CERTIK

Security Assessment

Skylabs Staking

Aug 11th, 2022

Table of Contents

Summary

This report has been prepared for Skylabs Staking to discover issues and vulnerabilities in the source code of the Skylabs Staking 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	Skylabs Staking
Platform	BSC
Language	Solidity
Codebase	https://github.com/Vetter-ai/vsl-token
Commit	• Prelim: ff5de6d28a7b418d2e4b961b6c7f6f61ba963bcf • Review: b293df4d6afcb8568511227dab7b4b3f79a9e9b6

Audit Summary

Delivery Date	Aug 11, 2022 UTC
Audit Methodology	Static Analysis, Manual Review

Vulnerability Summary

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

Audit Scope

ID	File	SHA256 Checksum
CKP	vsl-token-main/vslstaking.sol	505ce6efea86df312c30f36d20662ca7b6cd115ae15f0eae0b0bd9f9ed6e75bd

Findings

Critical	**0** (0.00%)
Major	**1** (14.29%)
Medium	**0** (0.00%)
Minor	**3** (42.86%)
Informational	**3** (42.86%)
Discussion	**0** (0.00%)

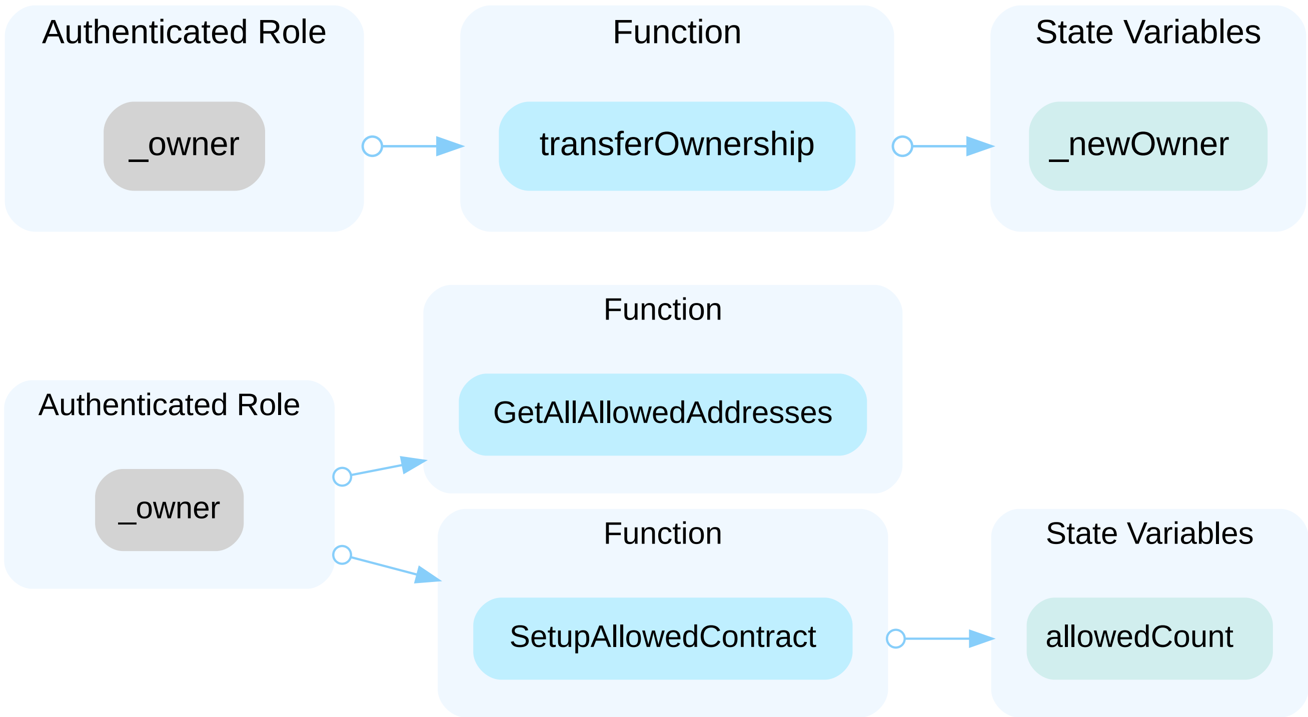
ID	Title	Category	Severity	Status
CKP-01	Centralization Related Risks	**Centralization / Privilege**	**Major**	Acknowledged
CKP-02	Weak PRNG	Volatile Code	Minor	Acknowledged
CKP-03	Quiet Exit On Unstake Failure	Coding Style	Minor	Resolved
CKP-04	Lack Of Validation On Variable `dbID`	Coding Style, Volatile Code	Minor	Acknowledged
CKP-06	Recommended Usage Of *Require*	Coding Style	Informational	Acknowledged
CKP-07	Third Party Dependencies	Volatile Code	Informational	Acknowledged
CKP-08	Missing Emit Events	Coding Style	Informational	Resolved

CKP-01 | Centralization Related Risks

Category	Severity	Location	Status
Centralization / Privilege	**Major**	vsl-token-main/vslstaking.sol: 73, 275, 306, 349, 764, 814, 1351, 1403, 1409, 1422, 1437, 1467, 1490, 1504, 1516, 1528, 1621, 1639, 1652, 1660, 1674, 1690, 1711, 1718	ⓘ Acknowledged

Description

In the contract `Ownable` the role `_owner` has authority over the function `SetupAllowedContract()`, `GetAllAllowedAddresses()` and `transferOwnership()` to assign privileged role to any address. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and take control over the assignment of privileged roles.



Meanwhile, the addresses in mapping `_allowedContract` can change most of the important state variables and transfer the tokens within the contract. They are crucial to the distribution of staking rewards, as they can call `DistributeStakingRewards()` to generate a new reward distribution, or call `TransferInternalAmount()` to withdraw the reward amount and penalty amount.

Specifically, these addresses have authority over the following functions:

- `DistributeStakingRewards()`

- `ConsolidateAndBurnDistributions()`
- `CheckTierChange()`
- `InternalTransfer()`
- `AddOrAdjustPackage()`
- `SetVSLContract()`
- `SetVetterContract()`
- `SetTierMultiplier()`
- `SetStakingTransferFee()`
- `SetClaimRewardAmount()`
- `SetUnlockAllFlag()`
- `SetEarlyUnstakeTime()`
- `SetEarlyUnstakePenalty()`
- `SetOldAfterTime()`
- `TransferForeignTokens()`
- `TransferForeignAmount()`
- `TransferInternalAmount()`
- `TransferBNBToAddress()`
- `TransferAllBNBToAddress()`
- `GetNumberOfBurnableDistributions()`
- `GetVSLContract()`
- `GetVetterContract()`

Any compromise to any `_allowedContract` account may allow a hacker to take advantage of this authority and cause severe consequences to normal staking activities.

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., multisignature 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.

Alleviation

[VSL Team]:

It is important to note that any contract with settings that can be adjusted will have the "Centralization Risks" warning. The main risk is loss of access to a wallet that can control the functions listed. This will be mitigated in the future by only providing access to these functions to a DAO or Foundation to make decisions to be executed. At that point, multi-sig and other means can be used to further reduce any risk. In addition, code will show that even if many of these functions are called, there are protections in place. For example, distributing tokens to stakers is a good thing and would not harm anyone if it is called. The fact that royalties have to be converted to VSL token and added to the contract to be distributed makes this functionality necessary as there is no way to seamlessly automate that process. Adjusting to the correct Vetter contract is needed to verify tiers of Vetter token holders, etc. Again, the main concern here is security of the wallets allowed to perform these functions and that is true and will be mitigated.

CKP-02 | Weak PRNG

Category	Severity	Location	Status
Volatile Code	Minor	vsl-token-main/vslstaking.sol: 728~729	Acknowledged

Description

On line 724, the code is generating a hash based on `block.difficulty`, `block.timestamp` and a constant number for randomness. `block.difficulty`, `block.timestamp` can be influenced by miners to some extent, and they should be avoided. Referring to Solidity documentation.

```
724         // Process the payout to the caller (if needed)
725         if(isRandom && processed != 0 && totalTokensPenalized > claimRewardAmount)
726         {
727             uint256 factor = 100;
728             uint256 rand = uint256(keccak256(abi.encodePacked(block.difficulty, block.timestamp, factor)));
729             if((rand % factor) > 30) // 70 % chance of payout...
730             {
731                 // Will need this to check the token balance to pay out...
732                 if(_vslContract == address(0x0)) revert MissingTokenContract();
733                 address _wallet = _msgSender();
734                 IERC20(_vslContract).transfer(_wallet, claimRewardAmount);
735                 totalTokensPenalized -= claimRewardAmount;
736             }
737         }
```

Recommendation

Instead of using `block.timestamp` or `block.difficulty` as a source of randomness, we recommend using a verifiable source of randomness, such as Chainlink VRF(https://docs.chain.link/docs/get-a-random-number/), for the purpose of random number generation.

Alleviation

[VSL Team]:

Weak PRNG is very acceptable in this case as it only controls the possibility of a reward IF the function successfully helps the staker. Worst case, the reward could be guaranteed if the timestamp were influenced, but no harm comes from a 100% chance rather than a 70% chance of reward as stakers are still helped in this case.

CKP-03 | Quiet Exit On Unstake Failure

Category	Severity	Location	Status
Coding Style	Minor	vsl-token-main/vslstaking.sol: 620	Resolved

Description

During the unstaking process, if `penaltyCount > available`, and `_okToTakePenalty` is set to *false*, the `numToDraw` amount will be set to 0. In this case, the function will process to the end quietly, without emitting an event or updating storage. The user won't receive any notification or feedback on the execution result.

Recommendation

In the mentioned scenario, we recommend either reverting the process with proper error message, or emitting an event to notify the user of the result.

Alleviation

Fixed in the commit: b293df4d6afcb8568511227dab7b4b3f79a9e9b6.

CKP-04 | Lack Of Validation On Variable `dbID`

Category	Severity	Location	Status
Coding Style, Volatile Code	Minor	vsl-token-main/vslstaking.sol: 704	Acknowledged

Description

In function `CollectTokens()`, `CheckTierChange()` and `AddOrAdjustPackage()`, a `dbID` is required as input argument. The code does not use it to update storage or check its validity, but only uses it to emit an event. This may pose an issue for function `CollectTokens()`, as the function is callable to all, and the value of `dbID` is not validated before emitting the event.

Recommendation

If this variable is necessary, we recommend adding a validity check (especially in `CollectTokens()`) to ensure the accuracy of the emitted event.

Alleviation

[VSL Team]:

The dbID check is for event listeners to monitor and helps guide them to the correct event, but no harm comes from an invalid ID being passed in for the event so no validity check is needed for processing down the line or internally in the smart contract.

CKP-06 | Recommended Usage Of *Require*

Category	Severity	Location	Status
Coding Style	Informational	vsl-token-main/vslstaking.sol: 469~474	Acknowledged

Description

The code uses *if* and *revert* for validity checks, while Solidity provides *require* function for the same purpose for slightly lower gas fees.

Recommendation

We recommend using *require* statements for better syntax and gas cost.

Alleviation

[VSL Team]:

We respectfully disagree with this assessment and have seen and conducted studies that show it is more efficient to use revert rather than require in terms of gas in many scenarios. This was the reason if...revert was selected as the standard for this project.

CKP-07 | Third Party Dependencies

Category	Severity	Location	Status
Volatile Code	Informational	vsl-token-main/vslstaking.sol: 117~118	Acknowledged

Description

The contract is serving as the underlying entity to interact with Vetter token contract for the value of wallet tier, which is out of scope of this audit. The scope of the audit treats external 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 cause severe impacts. In this specific case, it may affect the value of multiplier in reward calculation.

Recommendation

We understand that the business logic of VSLStaking requires interaction with Vetter token contract. We encourage the team to constantly monitor the statuses of the out-of-scope contract to mitigate the side effects when unexpected activities are observed.

Alleviation

[VSL Team]:

It is necessary for the functionality of staking to know the tier of Vetter held. To mitigate, there is a function to update the Vetter token contract if needed in the future to sustain this link.

CKP-08 | Missing Emit Events

Category	Severity	Location	Status
Coding Style	Informational	vsl-token-main/vslstaking.sol: 1402, 1414, 1437, 1489, 1503, 1515, 1527, 1620, 1638, 1651	Resolved

Description

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

- `SetVSLContract()`
- `SetVetterContract()`
- `SetupAllowedContract()`
- `SetTierMultiplier()`
- `SetStakingTransferFee()`
- `SetClaimRewardAmount()`
- `SetUnlockAllFlag()`
- `SetEarlyUnstakeTime()`
- `SetEarlyUnstakePenalty()`
- `SetOldAfterTime()`

Recommendation

Consider adding events for sensitive actions, and emit them in the function.

Alleviation

Fixed in commit: b293df4d6afcb8568511227dab7b4b3f79a9e9b6

Optimizations

ID	Title	Category	Severity	Status
CKP-05	Redundant Statements	Coding Style	Optimization	Resolved

CKP-05 | Redundant Statements

Category	Severity	Location	Status
Coding Style	Optimization	vsl-token-main/vslstaking.sol: 113	Resolved

Description

```
113     using Address for address;
```

The library `Address` is declared on a using-for directive in `VSLStaking` but its functionalities are never used. It does not affect the functionality of the codebase and appear to be either leftovers from test code or older functionality.

Recommendation

We advise to remove the unused contracts or libraries to better prepare the code for production environments.

Alleviation

Fixed in commit: b293df4d6afcb8568511227dab7b4b3f79a9e9b6

Appendix

Details on Formal Verification

Technical description

All Solidity smart contracts from the project that implement the ERC-20 interface are in scope of the analysis. Each such contract is compiled into a mathematical model which reflects all possible behaviors of the contract. All subsequent verification results are based on that model, which is designed specifically to be amenable to automated analysis by theorem provers and symbolic model checkers. Apart from representing all possible behaviors of the smart contract, the model also incorporates a verification harness that formalizes the initialization and interaction patterns for the contract. In particular, we use a verification harness that non-deterministically selects a public or external function and models its execution. The contract state is initialized non-deterministically (i.e. by arbitrary values) before invocation of the function. Hence, the mathematical model over-approximates the reachable state space of the contract throughout any actual deployment on chain. By doing so, all verification results carry over to the contract's behavior in arbitrary states after it has been deployed. Once the model is constructed, our analysis engine attempts to prove that all executions of the contract are subsumed by a set of pre-defined specifications which capture the desired and admissible behaviors of the smart contract. For the scope of this audit, we use 38 property specifications that cover the functionality of the functions as stated in Sec. Scope.

Assumptions and simplifications

The following assumptions and simplifications have been applied during formal verification:

- Gas consumption is not taken into account, i.e. we assume that executions do not terminate prematurely because they run out of gas.
- The contract's state variables are non-deterministically initialized before invocation of any of those functions. That ignores contract invariants and may lead to false positives. It is, however, a safe over-approximation.
- The verification engine reasons about unbounded integers. Machine arithmetic is modeled as operations on the congruence classes arising from the bit-width of the underlying numeric type. This ensures that over- and underflow characteristics are faithfully represented.

Formalism for property definitions

This section provides details on the 38 formal specifications that were in scope of the audit. All properties are expressed in linear temporal logic (LTL). In that context, we consider all invocations and returns from public and external functions as discrete time steps. Thus, our analysis reasons about the contract's state upon entering and leaving public and external functions.

Apart from the Boolean connectives and the modal operators "always" (written `[]`) and "eventually" (written `<>`), we use the following predicates to reason about the validity of atomic propositions. They are evaluated on the contract's state whenever a discrete time step occurs:

- `started(f, [cond])` Indicates an invocation of contract function `f` within a state satisfying formula `cond`.
- `willSucceed(f, [cond])` Indicates an invocation of contract function `f` within a state satisfying formula `cond` and considers only those executions that do not revert.
- `finished(f, [cond])` Indicates that execution returns from contract function `f` in a state satisfying formula `cond`. Here, formula `cond` may refer to the contract's state variables and to the value they had upon entering the function (using the `old` function).
- `reverted(f, [cond])` Indicates that execution of contract function `f` was interrupted by an exception in a contract state satisfying formula `cond`.

The verification performed in this audit is restricted to pre- and postconditions of procedure invocations. The used model consists of a harness that invokes a non-deterministically selected function of the contract's public and external interface. All formulas are analyzed w.r.t. the trace that corresponds to this function invocation.

Properties for ERC-20 function `transfer(to, amount)`

erc20-transfer-correct-amount

It is expected that non-reverting invocations of `transfer(recipient, amount)` that return `true` subtract the value in `amount` from the balance of the address `msg.sender` and add the same value to the balance entry of the `recipient` address.

```
[](willSucceed(transfer(to, value), to != msg.sender)
   ==> <> (finished(transfer(to, value),
                    return == true
                    ==> balance[msg.sender] == old(balance[msg.sender]) - value
                        && balance[to] == old(balance[to]) + value)))
```

erc20-transfer-correct-amount-self

It is expected that non-reverting invocations of `transfer(recipient, amount)` that return `true` and where the address in `recipient` equals the address of `msg.sender` (i.e. self-transfers) do not change the balance of address `msg.sender`

```
[](willSucceed(transfer(to, value), to == msg.sender)
   ==> <> (finished(transfer(to, value),
                    return == true
                    ==> balance[to] == old(balance[to]))))
```

Properties for ERC-20 function `transferFrom(from, to, amount)`

erc20-transferfrom-revert-from-zero

It is expected that calls of the form `transferFrom(from, dest, amount)` fail if the address value provided in the `from` in-parameter is the zero address.

```
[](started(transferFrom(from, to, value), from == 0)
   ==> <> (reverted(transferFrom) || finished(transferFrom, return == false)))
```

erc20-transferfrom-revert-to-zero

It is expected that calls of the form `transferFrom(from, dest, amount)` fail if the address value provided in the `dest` in-parameter is the zero address.

```
[](started(transferFrom(from, to, value), to == 0)
   ==> <> (reverted(transferFrom) || finished(transferFrom, return == false)))
```

erc20-transferfrom-fail-exceed-balance

Any call of the form `transferFrom(from, dest, amount)` with a value for `amount` that exceeds the balance of address `from` is expected to fail.

```
[](started(transferFrom(from, to, value), value > balance[from])
   ==> <> (reverted(transferFrom) || finished(transferFrom, return == false)))
```

erc20-transferfrom-correct-allowance

It is expected that non-reverting invocations of `transferFrom(from, to, amount)` that return `true` decrease the allowance of the address in `msg.sender` for the address in `from` by the value in `amount`. Two special cases are taken into account:

1. An allowance that equals `type(uint256).max` is treated as an exception and interpreted as an unlimited allowance that does not need to be reduced in order for this check to pass.
2. If the owner of the tokens that are transferred invokes `transferFrom` (i.e. when the address in `msg.sender` equals the address in `from`) we do not require an update of the allowance.

```
[](willSucceed(transferFrom(from, to, value))
   ==> <> finished(transferFrom(from, to, value),
                   return == true
                   ==> ((allowances[from][msg.sender] == old(allowances[from]
[msg.sender]) - value)
                        || (allowances[from][msg.sender] == old(allowances[from]
[msg.sender])
                            && (from == msg.sender
                                || old(allowances[from][msg.sender]) ==
type(uint256).max))))
```

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.

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.

CERTIK