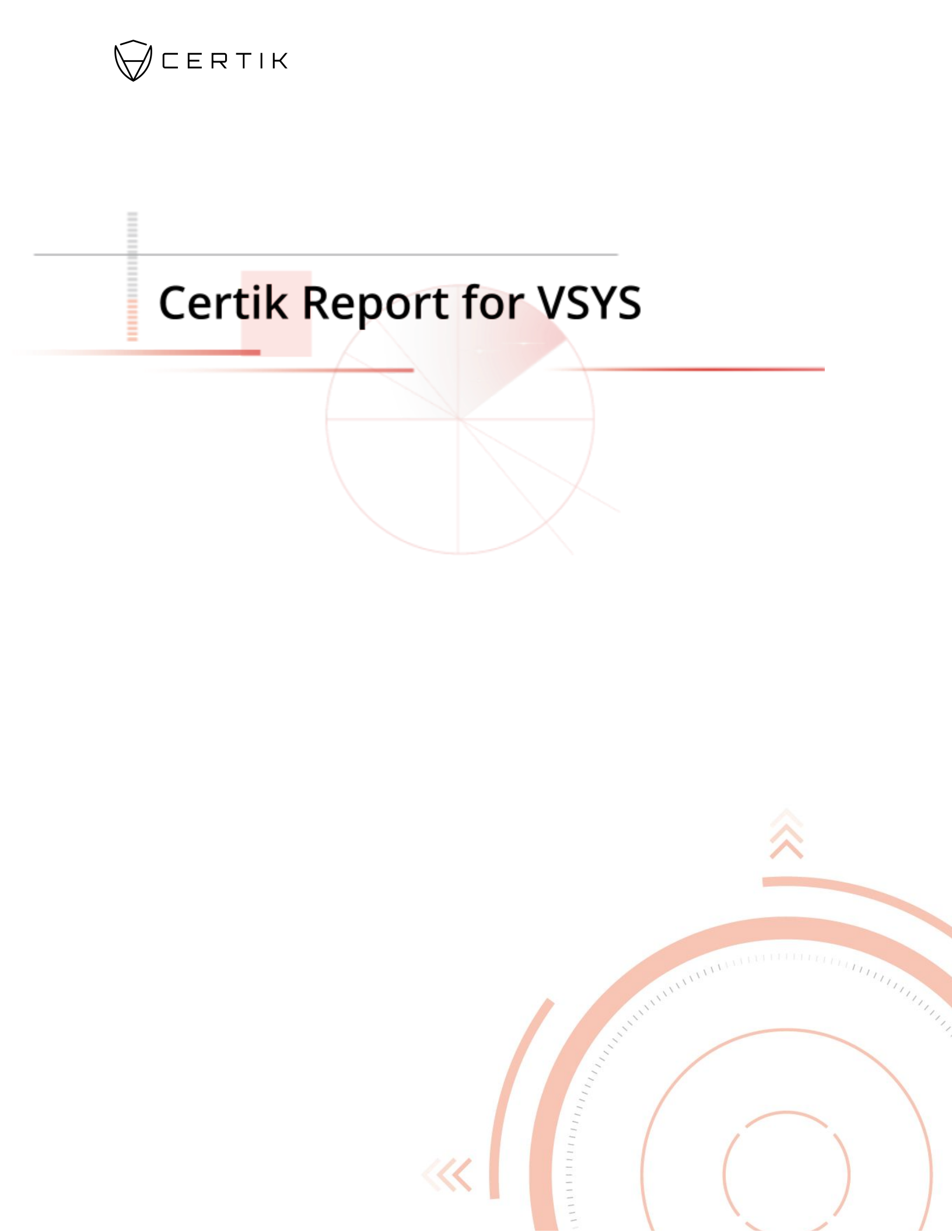




Certik Report for VSYS

Contents

Contents	1
Disclaimer	2
About CertiK	3
Executive Summary	3
Testing Summary	5
SECURITY LEVEL	5
SOURCE CODE	5
PLATFORM	5
VULNERABILITY OVERVIEW	5
LANGUAGE	5
REQUEST DATE	5
REVISION DATE	5
METHODS	5
Source Of Truth	6
From vsys smart contract document:	6
Scope Of Audit	6
Source Code SHA-256 Checksum	7
Contract	7
OpcDiff	7
Review Comments	9
blockchain/state/opcdiffs/	9
Best Practice	11
General	11
Logging	11
Arithmetic Vulnerability	11
Two's Complement / Integer underflow / overflow	11
Floating Points and Precision	11
Access & Privilege Control Vulnerability	11
Circuit Breaker	11
Restriction	12
DoS Vulnerability	12
Unexpected Revert	12
Human Factor Manipulation Vulnerability	12

Avoid state changes before validation checks	12
Visibility Vulnerability	13
Incorrect Interface Vulnerability	13
Documentation	13
Testing	14

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 Verification Services Agreement between CertiK and VSYS (the “Company”), or the scope of services/verification, and terms and conditions provided to the Company in connection with the verification (collectively, 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 without CertiK’s prior written consent.

About CertiK

CertiK is a technology-led blockchain security company founded by Computer Science professors from Yale University and Columbia University built to prove the security and correctness of smart contracts and blockchain protocols.

CertiK, in partnership with grants from IBM and the Ethereum Foundation, has developed a proprietary Formal Verification technology to apply rigorous and complete mathematical reasoning against code. This process ensures algorithms, protocols, and business functionalities are secured and working as intended across all platforms.

CertiK differs from traditional testing approaches by employing Formal Verification to mathematically prove blockchain ecosystem and smart contracts are hacker-resistant and bug-free. CertiK uses this industry-leading technology together with standardized test suites, static analysis, and expert manual review to create a full-stack solution for our partners across the blockchain world to secure 6.2B in assets. For more information: <https://certik.org>.

Executive Summary

V SYSTEMS, a distributed database project using cutting edge blockchain technology that allows all economic systems to build their app on top of the platform.

CertiK is invited by VSYSTEMS team for reviewing the Non-Turing-Complete, smart contract technology development. The development is planned into three phrase:

1. Token creation, distribution, and issuance
2. Token trading and management
3. Optimize the performance

Testing Summary

SECURITY LEVEL



VULNERABILITY OVERVIEW



This report has been prepared as a product of the Audit request by VSYS.

TYPE	Chain
SOURCE CODE	https://github.com/virtualeconomy/v-systems
PLATFORM	Custom
LANGUAGE	Scala
REQUEST DATE	June, 2020
REVISION DATE	June, 2020
METHODS	Static Analysis, and Manual Review, a comprehensive examination has been performed.

Source Of Truth

From **vsys smart contract document**:

Considering the technology development and industrial needs for smart contracts, V SYSTEMS will temporarily adopt the non-Turing-complete scripting language, so that smart contracts can be secure, resource-efficient, and easy to use and manage. In the near future, a Turing-complete model will eventually be adopted by V SYSTEMS.

- Smart contract ownership cannot be transferred, but the token issue right can be transferred. The contract creator has the final right to interpret the token issue right.
- The smart contract itself cannot be modified. It is a simple consensus and cannot be modified at will, but the parameters of some contracts can be changed. The contracts with modifiable parameters are relatively weak in consensus. These parameter revisions will provide choices and an advanced notice.

Scope Of Audit

CertiK was chosen by V Systems to audit the design and implementation of its smart contract technology based on VSYS chain. To ensure comprehensive protection, the source code has been analyzed by the proprietary CertiK formal verification engine and manually reviewed by our smart contract experts and engineers. That end-to-end process ensures proof of stability as well as a hands-on, engineering-focused process to close potential loopholes and recommend design changes in accordance with the best practices in the space.

Source Code SHA-256 Checksum

Contract

- **CallType.scala** \newline
c0749ca4dfb68a6d7e4501ca2188586215e7038f566bba5e75e2b9b523839b06
- **Contract.scala** \newline
901846cd2107b6dc7be69939ea6be4e91355e8bf0533f75fcc57ae6c8e7033b9
- **ContractDepositWithdraw.scala** \newline
e0976d56bb01152ab4a21c549300bd1a36f2fb6f3bacbb5a5deef762480958e5
- **ContractDepositWithdrawProductive.scala** \newline
d0c0854dc253ab0c89874fef922b00374e5bdf8b7d32633d7ade4a29df0148e8
- **ContractGen.scala** \newline
0c11a58f93edfbfbbb840d843cec73cd8a793066150d8036c72e9f4b15d7ed32
- **ContractPermitted.scala** \newline
e9ff852b4ad82e0a25ae0f5c0bda94f55b31f6eb79360a208d799a02ac14bd27
- **ContractSystem.scala** \newline
1ee61006c399404a6b56c488b4e3f2814dca745a299adc943ea44b7c81bb6470
- **DataEntry.scala** \newline
c49bc0f4cf1f7147b4afa0fca2c7eb22b5eaa46cc8bdcaa5a2d0c493140e59f6
- **DataType.scala** \newline
1a43fe7facb05fd6cd225f4b4455be1f2495e7b84b10eac191772f492aa7b79e
- **ExecutionContext.scala** \newline
bf3a1061976595ae026d6fedcaa9a358d217346222c7350b3c12a1e1804592ea

OpcDiff

- **AssertOpcDiff.scala** \newline
ec58ae24776979317c8099d89bb69e611f8a5568a00fe99c222651a630f376d5
- **BasicOpcDiff.scala** \newline
9eff8eca48e098b2aa5c6d2fe68204e065746d8fe39b4f64858083cc1f8eec6f
- **CDBVOpcDiff.scala** \newline
cf54b4bcc6a79b90230451d76f5050d1d7643855a05413fa66b625e7c83e73a2
- **CDBVROpcDiff.scala** \newline
6a994404c6975da8cffa80ed1629ea38526cd724bca6ef94d095130c89d84858
- **CallOpcDiff.scala** \newline
1680b67ec2f963d59ad63eeb09cf743f629d320cea98253730fcb05c2eb04148
- **CompareOpcDiff.scala** \newline
6860bfd38e7b42bdfb0c9a35b398b7265770cb3e8fbbb0d6aa3a58a867d4fff4
- **LoadOpcDiff.scala** \newline
aed0b93187b5bd80dd9d3c8c5047c60bdd3b4116497c92aa7aa3d5aa98712d6f
- **OpcDiff.scala** \newline
8f1416c30073c23955dde60fe2ec9fe754bd34a6b2b22b4ff4888863bb87b436
- **OpcDiffer.scala** \newline
147ae9590dba6813f35d43cc91ef8f53997eb09e323933e64440349296fd8cbd
- **OpcFuncDiffer.scala** \newline
0fd242d765b476028ffaa661ffcc695ff7bbf6f39db6abc19ef724641be86cb2
- **ReturnOpcDiff.scala** \newline
eba31d378c7edc70a2b7324144ff9f364063fe66ef75fb50b17bff464037f173
- **SystemTransferDiff.scala** \newline
718c7a53d0b07028effb30bf1ef0b6b5f139a17f89eaf4730432679c4c304ede
- **TDBAOpcDiff.scala** \newline
79e91b2618ae677c3540ecedc1fe4640156de6309da54e0e53522347231f75f8

- **TDBAROpDiff.scala** \newline
dd3a4a47f24fd9a2b8eab9e19ba302717bf686f18276364f4c42ea557835bcf7
- **TDBOpDiff.scala** \newline
e54a0188fc765b463ff0f3c9f7c94be66e038ca130d53dbb0f3219a0d9db897f
- **TDBROpDiff.scala** \newline
0e4f13e8e108f8189a5416afa3b9ad9d91e8119767bac8ba8778d0ef087cc84
4

Review Comments

- ContractDepositWithdrawProductive.scala
 - INFO
 - For depositTrigger and withdrawTrigger, the relation between tokenId and contractTokenId is defined to be equal, which seems not so intuitive.

blockchain/state/opcdiffs/

- **TDBAOpDiff.scala**
Token database rollback opcodes including deposit(), withdraw() and transfer(), where deposit() and withdraw() operation interact with contract address and the behavior of transfer() is like transferFrom() in Solidity.
 - INFO
 - Recommend ensuring the depositAmount always greater than 0 in deposit().
 - INFO
 - Recommend ensuring the withdrawAmount always greater than 0 in withdraw().
- **TDBAROpDiff.scala**
Get balance of given address.
 - INFO

- Recommend to check address in balance() is not zero address.
- TDBOpcDiff.scala
 - Token database operations that can create newToken and split by setting new unity.
 - [Question] Based on the knowledge that before splitting, token issuers should notice the centralized exchanges to stop all business and then update the unity.
We have some concerns on this unity change operation and would like to go deeper to the functionality of the split function if the balance would always be kept the same.
 - INFO
 - Recommend to replace magic numbers in parseBytes() with symbolic constants.
- TDBROpcDiff.scala
 - Set max, unity, total and desc states.
 - INFO
 - Recommend to replace magic numbers in parseBytes() with symbolic constants.

Best Practice

Design of smart contract development requires a particular engineering mindset. A failure in the initial construction can be catastrophic, and changing the project after the fact can be exceedingly difficult.

To ensure success and to avoid the challenges above, design of smart contracts should here to best practices at their conception. Below, we summarized a checklist of key points & vulnerability vectors that help to indicate a high overall quality of the current V Systems project. (✓ indicates satisfaction, × indicates dissatisfaction, – indicates inapplicability).

General

Logging

- [✓] Specify error cases by defining various classes and objects extends `ValidationError`
- [✓] Use status code to monitor transaction status

Arithmetic Vulnerability

Two's Complement / Integer underflow / overflow

- [✓] Use Math library with `addExact()` before all arithmetic operations to catch integer overflow and underflow errors

Floating Points and Precision

- [✓] Correct handling the right precision when dealing ratios and rates

Access & Privilege Control Vulnerability

Circuit Breaker

- [–] Provide pause functionality for control and emergency handling

Restriction

- [✓] Provide proper access control for functions
- [✓] Establish rate limiter for certain operations
- [✓] Restrict access to sensitive functions
- [-] Restrict permission to contract destruction
- [-] Establish [speed bumps](#) slow down some sensitive actions, any malicious actions occur, there is time to recover.

DoS Vulnerability

A type of attack that makes the contract inoperable within a certain period of time or permanently.

Unexpected Revert

- [✓] States would be changed if and only if the diffcodes passed all of the validation checks, so that functions would not be reverted in unexpected situations.

Human Factor Manipulation Vulnerability

Avoid state changes before validation checks

- [✓] States would be changed if and only if the diffcodes passed all of the validation checks.

Visibility Vulnerability

The visibility determines whether a function can be called externally by users, by other derived contracts, only internally or only externally.

- [✓] Specify the visibility of all functions in a contract, even if they are intentionally public

Incorrect Interface Vulnerability

A contract interface defines functions with a different type signature than the implementation, causing two different methods to be created. As a result, when the interface is called, the fallback method will be executed.

- [✓] Ensure the defined function signatures are match with the contract interface and implementation

Documentation

The presence of documentation helps keep track of all aspects of an application and it improves on the quality of a software product. Its main focuses are development, maintenance and knowledge transfer to other developers.

- [✓] Provide project README and execution guidance
- [✓] Provide inline comment for complex functions intention
- [✓] Provide instruction to initialize and execute the test files

Testing

Rigorous testing of components and systems, and their associated documentation, can help reduce the risk of failures occurring during operation. When defects are detected, and subsequently fixed, this contributes to the quality of the components or systems.

- [✓] Provide test scripts and coverage for potential scenarios

Overall we found the design of smart contracts based on opcodes to follow good practices. With the final update of source code and delivery of the audit report, we conclude that the design of smart contracts is structurally sound and not vulnerable to any classically known anti-patterns or security issues. The audit report itself is not necessarily a guarantee of correctness or trustworthiness, and we always recommend seeking multiple opinions, keep improving the codebase, and more test coverage and sandbox deployments.