Security Assessment

Z1 Financial

Dec 4th, 2021

Table of Contents

Summary

This report has been prepared for Z1 Financial to discover issues and vulnerabilities in the source code of the Z1 Financial 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	Z1 Financial
Platform	BSC
Language	Solidity
Codebase	https://bscscan.com/address/0x4053080538b038e65700fa569041cacbf16561e6 https://github.com/Sixsixone5/ztu/
Commit	28607bddaf93bde9a671b246d561734ef8c63080 a0b811aa35bda1c2aea50780c6fe51cb33de662a

Audit Summary

Delivery Date	Dec 04, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	

Vulnerability Summary

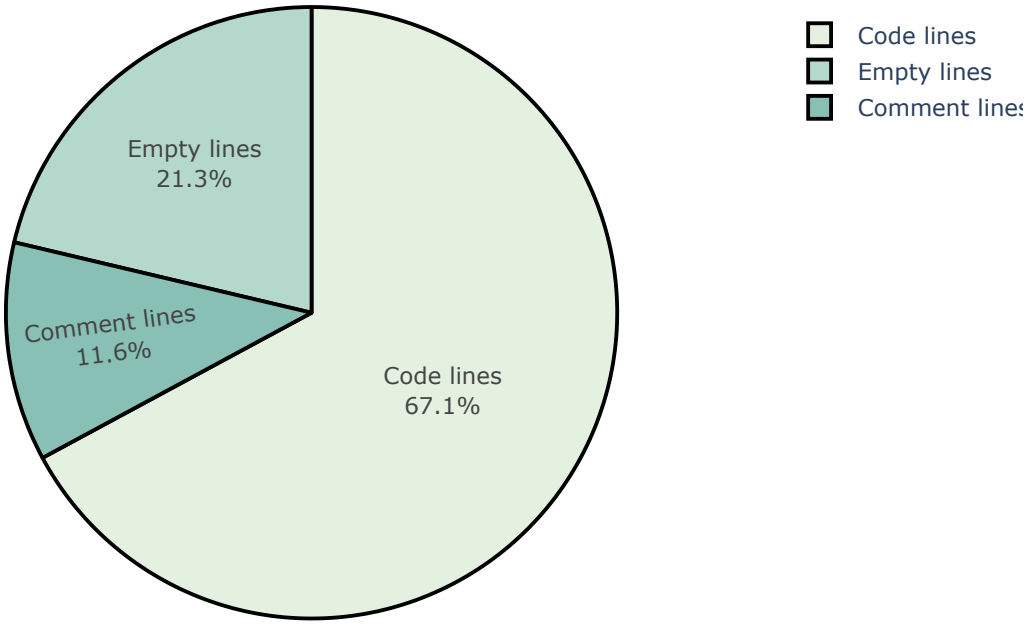
Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	0	0	0	0	0	0
Major	1	0	0	0	0	1
Medium	0	0	0	0	0	0
Minor	6	0	0	2	0	4
Informational	2	0	0	0	0	2
Discussion	0	0	0	0	0	0

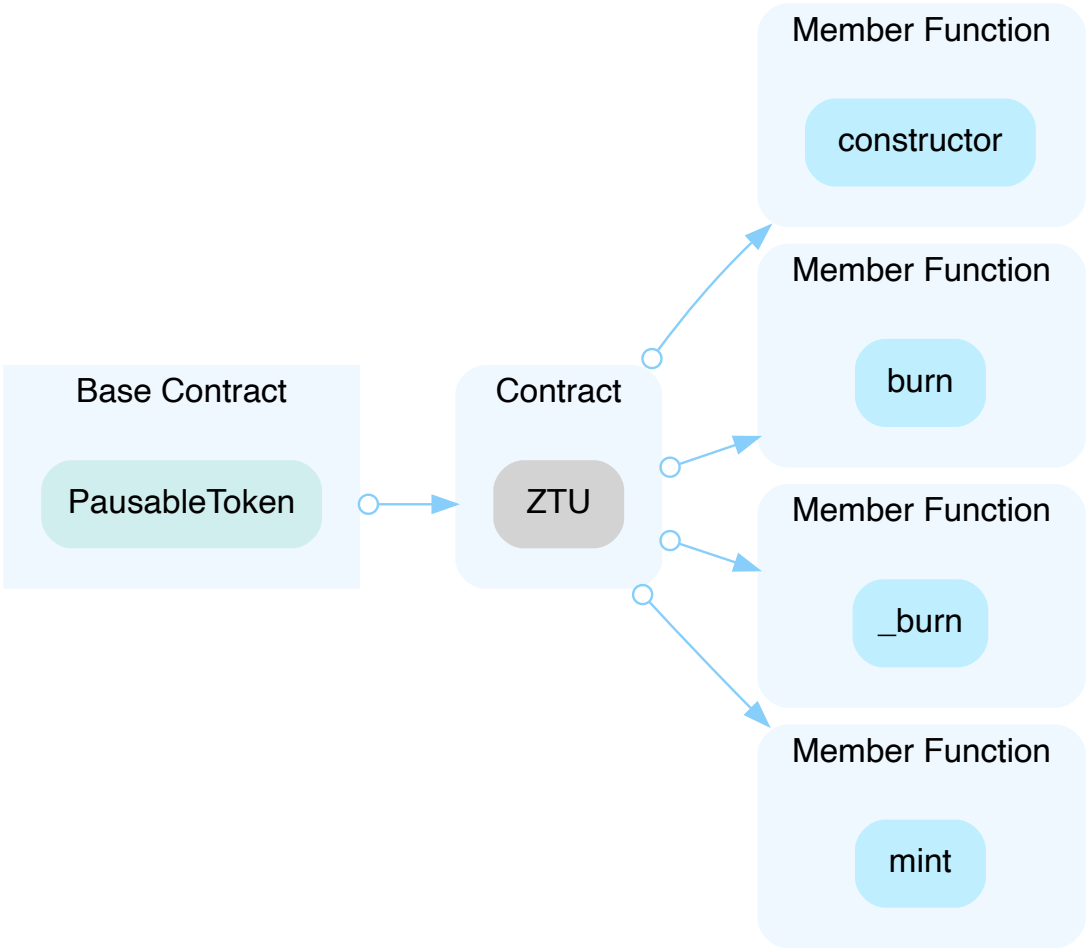
Audit Scope

ID	File	SHA256 Checksum
ZTU	ZTU.sol	9130edc19e5e2acd2b77572d4f02360f39334d649a9cbf954956dd5103d71e61

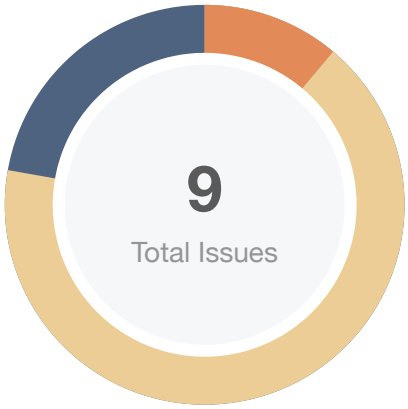
Diagrams

Source Line Chart




Base Contract
PausableToken
Contract
ZTU
Member Function
constructor
Member Function
burn
Member Function
_burn
Member Function
mint

Findings



Critical	**0** (0.00%)
Major	**1** (11.11%)
Medium	**0** (0.00%)
Minor	**6** (66.67%)
Informational	**2** (22.22%)
Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
ZTU-01	Insecure Compiler Version	Language Specific	Informational	Resolved
ZTU-02	Missing Zero Address Validation and Error Message	Control Flow	Minor	Resolved
ZTU-03	Improper Usage of `public` and `external` Type	Gas Optimization	Informational	Resolved
ZTU-04	Centralization Risk	**Centralization / Privilege**	**Major**	Resolved
ZTU-05	Pull-Over-Push Pattern	Logical Issue	Minor	Resolved
ZTU-06	Missing Zero Address Validation and Error Message	Control Flow	Minor	Resolved
ZTU-07	Missing Zero Address Validation and Error Message	Control Flow	Minor	Resolved
ZTU-08	Potential Risks on `approve` and `transferFrom` Methods	Logical Issue	Minor	Acknowledged
ZTU-09	Potential Risk On `increaseApproval` and `decreaseApproval` Methods	Logical Issue	Minor	Acknowledged

ZTU-01 | Insecure Compiler Version

Category	Severity	Location	Status
Language Specific	Informational	projects/Z1Financial/ZTU.sol (31c9b08): 5	Resolved

Description

Using a floating pragma can be problematic. The pragma included in the contract uses the "^" prefix specifier, denoting that a compiler version, greater than the version used in the contract, can be used for compilitation.

Recommendation

In order to avoid compiler-specific bugs, it is a general practice to lock the compiler at a specific version. We recommend locking the compiler at the lowest possible version that supports all the capabilities wished by the codebase. This will ensure that the project uses a compiler version that has been tested and in use for the longest time, and as such is less likely to contain yet-undiscovered bugs.

Alleviation

`[Z1 Financial Team]`: The client heeded the advice and fixed the finding in the commit 28607bddaf93bde9a671b246d561734ef8c63080

ZTU-02 | Missing Zero Address Validation and Error Message

Category	Severity	Location	Status
Control Flow	Minor	projects/Z1Financial/ZTU.sol (31c9b08): 189	Resolved

Description

The constructor uses the `tokenOwner` address without checking the address validity.

Recommendation

We advise the client to add a require statement for the zero address validation and to add an error message in the require statement as below

```
require(tokenOwner != address(0), "Zero address validation failed in constructor")
```

Alleviation

`[Z1 Financial Team]`: The client heeded the advice and fixed the finding in the commit 28607bddaf93bde9a671b246d561734ef8c63080

ZTU-03 | Improper Usage of `public` and `external` Type

Category	Severity	Location	Status
Gas Optimization	Informational	projects/Z1Financial/ZTU.sol (31c9b08): 45~49, 73~76, 80~83, 177~179, 199~201, 209~214	Resolved

Description

The functions have public visibility even though are never called from the contract.

Recommendation

Use the external attribute for functions never called from the contract.

Public and external functions differs in terms of gas usage. Solidity copies arguments to memory on a public function, while external read from calldata, which is cheaper than memory allocation. Public functions are called internally and externally. Internal calls are executed via jumps in the code because array arguments are passed internally by pointers to memory. When the compiler generates the code for an internal function, that function expects its arguments to be located in memory. That is why public functions are allocated to memory. External functions are never exectued internally, so there is no need to store arguments in memory for the internal calls. As a consequence, external functions require less gas compared to public functions.

Alleviation

`[Z1 Financial Team]`: The client heeded the advice and fixed the finding in the commit 28607bddaf93bde9a671b246d561734ef8c63080

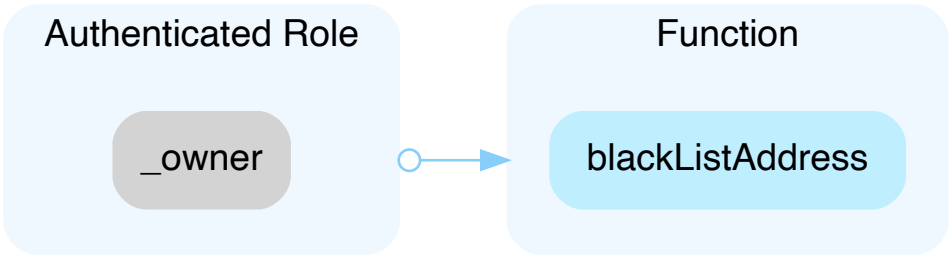
ZTU-04 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	Major	projects/Z1Financial/ZTU.sol (31c9b08): 177~179, 209~214	Resolved

Description

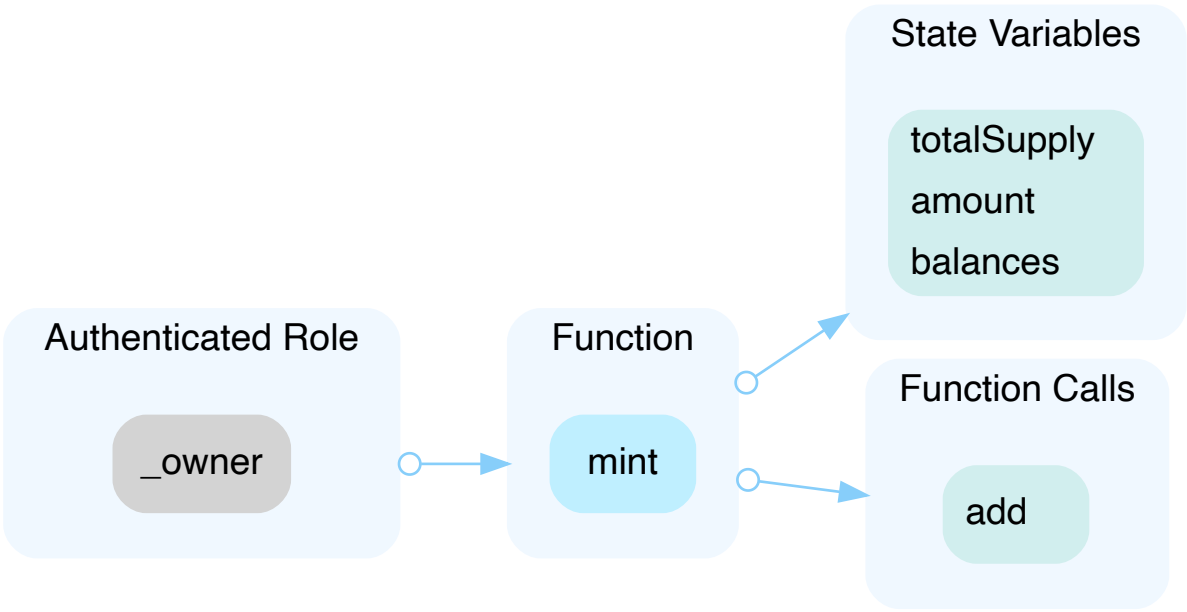
In the contract, `PausableToken`, the role `_owner`, has the authority over the functions shown in the diagram below. This ownership management introduces a single point of failure.

Any compromise to the privileged account which has access to `_owner` may allow the hacker to take advantage of this.



In the contract, `ZTU`, the role `_owner`, has the authority over the functions shown in the diagram below. This ownership management introduces a single point of failure.

Any compromise to the privileged account which has access to `_owner` may allow the hacker to take advantage of this.



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 is some feasible suggestions that would also mitigate the potential risk at the different level in term 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

`[Z1 Financial Team]`: The client implemented the multiple signer control in the ZTU.sol in the commit 28607bddaf93bde9a671b246d561734ef8c63080. The replacement of these signers are controlled by sensitive role `owner`.

`[Z1 Financial Team]`: `_blackList()` and `blackListAddress()` functions have been removed from the ZTU.sol in the commit a0b811aa35bda1c2aea50780c6fe51cb33de662a

`[Z1 Financial Team]`: The client implemented the multiple signer control in the ZTU.sol via the gnosis safe as recommended. The gnosis safe contract wallet has been deployed with address

0x205f4b1d9701b1c983E5A586bE44aEA092320292 and transaction hash 0xf5005ff527831d9b6b493cb192df9c4b02e9bd7b449707a5d260637183549b86.

The new ZTU.sol contract has been deployed with address 0xb4D4762Ed3591Ec6b672BfB6019BF98c4dda5ea4 and transaction hash 0x086e8c1f697cdc01f177c6937915c60629ed48d5a7e57691b0b2d3910cf563ec.

The owner of the new ZTU.sol contract is the address of the gnosis safe contract wallet 0x205f4b1d9701b1c983E5A586bE44aEA092320292 which has been configured with a 3 out of 4 signature threshold. With this development the ZTU.sol has now multiple owners:

- Sokheng Say : 0x2a0ed776098010CBaC94048C229a430620B3094D
- Sy Pauv : 0xEF08EbE79763268A85d2Cc4E1faF035f3DAC999B
- Bogdan Vorozhtsov : 0xe6D193295e1E967B42f044E688952B7361aD8F45
- Borapyn Py : 0x48173D77Bd40Ea81B9A56f0812A8636C7bEdAEAB

The detailed gnosis safe deployment can be found at https://z1f.io/2021/12/04/update-the-final-preparations-and-finalizing-ztu-smart-contract/

ZTU-05 | Pull-Over-Push Pattern

Category	Severity	Location	Status
Logical Issue	Minor	projects/Z1Financial/ZTU.sol (31c9b08): 47~51	Resolved

Description

The change of `owner` by function `transferOwnership()` overrides the previously set `owner` with the new one without guaranteeing the new `owner` is able to actuate transactions on-chain.

Recommendation

We advise the pull-over-push pattern to be applied here whereby a new `owner` is first proposed and consequently needs to accept the `owner` status ensuring that the account can actuate transactions on-chain.

The following code snippet can be taken as a reference:

```
address public potentialAdmin;

function transferAdmin(address pendingAdmin) external onlyAdmin {
    require(pendingAdmin != address(0), "potential admin can not be the zero address.")
    potentialAdmin = pendingAdmin;
    emit AdminNominated(pendingAdmin);
}

function acceptAdmin() external {
    require(msg.sender == potentialAdmin, 'You must be nominated as potential admin
before you can accept administer role');
    admin = potentialAdmin;
    potentialAdmin = address(0);
    emit AdminChanged(admin)
}
```

Alleviation

`[Z1 Financial Team]`: The client heeded the advice and fixed the finding in the commit 28607bddaf93bde9a671b246d561734ef8c63080

ZTU-06 | Missing Zero Address Validation and Error Message

Category	Severity	Location	Status
Control Flow	Minor	projects/Z1Financial/ZTU.sol (31c9b08): 170~172	Resolved

Description

Missing Zero Address Validation and Error Message in require statement.

Recommendation

We advise the client to add a require statement for the zero address validation and to add and error message in the require statement as below:

```
require(spender != address(0), "Zero address validation failed in approve function");
```

Alleviation

`[Z1 Financial Team]`: The client heeded the advice and fixed the finding in the commit 28607bddaf93bde9a671b246d561734ef8c63080

ZTU-07 | Missing Zero Address Validation and Error Message

Category	Severity	Location	Status
Control Flow	Minor	projects/Z1Financial/ZTU.sol (31c9b08): 202~208	Resolved

Description

Missing Zero Address Validation and Error Message in require statement.

Recommendation

We advise the client to add a `require` statement for the zero address validation and to add an error message as below:

```
202 function _burn(address _who, uint256 _value) internal {
203     require(_who != address(0) ,"Zero address validation failed in function burn");
204     require(_value <= balances[_who], "Insufficient Balance");
205     balances[_who] = balances[_who].sub(_value);
206     totalSupply = totalSupply.sub(_value);
207     emit Burn(_who, _value);
208     emit Transfer(_who, address(0), _value);
209 }
```

Alleviation

`[Z1 Financial Team]`: The client heeded the advice and fixed the finding in the commit 28607bddaf93bde9a671b246d561734ef8c63080

ZTU-08 | Potential Risks on `approve` and `transferFrom` Methods

Category	Severity	Location	Status
Logical Issue	Minor	projects/Z1Financial/ZTU.sol (31c9b08): 129~133, 117~128	Acknowledged

Description

The `approve` function could be used in an attack that allows a spender to transfer more tokens than the owner of the tokens ever wanted to allow the spender to transfer.

Here is a possible attack scenario:

Alice allows Bob to transfer N of Alice's tokens (N>0) by calling approve method on Token smart contract passing Bob's address and N as method arguments. After some time, Alice decides to change from N to M (M>0) the number of Alice's tokens Bob is allowed to transfer, so she calls approve method again, this time passing Bob's address and M as method arguments. Bob notices Alice's second transaction before it was mined and quickly sends another transaction that calls `transferFrom` method to transfer N Alice's tokens somewhere. If Bob's transaction will be executed before Alice's transaction, then Bob will successfully transfer N Alice's tokens and will gain the ability to transfer another M tokens before Alice noticed that something went wrong. Thus Bob can call `transferFrom` method again, this time to transfer M Alice's tokens.

So, Alice's attempt to change Bob's allowance from N to M (N>0 and M>0) made it possible for Bob to transfer N+M of Alice's tokens, while Alice never wanted to allow so many of her tokens to be transferred by Bob.

Recommendation

We recommend using `safeApprove` and `safeTransfer` from SafeERC20 instead of ERC20 `approve` and `transferFrom` methods.

ZTU-09 | Potential Risk On `increaseApproval` and `decreaseApproval` Methods

Category	Severity	Location	Status
Logical Issue	Minor	projects/Z1Financial/ZTU.sol (31c9b08): 137~141, 142~151	Acknowledged

Description

The methods `increaseApproval` and `decreaseApproval` are unsafe.

Here is the scenario:

Bob is allowed to transfer zero Alice's tokens. Alice allows Bob to transfer 100 of here tokens via approve or `increaseApproval` method and transaction is executed successfully. Alice sees that Bob is now allowed to transfer 100 of her tokens. After some time, Alice uses `decreaseApproval` method to decrease by 100 the number of her tokens Bob is allowed to transfer, and transaction is executed successfully and proper Approval event was logged. Alice sees that Bob is allowed to transfer 0 of her tokens. Now Alice may think that once `decreaseApproval` transaction was executed successfully, then Bob didn't manage to transfer any of her tokens before the allowance was decreased, but this assumption is wrong. Actually, Bob may or may not have transferred Alice's tokens before allowance was decreased, and Alice has no easy way to know for sure whether Bob transferred her tokens or not

Recommendation

We recommend to refer to the `SafeERC20` contract and to add `safeIncreaseAllowance` and `safeDecreaseAllowance` functions instead.

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.

Language Specific

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

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.