time)

Contract `Empire`:

- setNumTokensToSellForLiquidity(uint256 _numtokens)
- setDevWallet(address _devWallet)
- updateRouter(address _router)
- setNotReflectionFraction(uint256 _nr)
- excludeFromReward(address account)
- includeInReward(address account)
- excludeFromFee(address account)
- includeInFee(address account)
- setTaxFeePercent(uint256 taxFee)
- setLiquidityFeePercent(uint256 liquidityFee)

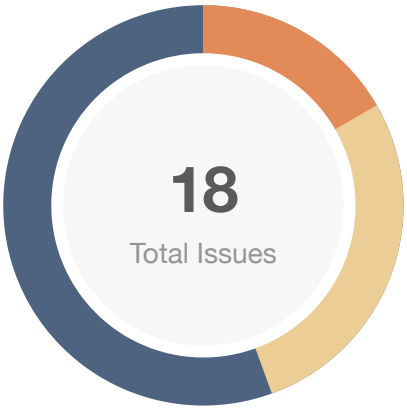
- setMaxTxPercent(uint256 maxTxPercent)
- setSwapAndLiquifyEnabled(bool _enabled)
- beforeListing()
- afterListing()
- enableBuying()

The `lockTheSwap` modifier:

Contract `Empire`:

- swapAndLiquify(uint256 contractTokenBalance)

Findings



Critical	**0** (0.00%)
Major	**3** (16.67%)
Medium	**0** (0.00%)
Minor	**5** (27.78%)
Informational	**10** (55.56%)
Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
GLOBAL-01	Third Party Dependencies	Volatile Code	Minor	Acknowledged
GLOBAL-02	Unlocked Compiler Version	Language Specific	Informational	Acknowledged
GLOBAL-03	Centralization Risk	**Centralization / Privilege**	**Major**	Acknowledged
GLOBAL-04	Function Visibility Optimization	Gas Optimization	Informational	Acknowledged
GLOBAL-05	Missing Emit Events	Coding Style	Informational	Acknowledged
EET-01	Possible to gain ownership after renouncing the contract ownership	Logical Issue	Minor	Acknowledged
EET-02	Variables Could Be Declared `Constant` or `Immutable`	Gas Optimization	Informational	Acknowledged
EET-03	The purpose of function `deliver`	Control Flow	Informational	Acknowledged
EET-04	Incorrect error message	Logical Issue	Minor	Acknowledged
EET-05	Potential Sandwich Attacks	Logical Issue	Minor	Acknowledged
EET-06	Redundant code	Logical Issue	Informational	Acknowledged
EET-07	Missing Input Validation	Logical Issue	Informational	Acknowledged
EET-08	Duplicate Deduction Of Fee	Logical Issue	Minor	Acknowledged

ID	Title	Category	Severity	Status
EET-09	Contract Gains Non-withdrawable ETH Via The `swapAndLiquify` Function	Logical Issue	Major	Acknowledged
EET-10	Return Value Not Handled	Volatile Code	Informational	Acknowledged
EET-11	Centralized Risk In `addLiquidity`	**Centralization / Privilege**	**Major**	Acknowledged
EET-12	Typos In The Contract	Coding Style	Informational	Acknowledged
EET-13	Function Name Typo	Coding Style	Informational	Acknowledged

GLOBAL-01 | Third Party Dependencies

Category	Severity	Location	Status
Volatile Code	Minor	Global	Acknowledged

Description

The scope of the audit treats 3rd party entities as black boxes and assumes 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 encourage the team to constantly monitor the statuses of 3rd parties to mitigate the side effects when unexpected activities are observed.

Alleviation

Customer team response:
Acknowledged - team will monitor any breaking change into third parties before new implementation on Empire.

GLOBAL-02 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	Informational	Global	Acknowledged

Description

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

- Empire.sol

Recommendation

We advise that the compiler version is alternatively 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

Customer team response:
Acknowledged, due to the low risk for our users, team will only fix in future implementation.

GLOBAL-03 | Centralization Risk

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

Description

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

- renounceOwnership()
- transferOwnership()
- lock()

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

- setNumTokensToSellForLiquidity()
- setDevWallet()
- updateRouter()
- setNotReflectionFraction()
- excludeFromReward()
- includeInReward()
- excludeFromFee()
- includeInFee()
- setTaxFeePercent()
- setLiquidityFeePercent()
- setMaxTxPercent()
- setSwapAndLiquifyEnabled()
- beforeListing()
- afterListing()
- enableBuying()

Any compromise to these accounts may allow the hacker to manipulate the project through these functions.

Recommendation

We advise the client to carefully manage the `owner` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be

improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

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

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

Alleviation

Customer team response:
Ownership will be transferred to the Gnosis Multisig.

GLOBAL-04 | Function Visibility Optimization

Category	Severity	Location	Status
Gas Optimization	Informational	Global	Acknowledged

Description

The following functions are declared as `public` and are not invoked in any of the contracts contained within the project's scope. The functions that are never called internally within the contract should have external visibility.

contract `Ownable`

- `renounceOwnership()` in L438
- `transferOwnership()` in L447
- `geUnlockTime()` in L453
- `lock()` in L458
- `unlock()` in L466

contract `Empire`

- `setNumTokensToSellForLiquidity()` in L760
- `setDevWallet()` in L763
- `updateRouter()` in L797
- `approve()` in L807
- `transferFrom()` in L812
- `increaseAllowance()` in L818
- `decreaseAllowance()` in L823
- `deliver()` in L836
- `setNotReflectionFraction()` in L844
- `excludeFromReward()` in L864
- `excludeFromFee()` in L906
- `includeInFee()` in L910
- `setSwapAndLiquifyEnabled()` in L929
- `beforeListing()` in L1022
- `afterListing()` in L1029
- `enableBuying()` in L1036

Recommendation

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

Alleviation

Customer team response:
Acknowledged, due to the low risk for our users, team will only fix in future implementation.

GLOBAL-05 | Missing Emit Events

Category	Severity	Location	Status
Coding Style	Informational	Global	Acknowledged

Description

The function that affects the status of sensitive variables should be able to emit events as notifications to customers.

contract `Empire`

- `setNumTokensToSellForLiquidity()`
- `setDevWallet()`
- `updateRouter()`
- `deliver()`
- `setNotReflectionFraction()`
- `excludeFromReward()`
- `includeInReward()`
- `setTaxFeePercent()`
- `setLiquidityFeePercent()`
- `setMaxTxPercent()`
- `enableBuying()`

Recommendation

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

Alleviation

Customer team response:
Acknowledged, due to the low risk for our users, team will only fix in future implementation.

EET-01 | Possible to gain ownership after renouncing the contract ownership

Category	Severity	Location	Status
Logical Issue	Minor	Empire.sol: 438~471	Acknowledged

Description

An owner has the possibility to gain ownership of the contract even if he calls function `renounceOwnership` to renounce the ownership. This can be achieved by performing the following operations:

1. Call `lock` to lock the contract. The variable `_previousOwner` is set to the current owner.
2. Call `unlock` to unlock the contract.
3. Call `renounceOwnership` to leave the contract without an owner.
4. Call `unlock` to regain ownership.

Recommendation

We advise updating/removing `lock` and `unlock` functions in the contract; or removing the `renounceOwnership` if such a privilege retains at the protocol level. If timelock functionality could be introduced, we recommend using the implementation of Compound finance as reference. Reference: https://github.com/compound-finance/compound-protocol/blob/master/contracts/Timelock.sol

Alleviation

Customer team response:
Ownership will be transfered to the Gnosis Multisig.

EET-02 | Variables Could Be Declared `Constant` or `Immutable`

Category	Severity	Location	Status
Gas Optimization	Informational	Empire.sol: 703, 704, 705	Acknowledged

Description

Variables `_name`, `_symbol` and `_decimals` could be declared as `constant` since these state variables are never to be changed.

Recommendation

We recommend declaring those variables as `constant`.

Alleviation

Customer team response:
Acknowledged, due to the low risk for our users, team will only fix in future implementation.

EET-03 | The purpose of function `deliver`

Category	Severity	Location	Status
Control Flow	Informational	Empire.sol: 836	Acknowledged

Description

The function `deliver` can be called by anyone. It accepts an uint256 number parameter `tAmount`. The function reduces the token balance of the caller by `rAmount`, which is `tAmount` reduces the transaction fee. Then, the function adds `tAmount` to variable `_tFeeTotal`, which represents the contract's total transaction fee.

Recommendation

We wish the team could explain more on the purpose of having such functionality.

Alleviation

Customer team response:
This function allow an address to give back it’s reflected token and increase reward for others.

EET-04 | Incorrect error message

Category	Severity	Location	Status
Logical Issue	Minor	Empire.sol: 876	Acknowledged

Description

The error message in `require(_isExcluded[account], "Account is already excluded")` does not describe the error correctly.

Recommendation

The message "Account is already excluded" can be changed to "Account is not excluded".

Alleviation

Customer team response:
Acknowledged, due to the low risk for our users, team will only fix in future implementation.

EET-05 | Potential Sandwich Attacks

Category	Severity	Location	Status
Logical Issue	Minor	Empire.sol: 1126~1132, 1140~1147	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 restrictions on slippage or minimum output amount, so transactions triggering these functions are vulnerable to sandwich attacks, especially when the input amount is large:

- `swapTokensForEth()`
- `addLiquidity()`

Recommendation

We recommend setting reasonable minimum output amounts, instead of 0, based on token prices when calling the aforementioned functions.

Alleviation

Customer team response:
Acknowledged, due to the lack of impact on our user trades, will be fixed in future version, in the meantime, team will track and compensate potential attack on the swap&liquify events.

EET-06 | Redundant code

Category	Severity	Location	Status
Logical Issue	Informational	Empire.sol: 1159	Acknowledged

Description

The condition `!_isExcluded[sender] && !_isExcluded[recipient]` can be included in `else` .

Recommendation

The following code can be removed:

```
... else if (!_isExcluded[sender] && !_isExcluded[recipient]) {
    _transferStandard(sender, recipient, amount);
} ...
```

Alleviation

Customer team response:

Acknowledged, due to the low risk for our users, team will only fix in future implementation.

EET-07 | Missing Input Validation

Category	Severity	Location	Status
Logical Issue	Informational	Empire.sol: 739, 763, 797	Acknowledged

Description

The given input is missing the sanity check for non-zero address in the aforementioned line.

Recommendation

We recommend adding the check for the passed-in values to prevent unexpected error as below:
constructor():

```
739  require(_devWallet != address(0), "_devWallet address cannot be 0");
740  require(_router != address(0), "_router address cannot be 0");
```

setDevWallet():

```
763  require(_devWallet != address(0), "_devWallet address cannot be 0");
```

updateRouter():

```
797  require(_router != address(0), "_router address cannot be 0");
```

Alleviation

Customer team response:
Business use of devWallet = address(0). (ie creating an auto-burn) and we acknowledged, due to the low risk for our users, team will only fix in future implementation.

EET-08 | Duplicate Deduction Of Fee

Category	Severity	Location	Status
Logical Issue	Minor	Empire.sol: 854	Acknowledged

Description

`rTransferAmount` has already been deducted and here is deducted again.

Recommendation

We recommend to modify as follows:

```
854  return rTransferAmount;
```

Alleviation

Customer team response:

Acknowledged, due to the low risk for our users, team will only fix in future implementation.

EET-09 | Contract Gains Non-withdrawable ETH Via The `swapAndLiquify` Function

Category	Severity	Location	Status
Logical Issue	Major	Empire.sol: 1094	Acknowledged

Description

The `swapAndLiquify` function converts half of the `contractTokenBalance` Empire tokens to ETH. The other half of Empire tokens and part of the converted ETH are deposited into the Empire-ETH pool on uniswap as liquidity. For every `swapAndLiquify` function call, a small amount of ETH leftover in the contract. This is because the price of Empire drops after swapping the first half of Empire tokens into ETHs, and the other half of Empire tokens require less than the converted ETH to be paired with it when adding liquidity. The contract doesn't appear to provide a way to withdraw those ETH, and they will be locked in the contract forever.

Recommendation

It's not ideal that more and more ETH are locked into the contract over time. The simplest solution is to add a `withdraw` function in the contract to withdraw ETH. Other approaches that benefit the Empire token holders can be:

- Distribute ETH to Empire token holders proportional to the amount of token they hold.
- Use leftover ETH to buy back Empire tokens from the market to increase the price of Empire.

Alleviation

Customer team response:
Acknowledged, due to the low risk for our users, team will only fix in future implementation.

EET-10 | Return Value Not Handled

Category	Severity	Location	Status
Volatile Code	Informational	Empire.sol: 1140~1147	Acknowledged

Description

The return values of function `addLiquidityETH` are not properly handled.

```
1140 uniswapV2Router.addLiquidityETH{value: ethAmount}(
1141     address(this),
1142     tokenAmount,
1143     0, // slippage is unavoidable
1144     0, // slippage is unavoidable
1145     owner(),
1146     block.timestamp
1147 );
```

Recommendation

We recommend using variables to receive the return value of the functions mentioned above and handle both success and failure cases if needed by the business logic.

Alleviation

Customer team response:
No business use-case in returned amounts at contract-level (off chain use to track potential sandwiches can easily be done via other means that is through subscribing and analyzing relevant events from pancakeswap).

EET-11 | Centralized Risk In `addLiquidity`

Category	Severity	Location	Status
Centralization / Privilege	**Major**	Empire.sol: 1145	Acknowledged

Description

```
// add the liquidity
uniswapV2Router.addLiquidityETH{value: ethAmount}(
    address(this),
    tokenAmount,
    0, // slippage is unavoidable
    0, // slippage is unavoidable
    owner(),
    block.timestamp
);
```

The `addLiquidity` function calls the `uniswapV2Router.addLiquidityETH` function with the `to` address specified as `owner()` for acquiring the generated LP tokens from the `Empire-WETH` pool. As a result, over time the `_owner` address will accumulate a significant portion of LP tokens.If the `_owner` is an EOA (Externally Owned Account), mishandling of its private key can have devastating consequences to the project as a whole.

Recommendation

We advise the `to` address of the `uniswapV2Router.addLiquidityETH` function call to be replaced by the contract itself, i.e. `address(this)`, and to restrict the management of the LP tokens within the scope of the contract's business logic. This will also protect the LP tokens from being stolen if the `_owner` account is compromised. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or via smart-contract based accounts with enhanced security practices, f.e. Multisignature wallets.

Indicatively, here are some feasible solutions that would also mitigate the potential risk:

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

Alleviation

Customer team response:

Ownership will be transfered to the Gnosis Multisig.

EET-12 | Typos In The Contract

Category	Severity	Location	Status
Coding Style	Informational	Empire.sol: 730, 934	Acknowledged

Description

There are several typos in the code and comments.

1. In the following code snippet, `tokensIntoLiqudity` should be `tokensIntoLiquidity`.

```
730 event SwapAndLiquify(
731     uint256 tokensSwapped,
732     uint256 ethReceived,
733     uint256 tokensIntoLiqudity
734 );
```

2. `recieve` should be `receive` and `swaping` should be `swapping` in the line of comment `//to recieve ETH from uniswapV2Router when swaping`.

Recommendation

We recommend correcting all typos in the contract.

Alleviation

Customer team response:

Acknowledged, due to the low risk for our users, team will only fix in future implementation.

EET-13 | Function Name Typo

Category	Severity	Location	Status
Coding Style	Informational	Empire.sol: 453	Acknowledged

Description

Function name is mistakenly set as `geUnlockTime()`.

Recommendation

We advise the client to fix the typo and set the correct name `getUnlockTime()` for the specific function.

Alleviation

Customer team response:

Acknowledged, due to the low risk for our users, team will only fix in future implementation.

Appendix

Finding Categories

Centralization / Privilege

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

Gas Optimization

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

Logical Issue

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

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

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

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

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

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

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

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

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

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

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

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

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.