Security Assessment

SafeMars

Aug 4th, 2021

Table of Contents

Summary

This report has been prepared for SafeMars to discover issues and vulnerabilities in the source code of the SafeMars 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	SafeMars
Platform	BSC
Language	Solidity
Codebase	https://bscscan.com/address/0x3ad9594151886ce8538c1ff615efa2385a8c3a88#code
Commit	

Audit Summary

Delivery Date	Aug 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	1	0	0	1	0	0
Medium	3	0	0	3	0	0
Minor	3	0	0	3	0	0
Informational	7	0	0	7	0	0
Discussion	0	0	0	0	0	0

Audit Scope

ID	File	SHA256 Checksum
CKP	safemars.sol	d0ef6e4af501d3cba32cb68b1fd7fa59793432c30ba1d4771b2fcc7971c644f1

Review Notes

Overview

The SafeMars Protocol is a decentralized finance (DeFi) token deployed on the Binance smart chain (BSC). SafeMars 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 SafeMars transaction is taxed two 2% fees totaling 4% of the transaction amount. The first fee is redistributed to all existing holders using a form of rebasing mechanism whilst the other 2% 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 BNB and the total being supplied to the PancakeSwap contract as liquidity.

LP Acquisition

The LP acquisition mechanism can be indirectly triggered by any normal transaction of the token as all transfers evaluate the set of conditions that trigger the mechanism. The main conditions of the mechanism are whether the sender is different than the LP pair and whether the accumulation threshold has been reached. Should these conditions be satisfied, the `swapAndLiquify` function is invoked with the current contract's SafeMars balance.

The `swapAndLiquify` function splits the contract's balance into two halves properly accounting for any truncation that may occur. The first half is swapped to BNB via the PancakeSwap Router using the SafeMars-BNB pair and thus temporarily driving the price of the SafeMars token down. Afterwards, the resulting BNB balance along with the remaining SafeMars balance is supplied to the SafeMars-BNB liquidity pool as liquidity via the Router. The recipient of the LP units is defined as the current `owner` of the SafeMars contract, a characteristic outlined in more depth within finding CKP-01.

Static Reward (Reflection)

Balances in the SafeMars token system are calculated in one of two ways. The first method, which most users should be familiar with, is a traditional fixed number of units being associated with a user's address. The second method, which is of interest to static rewards, represents a user's balance as a proportion of the total supply of the token. This method works similarly to how dynamic rebasing mechanisms work such as that of Ampleforth.

Whenever a taxed transaction occurs, the 2% meant to be re-distributed to token holders is deducted from the total "proportion" supply resulting in a user's percentage of total supply being increased. Within the system, not all users are integrated into this system and as such the 2% fee is rewarded to a subset of the

total users of the SafeMars token. The `owner` of the contract is able to include and exclude users from the dynamic balance system at will.

Privileged Functions

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

Account management functions for inclusion and exclusion in the fee and reward system:

- function excludeFromReward(address account)
- function includeInReward(address account)
- function excludeFromFee(address account)
- function includeInFee(address account)

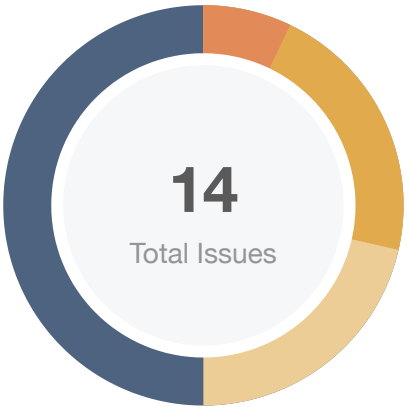
Modification of liquidation, tax and max transaction percents, and trading status of the system:

- function setTaxFeePercent(uint256 taxFee)
- function setLiquidityFeePercent(uint256 liquidityFee)
- function setMaxTxPercent(uint256 maxTxPercent)
- function enableTrading()

Toggle feature of the LP acquisition mechanism:

- function setSwapAndLiquifyEnabled(bool _enabled)

Findings



Critical	0 (0.00%)
Major	1 (7.14%)
Medium	3 (21.43%)
Minor	3 (21.43%)
Informational	7 (50.00%)
Discussion	0 (0.00%)

ID	Title	Category	Severity	Status
CKP-01	Centralized Risk in `addLiquidity`	**Centralization / Privilege**	**Major**	Acknowledged
CKP-02	Contract Gains Non-withdrawable BNB via the `swapAndLiquify` Function	Logical Issue	Medium	Acknowledged
CKP-03	Centralization Risk	**Centralization / Privilege**	**Medium**	Acknowledged
CKP-04	Possible to Gain Ownership after Renouncing the Contract Ownership	**Logical Issue, Centralization / Privilege**	**Medium**	Acknowledged
CKP-05	Incorrect Error Message	Logical Issue	Minor	Acknowledged
CKP-06	Third Party Dependencies	Control Flow	Minor	Acknowledged
CKP-07	Potential Sandwich Attack	Volatile Code	Minor	Acknowledged
CKP-08	Typos in the Contract	Coding Style	Informational	Acknowledged
CKP-09	Return Value Not Handled	Volatile Code	Informational	Acknowledged
CKP-10	Redundant Code	Logical Issue	Informational	Acknowledged
CKP-11	Missing Event Emitting	Coding Style	Informational	Acknowledged
CKP-12	Unused Event	Coding Style	Informational	Acknowledged
CKP-13	Variable Could Be Declared As `constant`	Gas Optimization	Informational	Acknowledged

ID	Title	Category	Severity	Status
CKP-14	Function and Variable Naming Doesn't Match the Operating Environment	Coding Style	Informational	Acknowledged

CKP-01 | Centralized Risk in `addLiquidity`

Category	Severity	Location	Status
Centralization / Privilege	Major	safemars.sol: 1048	Acknowledged

Description

The `addLiquidity` function calls the `uniswapV2Router.addLiquidityETH` function listed below with the `to` address specified as `owner()` for acquiring the generated LP tokens from the `SafeMars-BNB` 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.

```
1042  // add the liquidity
1043  uniswapV2Router.addLiquidityETH{value: ethAmount}(
1044      address(this),
1045      tokenAmount,
1046      0, // slippage is unavoidable
1047      0, // slippage is unavoidable
1048      owner(),
1049      block.timestamp
1050  );
```

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, e.g., Multisignature wallets.

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

- 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

[**SafeMars Team**]: The deployer regularly burns those LP tokens (send to zero address). So the risk is mitigated.

In fact, we have been taking advantage of this to actually move liquidity slowly from v1 to v2. As liquidity in v1 is locked forever (LP tokens burnt) we can't withdraw and move it to PancakeSwap v2. However, as new LP tokens are generated for the deployer, we remove that new liquidity and add it to v2.

[**CertiK**] The owner of the contract has been set to `0x2963B4311e07da3D9c50F2e1eb14659e42e3f4A9`. Relevant transactions can be checked on bscscan.

CKP-02 | Contract Gains Non-withdrawable BNB via the `swapAndLiquify` Function

Category	Severity	Location	Status
Logical Issue	Medium	safemars.sol: 997	Acknowledged

Description

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

Recommendation

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

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

Alleviation

[**SafeMars Team**]: There is nothing we can do about it.

CKP-03 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	Medium	safemars.sol: 791, 800, 823, 827, 831, 835, 839, 845, 850	Acknowledged

Description

With the modifier `onlyOwner`, the 'owner' has the authority to call the following sensitive functions to change settings of the `SAFEMARS` contract:

- `SAFEMARS.excludeFromReward(address account)`
- `SAFEMARS.includeInReward(address account)`
- `SAFEMARS.excludeFromFee(address account)`
- `SAFEMARS.includeInFee(address account)`
- `SAFEMARS.setTaxFeePercent(uint256 taxFee)`
- `SAFEMARS.setLiquidityFeePercent(uint256 liquidityFee)`
- `SAFEMARS.setMaxTxPercent(uint256 maxTxPercent)`
- `SAFEMARS.setSwapAndLiquifyEnabled(bool _enabled)`
- `SAFEMARS.enableTrading()`

Any compromise to the `owner` account may allow the hacker to adversarially manipulate the settings of the contract.

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 via smart-contract-based accounts with enhanced security practices, e.g. Multisignature wallets.

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

- Timelock 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

N/A

CKP-04 | Possible to Gain Ownership after Renouncing the Contract Ownership

Category	Severity	Location	Status
Logical Issue, Centralization / Privilege	**Medium**	safemars.sol: 418	Acknowledged

Description

An owner is possible to gain ownership of the contract even if he/she calls the 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 a reference.

Reference: https://github.com/compound-finance/compound-protocol/blob/master/contracts/Timelock.sol

Alleviation

N/A

CKP-05 | Incorrect Error Message

Category	Severity	Location	Status
Logical Issue	Minor	safemars.sol: 801	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" should be changed to "Account is not excluded" .

Alleviation

N/A

CKP-06 | Third Party Dependencies

Category	Severity	Location	Status
Control Flow	Minor	safemars.sol: 691	Acknowledged

Description

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

Recommendation

We understand that the business logic of the SafeMars protocol requires the interaction PancakeSwap protocol for adding liquidity to the SafeMars-BNB pool and swapping tokens. We encourage the team to constantly monitor the statuses of those third-party entities to mitigate the side effects when unexpected activities are observed.

Alleviation

N/A

CKP-07 | Potential Sandwich Attack

Category	Severity	Location	Status
Volatile Code	Minor	safemars.sol: 1029~1035, 1043~1050	Acknowledged

Description

Potential sandwich attacks could happen if calling `uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens` and `uniswapV2Router.addLiquidityETH` without setting restrictions on slippage.

For example, when we want to make a transaction of swapping 100 AToken for 1 ETH, an attacker could raise the price of ETH by adding AToken into the pool before the transaction so we might only get 0.1 ETH. After the transaction, the attacker would be able to withdraw more than he deposited because the total value of the pool increases by 0.9 ETH.

Recommendation

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

Alleviation

N/A

CKP-08 | Typos in the Contract

Category	Severity	Location	Status
Coding Style	Informational	safemars.sol: 413, 679	Acknowledged

Description

The function name `geUnlockTime()` should be `getUnlockTime()`, and the variable name `tokensIntoLiqudity` should be `tokensIntoLiquidity`.

Recommendation

We recommend correcting these typos in the contract.

Alleviation

N/A

CKP-09 | Return Value Not Handled

Category	Severity	Location	Status
Volatile Code	Informational	safemars.sol: 1043~1050	Acknowledged

Description

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

```
1043        uniswapV2Router.addLiquidityETH{value: ethAmount}(
1044            address(this),
1045            tokenAmount,
1046            0, // slippage is unavoidable
1047            0, // slippage is unavoidable
1048            owner(),
1049            block.timestamp
1050        );
```

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

N/A

CKP-10 | Redundant Code

Category	Severity	Location	Status
Logical Issue	Informational	safemars.sol: 1063	Acknowledged

Description

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

Recommendation

The following code can be removed:

```
1062  ... else if (!_isExcluded[sender] && !_isExcluded[recipient]) {
1063      _transferStandard(sender, recipient, amount);
1064  } ...
```

Alleviation

N/A

CKP-11 | Missing Event Emitting

Category	Severity	Location	Status
Coding Style	Informational	safemars.sol: 831, 835, 839, 850, 823, 827	Acknowledged

Description

In contract `SAFEMARS`, there are a bunch of functions that can change state variables. However, these functions do not emit events to pass the changes out of the chain. Some example functions are listed below:

- `SAFEMARS.excludeFromFee(address)`
- `SAFEMARS.includeInFee(address)`
- `SAFEMARS.setTaxFeePercent(uint256)`
- `SAFEMARS.setLiquidityFeePercent(uint256)`
- `SAFEMARS.setMaxTxPercent(uint256)`
- `SAFEMARS.enableTrading()`

Recommendation

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

Alleviation

N/A

CKP-12 | Unused Event

Category	Severity	Location	Status
Coding Style	Informational	safemars.sol: 674	Acknowledged

Description

`MinTokensBeforeSwapUpdated` event is declared but never used.

Recommendation

We recommend removing `MinTokensBeforeSwapUpdated` event.

Alleviation

N/A

CKP-13 | Variable Could Be Declared As `constant`

Category	Severity	Location	Status
Gas Optimization	Informational	safemars.sol: 650, 654~656, 672	Acknowledged

Description

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

Recommendation

We recommend declaring those variables as `constant`.

Alleviation

N/A

CKP-14 | Function and Variable Naming Doesn't Match the Operating Environment

Category	Severity	Location	Status
Coding Style	Informational	safemars.sol	Acknowledged

Description

The SafeMars contract uses PancakeSwap for swapping and adding liquidity to the PancakeSwap pool, but naming it Uniswap. Function `SAFEMARS.swapTokensForEth(uint256)` swaps SafeMars token for BNB instead of ETH.

Recommendation

We recommend changing "Uniswap" and "ETH" to "PancakeSwap" and "BNB" in the contract respectively to match the operating environment and avoid confusion.

Alleviation

N/A

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.

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.