SOLIDIFIED

Audit Report for Xaya - March 29, 2021

Summary

Audit Report prepared by Solidified covering the WCHI token smart contracts.

Process and Delivery

Three (2) independent Solidified experts performed an unbiased and isolated audit of the code below. The final debrief took place on 29 March 2021, and the results are presented here.

Audited Files

The source code has been supplied in the form of a GitHub repository:

https://github.com/xaya/wchi

Commit number: `adc0207f3020d92f3b13790b45ec46c1b430cbb7`

The scope of the audit was limited to the following files:

```
contracts
├── HTLCs.sol
├── IERC20.sol
├── IHTLCs.sol
├── IWCHI.sol
└── WCHI.sol
```

Intended Behavior

The smart contracts implement an ERC-20 token designed to represent the CHI token as a wrapped asset on the Ethereum blockchain. The fully token supply is minted to the contract deployer (the Xaya team) who distribute the token to Chi holders.

SOLIDIFIED

Audit Report for Xaya - March 29, 2021

Code Complexity and Test Coverage

Smart contract audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of a smart contract system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**.

Note, that high complexity or lower test coverage does equate to a higher risk. Certain bugs are more easily detected in unit testing than a security audit and vice versa. It is, therefore, more likely that undetected issues remain if the test coverage is low or non-existent.

Criteria	Status	Comment
Code complexity	Low	-
Code readability and clarity	High	-
Level of Documentation	High	-
Test Coverage	High	-

Test coverage report (Pool Package):

```
-------------|----------|----------|----------|----------|----------------|
File         |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
-------------|----------|----------|----------|----------|----------------|
 contracts/  |      100 |    92.31 |      100 |      100 |                |
  HTLCs.sol  |      100 |    85.71 |      100 |      100 |                |
  IERC20.sol |      100 |      100 |      100 |      100 |                |
  IHTLCs.sol |      100 |      100 |      100 |      100 |                |
  IWCHI.sol  |      100 |      100 |      100 |      100 |                |
  WCHI.sol   |      100 |      100 |      100 |      100 |                |
-------------|----------|----------|----------|----------|----------------|
All files    |      100 |    92.31 |      100 |      100 |                |
-------------|----------|----------|----------|----------|----------------|
```

Issues Found

Solidified found that the WCHI contracts contain no critical issue, no major issue, 1 minor issue1, in addition to 1 informational note.

We recommend all issues are amended, while the notes are up to the team's discretion, as they refer to best practices.

Issue #	Description	Severity	Status
1	WCHI.sol: Token vulnerable to ERC-20 approve attack	Minor	Pending
2	WCHI.sol: Function approve() does not validate spender	Note	-

SOLIDIFIED

Audit Report for Xaya - March 29, 2021

Critical Issues

No critical issues have been found.

Major Issues

No major issues have been found.

Minor Issues

1. WCHI.sol: Token vulnerable to ERC-20 approve attack

The ERC-20 standard has a flaw related to the `approve()` function. This allows overspending in the case of subsequent approvals, as described here: https://docs.google.com/document/d/1YLPtQxZu1UAvO9cZ1O2RPXBbT0mooh4DYKjA_jp-RLM/edit#

Recommendation

There are several potential solutions to this. One option is to require users to call `approve()` with a zero amount before calling it again with the new value, requiring an additional transaction. Another option is to add `increaseAllowance()` and `decreaseAllowance()` methods, such as those used in the OpenZeppelin ERC-20 implementation (https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/ERC20.sol#L170).

Informative Notes

2. WCHI.sol: Function approve() does not validate spender

Recommendation
Consider checking that `spender != address(0)` before allowing the function to continue with allowance approval.

Audit Report for Xaya - March 29, 2021

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of Xaya or its products. This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors of Solidified platform from legal and financial liability.

Solidified Technologies Inc.