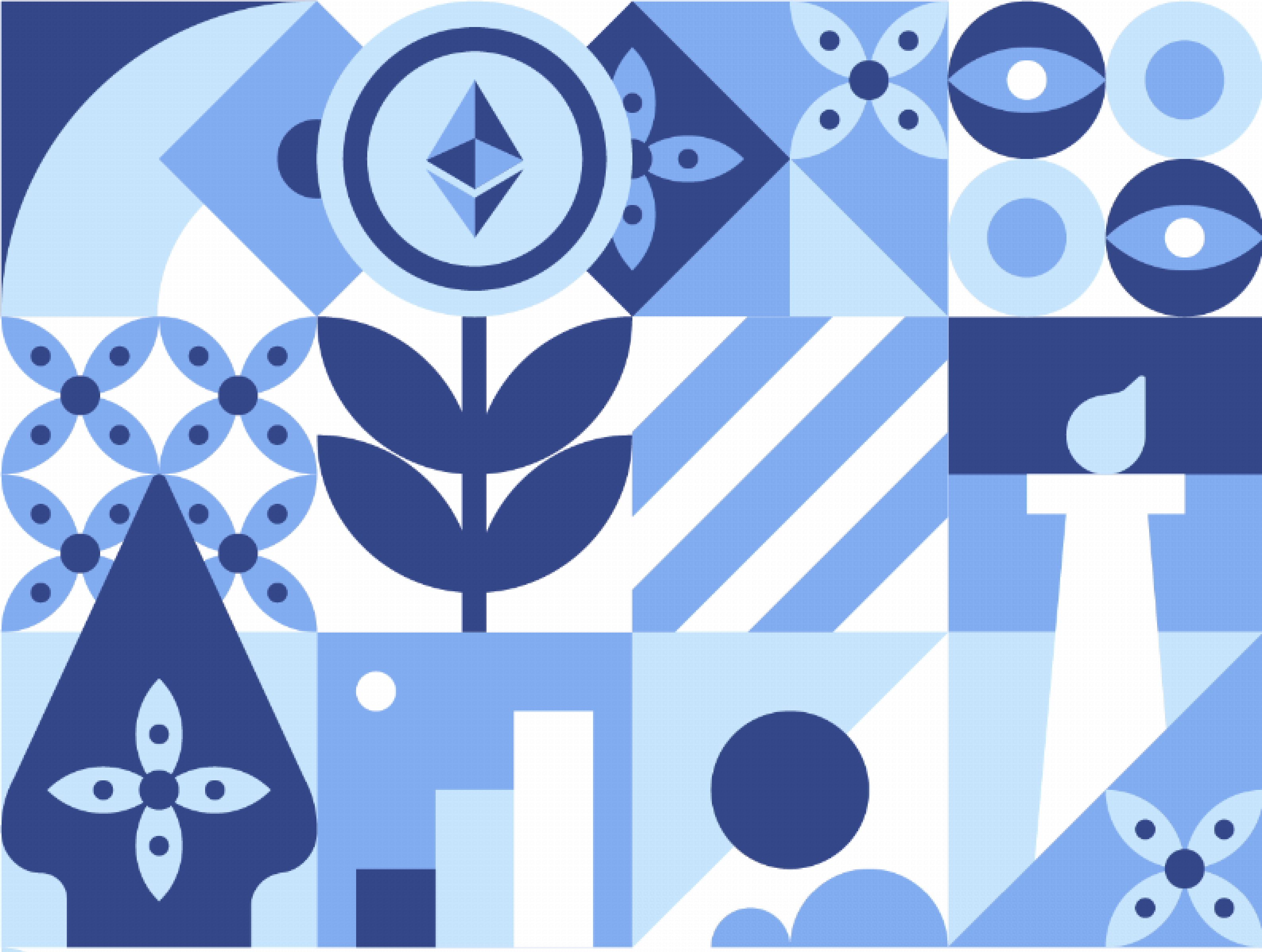

LoopringCoin V2

Smart Contract Audit Report

SYSFIXED

15 Nov 2019

TABLE OF CONTENTS

AUDITED DETAILS

Audited Project

Project name	Token ticker	Blockchain
LoopringCoin V2	LRC	Ethereum

Addresses

Contract address	0xbbbbca6a901c926f240b89eacb641d8aec7aeafd
Contract deployer address	0xBBbbCA6A901c926F240b89EacB641d8Aec7AEafD

Project Website

https://loopring.org/

Codebase

https://etherscan.io/address/0xbbbbca6a901c926f240b89eacb641d8aec7aeafd#code

SUMMARY

Loopring protocol is an open-source zkRollup protocol. It is a collection of Ethereum intelligent contracts and ZK circuits which describe how to build highly secure, highly-scalable order book-based DEXes, AMMs, and payment apps. It was the first rollup protocol deployed on Ethereum and has led the ecosystem into the era of Layer 2 scaling (L2). The current protocol version, v3.8, is our fifth deployed version (the first three were non-zkRollup). It provides a solution for the most unique challenge of all decentralized protocols - performance, or more precisely, much higher performance without a tradeoff in security.

Contract Summary

Documentation Quality

LoopringCoin V2 provides a very good documentation with standard of solidity base code.

- The technical description is provided clearly and structured and also dont have any high risk issue.

Code Quality

The Overall quality of the basecode is standard.

- Standard solidity basecode and rules are already followed by LoopringCoin V2 with the discovery of several low issues.

Test Coverage

Test coverage of the project is 100% (Through Codebase)

Audit Findings Summary

- SWC-100 SWC-108 | Explicitly define visibility for all state variables on lines 70, 71 and 72.
- SWC-102 | It is recommended to use a recent version of the Solidity compiler on lines 9.
- SWC-110 SWC-123 | It is recommended to use of revert(), assert(), and require() in Solidity, and the new REVERT opcode in the EVM on lines 60.

CONCLUSION

We have audited the LoopringCoin V2 project released on November 2019 to discover issues and identify potential security vulnerabilities in LoopringCoin V2 Project. This process is used to find technical issues and security loopholes which might be found in the smart contract.

The security audit report provides satisfactory results with low-risk issues.

The issues found in the LoopringCoin V2 smart contract code do not pose a considerable risk. The writing of the contract is close to the standard of writing contracts in general. The low-risk issues found an outdated compiler version is used, state variable visibility is not set, and an assertion violation was triggered. An outdated compiler version is used. The compiler version specified in the pragma directive may have known bugs. It is recommended to use the latest minor release of solc 0.5 or 0.6. For more information on Solidity compiler bug reports and fixes refer to https://github.com/ethereum/solidity/releases. It is best practice to set the visibility of state variables explicitly. The default visibility for "balances" is internal. Other possible visibility settings are public and private. It is possible to cause an assertion violation. Note that Solidity assert() statements should only be used to check invariants. Review the transaction trace generated for this issue and either make sure your program logic is correct, or use require() instead of assert() if your goal is to constrain user inputs or enforce preconditions. Remember to validate inputs from both callers (for instance, via passed arguments) and callees (for instance, via return values).

AUDIT RESULT

Article	Category	Description	Result
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	ISSUE FOUND
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	PASS
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	ISSUE FOUND
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	PASS
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	PASS
Unprotected Ether Withdrawal	SWC-105	Due to missing or insufficient access controls, malicious parties can withdraw from the contract.	PASS
SELFDESTRUCT Instruction	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	PASS
Reentrancy	SWC-107	Check effect interaction pattern should be followed if the code performs recursive call.	PASS
Uninitialized Storage Pointer	SWC-109	Uninitialized local storage variables can point to unexpected storage locations in the contract.	PASS
Assert Violation	SWC-110 SWC-123	Properly functioning code should never reach a failing assert statement.	ISSUE FOUND
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	PASS
Delegate call to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	PASS

DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless required.	PASS
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	PASS
Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	PASS
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	PASS
Signature Unique ID	SWC-117 SWC-121 SWC-122	Signed messages should always have a unique id. A transaction hash should not be used as a unique id.	PASS
Incorrect Constructor Name	SWC-118	Constructors are special functions that are called only once during the contract creation.	PASS
Shadowing State Variable	SWC-119	State variables should not be shadowed.	PASS
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	PASS
Write to Arbitrary Storage Location	SWC-124	The contract is responsible for ensuring that only authorized user or contract accounts may write to sensitive storage locations.	PASS
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order. The rule of thumb is to inherit contracts from more /general/ to more /specific/.	PASS
Insufficient Gas Griefing	SWC-126	Insufficient gas griefing attacks can be performed on contracts which accept data and use it in a sub-call on another contract.	PASS
Arbitrary Jump Function	SWC-127	As Solidity doesnt support pointer arithmetics, it is impossible to change such variable to an arbitrary value.	PASS

Typographical Error	SWC-129	A typographical error can occur for example when the intent of a defined operation is to sum a number to a variable.	PASS
Override control character	SWC-130	Malicious actors can use the Right-To-Left-Override unicode character to force RTL text rendering and confuse users as to the real intent of a contract.	PASS
Unused variables	SWC-131 SWC-135	Unused variables are allowed in Solidity and they do not pose a direct security issue.	PASS
Unexpected Ether balance	SWC-132	Contracts can behave erroneously when they strictly assume a specific Ether balance.	PASS
Hash Collisions Variable	SWC-133	Using abi.encodePacked() with multiple variable length arguments can, in certain situations, lead to a hash collision.	PASS
Hardcoded gas amount	SWC-134	The transfer() and send() functions forward a fixed amount of 2300 gas.	PASS
Unencrypted Private Data	SWC-136	It is a common misconception that private type variables cannot be read.	PASS

SMART CONTRACT ANALYSIS

Started	Thursday Nov 14 2019 03:37:03 GMT+0000 (Coordinated Universal Time)
Finished	Friday Nov 15 2019 00:03:03 GMT+0000 (Coordinated Universal Time)
Mode	Standard
Main Source File	LRC_v2.sol

Detected Issues

ID	Title	Severity	Status
SWC-102	AN OUTDATED COMPILER VERSION IS USED.	low	acknowledged
SWC-108	STATE VARIABLE VISIBILITY IS NOT SET.	low	acknowledged
SWC-108	STATE VARIABLE VISIBILITY IS NOT SET.	low	acknowledged
SWC-108	STATE VARIABLE VISIBILITY IS NOT SET.	low	acknowledged
SWC-110	AN ASSERTION VIOLATION WAS TRIGGERED.	low	acknowledged

SWC-102 | AN OUTDATED COMPILER VERSION IS USED.

LINE 9

low SEVERITY

The compiler version specified in the pragma directive may have known bugs. It is recommended to use the latest minor release of solc 0.5 or 0.6. For more information on Solidity compiler bug reports and fixes refer to https://github.com/ethereum/solidity/releases.

Source File

- LRC_v2.sol

Locations

```
8
9   pragma solidity 0.5.7;
10
11   /**
12   * @title ERC20Basic
13
```

SWC-108 | STATE VARIABLE VISIBILITY IS NOT SET.

LINE 70

low SEVERITY

It is best practice to set the visibility of state variables explicitly. The default visibility for "balances" is internal. Other possible visibility settings are public and private.

Source File

- LRC_v2.sol

Locations

```
69   using SafeMath for uint256;
70   mapping(address => uint256) balances;
71   uint256 totalSupply_;
72   uint256 burnedTotalNum_;
73
74
```

SWC-108 | STATE VARIABLE VISIBILITY IS NOT SET.

LINE 71

low SEVERITY

It is best practice to set the visibility of state variables explicitly. The default visibility for "totalSupply_" is internal. Other possible visibility settings are public and private.

Source File

- LRC_v2.sol

Locations

```
70   mapping(address => uint256) balances;
71   uint256 totalSupply_;
72   uint256 burnedTotalNum_;
73
74   /**
75
```

SWC-108 | STATE VARIABLE VISIBILITY IS NOT SET.

LINE 72

low SEVERITY

It is best practice to set the visibility of state variables explicitly. The default visibility for "burnedTotalNum_" is internal. Other possible visibility settings are public and private.

Source File

- LRC_v2.sol

Locations

```
71   uint256 totalSupply_;
72   uint256 burnedTotalNum_;
73
74   /**
75   * @dev total number of tokens in existence
76
```

SWC-110 | AN ASSERTION VIOLATION WAS TRIGGERED.

LINE 60

low SEVERITY

It is possible to cause an assertion violation. Note that Solidity assert() statements should only be used to check invariants. Review the transaction trace generated for this issue and either make sure your program logic is correct, or use require() instead of assert() if your goal is to constrain user inputs or enforce preconditions. Remember to validate inputs from both callers (for instance, via passed arguments) and callees (for instance, via return values).

Source File

- LRC_v2.sol

Locations

```
59   uint256 c = a + b;
60   assert(c >= a);
61   return c;
62   }
63   }
64
```

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 Sysfixed'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 Sysfixed 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 is a limited report on our findings based on our analysis, in accordance with good industry practice as of the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the below disclaimer below – please make sure to read it in full.

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.

This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and Sysfixed and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (Sysfixed) owe no duty of care.

ABOUT US

Sysfixed is a blockchain security certification organization established in 2021 with the objective to provide smart contract security services and verify their correctness in blockchain-based protocols. Sysfixed automatically scans for security vulnerabilities in Ethereum and other EVM-based blockchain smart contracts. Sysfixed a comprehensive range of analysis techniques—including static analysis, dynamic analysis, and symbolic execution—can accurately detect security vulnerabilities to provide an in-depth analysis report. With a vibrant ecosystem of world-class integration partners that amplify developer productivity, Sysfixed can be utilized in all phases of your project's lifecycle. Our team of security experts is dedicated to the research and improvement of our tools and techniques used to fortify your code.