December 2021

# Stader Security Audit Report

Security Assessment

# Summary

This report has been prepared for Stader Labs to discover issues and vulnerabilities in the source code of the staking contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis, Manual Review, and Testnet Deployment 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** | Stader |
| **Description** | Stader offers the most convenient & safest way to maximize your returns on staking. |
| **Platform** | Terra |
| **Language** | Rust |
| **Codebase** | (Private) |
| **Commits** | e2561633859cc846be<br>a024854b61248dd6f28665 |

## Audit Summary

| | |
|---|---|
| **Delivery Date** | Dec 05, 2021 |
| **Audit Methodology** | Static Analysis, Manual Review |
| **Key Components** | Airdrops Registry, Reward Contract, Staking Contract |

## Audit Scope

| Name | File | SHA Checksum |
|---|---|---|
| Airdrops Registry | contracts/airdrops-registry/src/contract.rs | a12f946871267f6d3be2964177ff31a29f0dceb9c5137e284602748cb031caf0 |
| Reward Contract | contracts/reward/src/contract.rs | 4a317d7c767c636d874525acc4b39c7980269bf84b789668ee99120aae90ac3e |
| Staking Contract | contracts/staking/src/contract.rs | a6ddf316bef4f34597073d2ca51f38042e68bf33528b8131bd18d944e9d5ba19 |

# Contents

# 1 Understandings

## 1.1 Overview

The Stader Protocol is a decentralized staking protocol supported on different blockchains. This audit report is specific to Terra's version of its protocol. Stader employs two novel features in it's protocol: plain staking and liquid staking. Plain staking accumulates staking rewards from curated validators pool, while liquid staking enables users to enjoy LP incentives across DEXs and leverage the liquid token while accruing staking rewards.

In addition, it supports community farming for users to earn SD tokens if they stake their LUNA in the first two month since launch.

Users can choose any validator pools, which are a set of multiple validators carefully curated based on their performance to ensure the best returns for users, to stake their token. Once placed into the pool, there are different staking strategies for users to choose. For the auto-compounding strategy, after the community farming stage, the user will be charged 3% of the reward as fee. There will be different fee structures for each strategy.

## 1.2 Plain Staking

Stader stake pools went live on Terra on November 23rd. The staking pool already caters to the staking needs of over 7200 users on Terra and has a TVL of over 4.2 M Luna. Currently, there are 3 validator pools on Stader:

1. Blue Chip with 5 validators
2. Airdrops Plus with 3 validators
3. Community with 5 validators

The selection criteria for the validators has been the following:

1. Uptime criteria $> 99.85\%$ measured between 1st Oct '21 and 1st Nov '21

2. No slashing in the past 3 months
3. 1–2 Pool-specific criteria

There will be a 1% withdrawal fee if users undelegate their token from a pool during the first two month. The unbonding period is 21 days and there will not be any reward during undelegation.

## 1.3 Liquid Staking

Stader supports liquid staking by introducing a token called LunaX. LunaX is a liquid staking token that enables instant unlocking of staked Luna & opens up possibilities across DeFi protocols. Here's a look at the key mechanisms across the life-cycle of LunaX.

1. **Minting**: Users can mint LunaX by staking with the Stader liquidity pool. The deposit function is invoked when users deposit tokens into a pool and the LunaX is minted by Lunax `cw20_contract`

   (address: `terra1xacqx447msqp46qmv8k2sq6v5jh9fdj37az898`).
2. **Value Accrual & Slashing**:
   a) **Accrual**: The rewards generated on the staked Luna (including stablecoins) would be restaked at regular intervals (24 hr to begin with).
   b) **Slashing**: In the event of slashing, LunaX supply remains the same but the price goes down as the quantity of Luna staked will decrease.
3. **Airdrops to LunaX**: Weekly snapshots will be taken at a random time & all holders of LunaX would be allocated protocol airdrops like ANC, MIR, VKR, MINE etc.
4. **Burning**: Users can burn LunaX and unstake Luna at the current exchange rate for a small withdrawal fee.

## 1.4 Privileged Functions

The contract contains the following privileged functions that can only be called if the caller is config.manager, which is determined on the instantiation of the contract. They are used to modify the contract configurations and address attributes. We grouped these functions below:

Updating configuration parameters when calling update_config():

- Parameters that can be updated:
  - `cw20_token_contract`
  - `airdrop_registry_contract`
  - `min_deposit`
  - `max_deposit`
  - `active`
  - `protocol_deposit_fee`
  - `protocol_withdraw_fee`
  - `protocol_reward_fee`
  - `undelegation_cooldown`
  - `unbonding_period`
  - `swap_cooldown`
  - `reinvest_cooldown`

Curating the validators pool by adding or removing validator:

- `add_validator()`
- `remove_validator_from_pool()`

Administrative operation of maintaining the staking pool:

- `rebalance_pool()`
- `swap_rewards()`
- `reinvest()`

## 1.5 Risks of Exchange Rate Manipulation

We analyzed multiple risk factors concerning exchange rate manipulation. In conclusion, there isn't major risks regarding the mechanism of liquidity tokens. For further references, please read Appendix A.

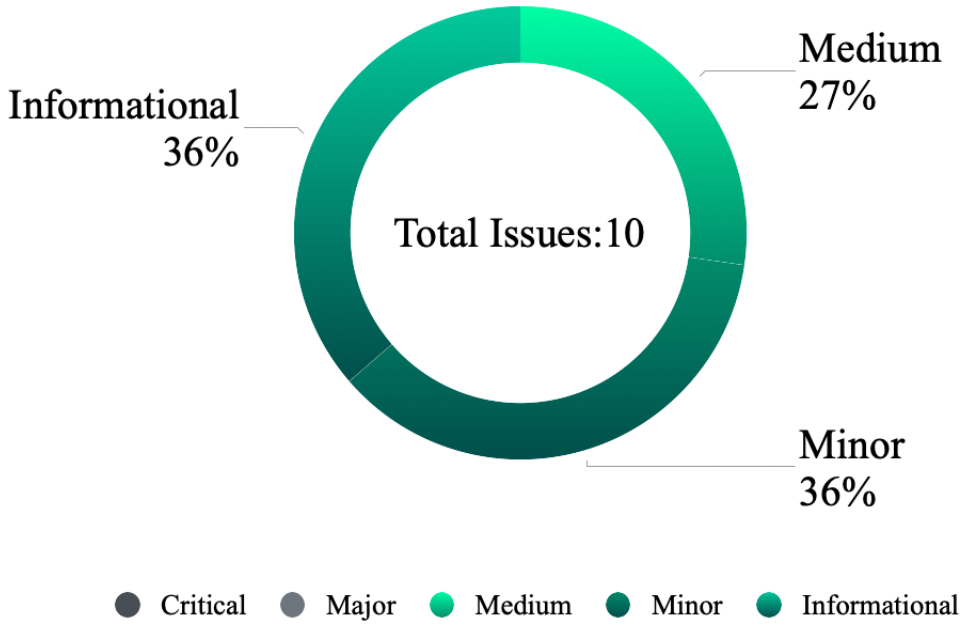
## 1.6 Consistency with Specifications

The team has successfully implemented functions according to the specifications. We list the conclusion below:

- `Instant Liquidity`: This is implemented by introducing the liquid staking token LunaX. Users will receive the minted token once they deposit into the Staking contract.
- `Increased Profits`: The Stader Team maintains a set of scripts that will periodically redeem the contract rewards and reinvest into the staking pool for auto-compounding purposes.

For detailed reference, please read Appendix B.

# 2 Findings




| ID | Title | Category | Severity | Status |
|---|---|---|---|---|
| SSL-01 | Unclear error message | Coding Style | Informational | Acknowledged |
| SSL-02 | Redundant code | Logical Issue | Informational | Acknowledged |
| SSL-03 | Unoptimized loop break | Gas Optimization | Medium | Acknowledged |
| SSL-04 | Missing event emitting | Coding Style | Informational | Acknowledged |
| SSL-05 | Missing error type | Logical Issue | Minor | Acknowledged |
| SSL-06 | Improper variable name | Coding Style | Informational | Acknowledged |
| SSL-07 | Unoptimized code block | Gas Optimization | Medium | Acknowledged |
| SSL-08 | Privileged ownership | Centralization / Privilege | Minor | Acknowledged |
| SSL-09 | Possibility of for loop exceeds block limit | Volatile Code | Medium | Acknowledged |
| SSL-10 | Unused imports | Coding Style | Minor | Acknowledged |

## 2.1 SSL-01 | Unclear error message

| Category | Severity | Location | Status |
|---|---|---|---|
| Coding Style | Minor | staking/contract.rs: 1123 | Acknowledged |

### 2.1.1 Description

The error message in the following code snippet is unclear.

```
compute_withdrawable_funds(deps.storage, batch_id, &user_addr)
.expect("compute_withdrawable_funds failed")}
```

### 2.1.2 Recommendation

The original error message was “compute_withdrawable_funds failed". We recommend it to be “Failure while computing withdrawable funds”.

### 2.1.3 Alleviation

The adjustment is beneficial but not crucial. The team acknowledged the finding. Given that this will not affect the well-being of the users, the team decided to retain the code base unchanged and keep the recommendations for future updates.

## 2.2 SSL-02 | Redundant code

| Category | Severity | Location | Status |
|---|---|---|---|
| Logical Issue | Informational | staking/contract.rs: 711 | Acknowledged |

### 2.2.1 Description

Contract state update is unnecessary since the state won't be changed in this case.

### 2.2.2 Recommendation

The following code can be removed:

```
STATE.save(deps.storage, &state)?
```

### 2.2.3 Alleviation

The adjustment is beneficial but not crucial. The team acknowledged the finding. Given that this will not affect the well-being of the users, the team decided to retain the code base unchanged and keep the recommendations for future updates.

## 2.3 SSL-03 | Unoptimized loop

| Category | Severity | Location | Status |
|---|---|---|---|
| Gas Optimization | Medium | staking/contract.rs: 772 | Acknowledged |

### 2.3.1 Description

In the following code snippet, the loop-breaking condition check should be executed before any other operation. Note that this does not affect users' funds in any way but only the gas cost of the contract operator.

```
for index in (0..stake_tuples.len()).rev() {
    let tuple_val = stake_tuples.get(index).unwrap().clone();
    if to_undelegate.is_zero() {
        break;
    }
}
```

### 2.3.2 Recommendation

We advise changing the code to:

```
for index in (0..stake_tuples.len()).rev() {
    if to_undelegate.is_zero() {
        break;
    }
    let tuple_val = stake_tuples.get(index).unwrap().clone();
    }
}
```

### 2.3.3 Alleviation

The adjustment is beneficial but not crucial. The team acknowledged the finding. Given that this will not affect the well-being of the users, the team decided to retain the code base unchanged and keep the recommendations for future updates.

## 2.4 SSL-04 | Missing event emitting

| Category | Severity | Location | Status |
|---|---|---|---|
| Coding Style | Informational | `*/contract.rs` | Acknowledged |

### 2.4.1 Description

In the contracts, there are several functions that can change the state variables or perform informational operations. However, these functions do not emit events to pass the changes out of the chain.

### 2.4.2 Recommendation

Avoid using `Response::default()`. Each state updated should have an event emitted.

### 2.4.3 Alleviation

The adjustment is beneficial but not crucial. The team acknowledged the finding. Given that this will not affect the well-being of the users, the team decided to retain the code base unchanged and keep the recommendations for future updates.

## 2.5 SSL-05 | Missing error type

| Category | Severity | Location | Status |
|---|---|---|---|
| Logical Issue | Minor | `staking/contract.rs: 345, 459, 487` | Acknowledged |

### 2.5.1 Description

There are missing error types for the following functions:

1. `check_slashing()`
2. `compute_deposit_breakdown()`
3. `redeem_rewards()`

### 2.5.2 Recommendation

We advise adding error types to the above functions. Such as:

1. `ContractError:UpdateSlashingError`
2. `ContractError:ComputeError`
3. `ContractError:RedeemError`

### 2.5.3 Alleviation

The adjustment is beneficial but not crucial. The team acknowledged the finding. Given that this will not affect the well-being of the users, the team decided to retain the code base unchanged and keep the recommendations for future updates.

## 2.6 SSL-06 | Improper variable name

| Category | Severity | Location | Status |
|---|---|---|---|
| Coding Style | Informational | staking/contract.rs:164, 344, 523 | Acknowledged |

### 2.6.1 Description

There are several variables that are not named properly. We describe the findings in the following:

1. The `config.active` is only used in `deposit()`, not in other functions
2. The `check_slashing` function not only checks slashing but may also update `STATE(total_staked, exchange_rate)` and `VALIDATOR_META(staked, slashed)`
3. The `swap_rewards()` function swaps all rewards only into Luna, no other tokens.

### 2.6.2 Recommendation

We suggest to do the following adjustment:

1. Rename `config.active` to `config.is_open_for_deposit`
2. Rename `check_slashing` function to `update_slashing`
3. Rename `swap_rewards()` to `swap_rewards_to_luna()`

### 2.6.3 Alleviation

The adjustment is beneficial but not crucial. The team acknowledged the finding. Given that this will not affect the well-being of the users, the team decided to retain the code base unchanged and keep the recommendations for future updates.

## 2.7 SSL-07 | Unoptimized coding block

| Category | Severity | Location | Status |
|---|---|---|---|
| Gas Optimization | Medium | staking/contract.rs: 733 | Acknowledged |

### 2.7.1 Description

The contract uses `get_active_validators_sorted_by_stake()` to get a validators list sorted by the staking amount. Then, it reverses the loop and iterates it to undelegate from the largest staked validators until the fulfillment of the intended undelegation amount.

```
let stake_tuples = get_active_validators_sorted_by_stake(
        deps.querier,
        env.contract.address.clone(),
        validators,
    )?;

    for index in (0..stake_tuples.len()).rev() {
        ...
    }
```

### 2.7.2 Recommendation

Instead of sorting the validator array in ascending order and calling .rev() afterward, we advise the following changes for run-time optimization:

```
use std::cmp::Reverse;
vec.sort_by_key(|w| Reverse(*w));
```

### 2.7.3 Alleviation

The adjustment is beneficial but not crucial. The team acknowledged the finding. Given that this will not affect the well-being of the users, the team decided to retain the code base unchanged and keep the recommendations for future updates.

## 2.8 SSL-08 | Privileged ownership

| Category | Severity | Location | Status |
|---|---|---|---|
| Centralization / Privilege | Minor | `staking/contract.rs` | Acknowledged |

### 2.8.1 Description

The manager of the Staking contract has the permission to:

1. update contract configurations
2. add new validators
3. remove existing validators
4. rebalance delegation between validators

without obtaining the consensus of the community.

In the worst-case scenario, the manager can set the `protocol_withdraw_fee` to 100% and users will lose all funds when calling `withdraw_funds_to_wallet()`.

### 2.8.2 Recommendation

Renounce ownership when it is the right timing, or gradually migrate to a timelock plus multisig governing procedure and let the community monitor in respect of transparency considerations. It is also advised to set a maximum possible value of the `protocol_withdraw_fee` to limit the power of the manager.

### 2.8.3 Alleviation

[Stader Team]:

- Validators are handpicked by Stader based on the data of past performance. This isn't data available on Terra blockchain currently but rather through an analysis on the off-chain side. Once a set of validators are selected, Stader will monitor, readjust validators once a month.
- Our UI will show how much the withdrawal fee is directly reading from the contract. Thus this is not going to be an issue. Further, we let this flexibility

remain so as to deploy some of these contracts for specific firms so they can manage it themselves. This gives them greater flexibility.

## 2.9 SSL-09 | Possibility of for loop exceeds block limit

| Category | Severity | Location | Status |
|---|---|---|---|
| Volatile Code | Medium | `staking/contract.rs: 350, 497, 772` | Acknowledged |

### 2.9.1 Description

There are loops that iterate through the validators list, which might potentially result in exceeding the block limit.

```
for val_addr in state.validators.iter(){
};
for val_addr in state.validators{
};
for index in (0..stake_tuples.len()).rev(){
};
```

### 2.9.2 Recommendation

We recommend adding block limit variables for each for loop as condition checks.

### 2.9.3 Alleviation

[Stader Team]: We will likely not address this as we expect only 10 validators in our business logic in the staking contract. Test runs have succeeded without timing out.

## 2.10 SSL-10 | Unused imports

| Category | Severity | Location | Status |
|---|---|---|---|
| Coding Style | Minor | `staking/contract.rs` | Acknowledged |

### 2.10.1 Description

There are unused imports in Stader's contracts, which could be removed.

### 2.10.2 Recommendation

We recommend using the "organized import" shortcut in vscode to remove all the unused imports.

### 2.10.3 Alleviation

The adjustment is beneficial but not crucial. The team acknowledged the finding. Given that this will not affect the well-being of the users, the team decided to retain the code base unchanged and keep the recommendations for future updates.

# Appendices

# Appendix A

## Exchange Rate Manipulation Analysis

### Calculation Formula

```
exchange rate = total_staked / total_token_supply
```

### Expectations

All of the following expectations have been tested to ensure the correctness.

1. Executing `deposit()` should not change the exchange rate
2. Executing `undelegate_stake()`should not change the exchange rate
3. Executing `reinvest()` should increase the exchange rate, since no new tokens will be minted
4. If any of the validators has been slashed, the exchange rate should be decreased
5. If new yield are generated from staking, the exchange rate should be increased

### Variables Analysis

- `total_staked`
  - Functions that can update this variable:
    - Increase
      - `deposit()`: Update the value to increase by the current user's deposit subtracting the protocol fee

        - `reinvest()`: Update the value to increase by the reward contract's balance subtracting protocol fee
        - `reimburse_slashing()`: Update the value to increase by the reimburse amount
        - `check_slashing()`: Update the value to increase by the new staking yield amount if there is any
    * Decrease
        - `undelegate_stake()`: Update the value to decrease by the undelegation amount
        - `check_slashing()`: Update the value to decrease by the slashed amount if any of the validators has been slashed

- `total_token_supply`
  - Functions that can update this variable:
    * Increase
        - `deposit()`: Update the value to increase by the new tokens to mint with `create_mint_message()`
            1. According to the provided contract code, the token minter is set to the Staking contract and cannot be changed. There is no other way to mint new tokens.
            2. The token contract is based on cw20, a token standard similar to ERC20, referencing this implementation where we examined the token mint function: `https://github.com/CosmWasm/cw-plus/blob/6287de98c07f4d9be8ed610552ccbd9987028491/contracts/cw20-base/src/contract.rs#L278`
            3. We verified that the minter address of the token contract shown in the Terra Explorer is the same as the Staking contract address: `terra1xacqx447msqp46qmv8k2sq6v5jh9fdj37az898`: `https://finder.terra.money/mainnet/address/terra17y9qkl8dfkeg4py7n0g5407emqnemc3yqk5rup`.
    * Decrease
        - `undelegate_stake()`: Update the value to decrease by the undelegated token amount with `burn_minted_tokens()`
            1. Only the tokens minted by the Staking contract can be

burned. There is no other way to burn tokens.

## Conclusion

The exchange rate may be changed under the following four scenarios. We conclude that there aren't major risk concerns:

1. Reinvest: Contract rewards will be reinvested periodically. It is less likely that the rewards will surge in a short period of time.
2. Reimbursed slashing: Anyone can stake into the Staking contract without minting new tokens which increases the exchange rate. The higher the exchange rate, the higher the `user_withdrawal_amount`. However, an adversary will not have incentive to pump the exchange rate like this because he will not gain any LunaX token for his new stake. Furthermore, this exchange rate pump will benefit all users with withdrawable funds. Other users can simply withdraw their funds once they notice a higher exchange rate. It is considered not profitable if there's more than one user staking and the relevant protocol fee is greater than 0
3. Validators slashed: the team clarified that the maximum amount a validator can be slashed is around 5%. So far only once or twice has the 5% slashing happened in Terra. Most slashing events are 0.01% of the total stake. Given that Stader uses performance data to handpick validators, there's a good chance Stader will seldom see slashing happening across any of its validators.
4. New staking yield generated: the staking yield generation should be stable and slowly distributed to the eligible validators. It is unlikely to see a spike in the yield amount if a user maliciously deposits and undelegates in a short period of time.

[1]

[1]Note: `deposit()` and `undelegate_stake()` won't change the exchange rate. Hence there is no possibility to manipulate the exchange rate externally through common techniques such as the flash loan attack.

# Appendix B

## Consistency with Specifications

### Instant Liquidity

By introducing LunaX, which is minted to users upon deposit into the Staking contract, there will be another pool for users to swap LunaX with Luna as in common decentralized exchanges. Compared to undelegating directly through Stader's Staking contract, users can enjoy instant liquidity instead of waiting for 21 days to undelegate their stake.

### Increased Profit

To maximize users' profits, the Stader Team maintains a set of scripts to call `redeem_reward()`, `reinvest()`, `swap_rewards()` every N hours. The two functions are also available for the community to execute themselves in their best interest. However, it should be noted that the auto-compounding will stop once the above-mentioned functions are not executed.

For real time execution details, please refer to the Terra Explorer of the deployed Staking contract at `https://terra.stake.id/?#/address/terra1xacqx447msqp46qmv8k2sq6v5jh9fdj37az898`.

# Appendix C

### Finding Categories

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

**Mathematical Operations** Mathematical Operation findings relate to mishandling of math formulas, such as overflows, incorrect operations etc. Logical Issue Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

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

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

**Data Flow** Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a struct assignment operation affecting an in-memory struct rather than an in-storage one.

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

**Centralization / Privilege** Centralization / Privilege findings refer to the logic or implementation of the code exposing to concerns or scenarios that would go against decentralized manners.

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

**Inconsistency** Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code,such as a constructor assignment imposing different require statements on the input variables than a setter function.

**Magic Numbers** Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as constant contract variables aiding in their legibility and maintainability.

**Compiler Error** Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

# 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 is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. 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 intended 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. The Z Institute's position is that each company and individual are responsible for their own due diligence and continuous security. The Z Institute'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 The Z Institute are 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 lay-

ers 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 "AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, THE Z INSTITUTE HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, THE Z INSTITUTE SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANT ABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM THE COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, THE Z INSTITUTE 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 THE 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, THE Z INSTITUTE 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 THE Z INSTITUTE NOR ANY OF THE Z INSTITUTE'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. THE Z INSTITUTE 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 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 THE Z INSTITUTE'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 THE Z INSTITUTE WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.THE REPRESENTATIONS AND WARRANTIES OF THE Z INSTITUTE 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 THE Z INSTITUTE 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 REPORT FOR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

# About

The Z Institute is an online blockchain talent incubator and accelerator with a focus on bringing more developers to the space and a blockchain transformer that empowers businesses in technology and research. Since 2017, The Z Institute's founding team has been providing consulting services in customized smart contracts, DApp development, and security audit.

*consulting@zinstitute.net*


Z INSTITUTE