CryptoSec

SECURITY FOR BLOCKCHAIN

REPORT
SMART CONTRACTS AUDIT RESULTS
FOR WANCHAIN

Change history

Version	Date	Author	Changes
1.0	07.15.2018	CryptoSec	Report created

Confirmation

Name	Company	Position	Location	Email

Contacts

Company name	Tomahawk Technologies Inc.
Address	1801 Wedemeyer, San Francisco, 94129
PGP	CCFD 364F 0180 287E 4DD6 A0AB 6CDF F9EC 1BAE D618
Email	info@cryptosec.us
Web	https://www.cryptosec.us

Table of Contents

1 Limitations on disclosure and usage of this report

This report has been developed by the company CryptoSec (the Service Provider) based on the result of security audit of Ethereum Smart Contracts defined by Wanchain (the Client). The document contains information on discovered vulnerabilities, their severity and methods of exploiting those vulnerabilities, discovered in the process of the audit.

The information, presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of the Client.

If you are not the intended receiver of this document, remember that any disclosure, copying or dissemination of it is forbidden.

2 Acronyms and Abbreviations

ETHEREUM – An open source platform for creating decentralized online services (called DApps or Decentralized applications) based on the blockchain that use smart contracts.

ETH (ether) – The cryptocurrency and token in the Ethereum blockchain that is used for payment of transactions and computing services on the Ethereum network.

SOLIDITY – An object-oriented programming language for creating smart contracts defined for use on Ethereum platform.

SMART CONTRACT – A computer algorithm designed to create and support contracts recorded on a blockchain.

ERC20 – The Ethereum token standard used for Ethereum smart contracts. It is a set of rules for the implementation of Ethereum tokens.

SAFEMATH – A Solidity library created for secure mathematical operations.

SOLC – A compiler for Solidity.

3 Introduction

3.1 Vulnerability Level

- Low severity – A vulnerability that does not have a significant impact on the use of the contract and is probably subjective.
- Medium severity – A vulnerability that could affect the desired outcome of executing the contract in certain scenarios.
- High severity – A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.
- Critical severity – A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates the risk that the contract may be broken.

4 Main Results

Source code:
1) https://github.com/wanchain/wanchain-rpc-test
Commit: f728a3b81500d40088866acbd6f093d890e7a79f
2) https://github.com/wanchain/wanchain-crosschain-contracts
Commit: 72c11e5b46256d9b209f40f3727f212e44ab8b09

4.1 Vulnerability table for all smart contracts

Contract name	Vulnerability level			
	Critical	High	Medium	Low
wan-alt.sol	-	-	3	8
wan-token.sol	-	-	-	-
ConvertLib.sol	-	-	-	2
ERC20Protocol.sol	-	-	-	1
HTLCBase.sol	-	-	-	1
HTLCETH.sol	-	-	-	1
HTLCWETH.sol	-	-	-	3
Halt.sol	-	-	-	-
Owned.sol	-	-	-	1
SafeMath.sol	-	-	-	2
StandardToken.sol	-	-	-	1
StoremanGroupAdmin.sol	-	-	-	1
WETH.sol	-	-	-	-
WETHManager.sol	-	-	-	-
Genaral comments	-	-	1	5
Totals	-	-	4	26

4.2 Evaluation of wan-alt.sol

4.2.1 Medium Severity

✗ There is no protection for transferring tokens to the address of the contract (this situation is quite common). And since the contract does not provide any functions for handling tokens on the contract balance, any tokens accidentally transferred to the contract address will be lost.

✓ *Recommendation*: Add protection function for this case.

✗ Currently, approve method does not allow to set allowance to non-zero value if current allowance is non-zero. This directly violates ERC20 standard that says: “Allows _spender to withdraw from your account multiple times, up to the _value amount. If this function is called again it overwrites the current allowance with _value”.

EIP-20 pull request adds the following to the description of approve method: To prevent attack vectors like the one described here and discussed here, clients SHOULD make sure to create user interfaces in such a way that they set the allowance first to 0 before setting it to another value for the same spender. THOUGH The contract itself shouldn't enforce it, to allow backwards compatibility with contracts deployed before.

So current implementation of approve violates EIP-20 as well.

✓ *Recommendation:* Remove a internal “assert” check.

✗ Functions “transfer” and “transferFrom” has the address as one of the input parameters, but don’t check the address for a null value (in the Ethereum virtual machine, omitted values are interpreted as null values), which means that token loss is possible when using this function (they will be sent to the address 0x0).

✓ Recommendation: Implement a null address check.

4.2.2 Low Severity

✗ This file contains the "approve" function, which has a check to deflect attack vectors for the Approve/TransferFrom functions. However, there are no functions to increase or decrease the amount of "approved" tokens, which leads to less flexibility in the use of tokens (the number of "approved" tokens must always be reduced to 0 first, which means there are two function calls instead of one).

✓ *Recommendation*: To reduce overhead costs, implement the increaseApproval and decreaseApproval functions wherever this is appropriate.

✗ This file contains "transfer" and “transferFrom” functions, which return “true” if transfer is successful and “false” if transfer is not successful. But we return “false” for successful transaction in blockchain. Thus user should check answer from function.

✓ *Recommendation*: replace “if” control structure by “require”.

✗ This file contains "safeMath" library with “mul” function, which is not work for case “a” is null. You can improve mul performance and reduce gas cost.

✓ *Recommendation*: remove check “a == 0” from “assert” and update “mul” function for case “a” is null.

- ✗ This file contains "safeMath" library with "div" function. Solidity automatically throws when dividing by 0.
- ✓ *Recommendation*: remove checks for this function.

- ✗ The "constant" identifier has been deprecated, and now the "view" or "pure" identifier is used instead. However, the "constant" identifier is currently an alias of the "view" identifier. But functions can be declared "pure" in which case they promise not to read from or modify the state.
- ✓ *Recommendation*: Replace the "constant" identifier with the "pure" or "view" identifier for all functions.

- ✗ This file contains "safeMath" library with functions "max64", "min64", "max256" and "min256", whick are not used in contract.
- ✓ *Recommendation*: Delete these functions.

- ✗ Function "buyWanCoin" check the address for a null value, but it compares the address data type with a null value. At this time, implicit type conversion is being performed by the compiler, but we recommend to do this explicitly.
- ✓ *Recommendation*: Before 0, add an explicit conversion to the address type (address(0)).

- ✗ Many functions have not a visibility specifier. Default visibility specifier is "public".
- ✓ *Recommendation*: Add visibility specifier for each function.

4.3 Evaluation of wan-token.sol

Please see Evaluation of wan-alt.sol.

4.4 Evaluation of ConvertLib.sol

4.4.1 Low Severity

- ✗ Function "convert" have not a visibility specifier. Default visibility specifier is "public".
- ✓ *Recommendation*: Add visibility specifier for "convert" function.

- ✗ Functions can be declared "pure" in which case they promise not to read from or modify the state..
- ✓ *Recommendation*: Add "pure" identifier for "convert" function.

4.5 Evaluation of ERC20Protocol.sol

4.5.1 Low Severity

- ✗ The "constant" identifier has been deprecated, and now the "view" or "pure" identifier is used instead. However, the "constant" identifier is currently an alias of the "view" identifier. Also functions can be declared "view" in which case they promise not to modify the state.

✓ *Recommendation*: Replace the "constant" identifier with the "view" identifier for "allowance" and "balanceOf" functions.

4.6 Evaluation of HTLCBase.sol

4.6.1 Low Severity

✗ Function "refundHTLCTx" uses "keccak256" function, and send data with type "bytes32" to "keccak256" function. But "keccak256" function only accepts a single "bytes" argument.

✓ *Recommendation*: Use "abi.encodePacked(...)" or a similar function to encode the data.

4.7 Evaluation of HTLCETH.sol

4.7.1 Low Severity

✗ Functions "eth2wethLock" and "weth2ethLock" has the address as one of the input parameters, but don't check the address for a null value (in the Ethereum virtual machine, omitted values are interpreted as null values), which means that token loss is possible when using this function (they will be sent to the address 0x0).

✓ *Recommendation*: Implement a null address check.

4.8 Evaluation of HTLCWETH.sol

4.8.1 Low Severity

✗ This file contains functions, which has "if" control structures with "revert" function.

✓ *Recommendation*: Replace "if" control structures with "revert" function by "require" function.

✗ This file contains function "getWeth2EthFee", which uses a "var" keyword, this keyword is deprecated in solidity version 0.4.20.

✓ *Recommendation*: Remove "var" keyword and update function.

✗ This file contains "StoremanGroupAdmin" and "WETHManager" contracts without implementation. You can replace this contracts by interfaces.

✓ *Recommendation*: Replace contracts by interfaces.

4.9 Evaluation of Halt.sol

This library meets modern security requirements and does not require changes.

4.10 Evaluation of Owned.sol

4.10.1 Low Severity

- ✗ This file contains the "acceptOwnership" function, which has “if” block for checking a input address. This function is executed independently from the input address and gas is spent.
- ✓ *Recommendation*: Replace “if” block with “require” for gas saving.

4.11 Evaluation of SafeMath.sol

4.11.1 Low Severity

- ✗ This file contains "safeMath" library with “mul” function, which is not work for case “a” is null. You can improve mul performance and reduce gas cost.
- ✓ *Recommendation*: remove check “a == 0” from “assert” and update “mul” function for case “a” is null.

- ✗ This file contains "safeMath" library with “div” function. Solidity automatically throws when dividing by 0.
- ✓ *Recommendation*: remove checks for this function.

4.12 Evaluation of StandardToken.sol

Also please see Evaluation of wan-alt.sol.

4.12.1 Low Severity

- ✗ This file contains “onlyPayloadSize” modifier functions, which protect from short address attack. An honorable effort, indeed, but it was done at the wrong layer and in response to a single isolated case of very poor validation in the upper layers.
- ✓ *Recommendation*: Remove “onlyPayloadSize” modifier, and remove this modifier for each function.

4.13 Evaluation of StoremanGroupAdmin.sol

4.13.1 Low Severity

- ✗ Function “deposit”, “applyUnregister” and “smgWithdrawAble” use “keccak256” function, and send data with type “bytes32” to “keccak256” function. But “keccak256” function only accepts a single "bytes" argument.

✓ *Recommendation*: Use "abi.encodePacked(...)" or a similar function to encode the data.

4.14 Evaluation of WETH.sol

This library meets modern security requirements and does not require changes.

4.15 Evaluation of WETHManager.sol

This library meets modern security requirements and does not require changes.

4.16 General Comments

4.16.1 Medium Severity

✗ Current code is written for previous versions of solc. Use latest version of Solidity.

✓ *Recommendation*: Use solc version 0.4.24.

4.16.2 Low Severity

✗ Contracts should be deployed with the same compiler version and flags that they have been tested the most with. Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, the latest compiler which may have higher risks of undiscovered bugs. Contracts may also be deployed by others and the pragma indicates the compiler version intended by the original authors.

✓ *Recommendation*: Lock pragmas to specific compiler version (replace “^0.4.11” by “0.4.11”).

✓ Defining constructors as functions with the same name as the contract is deprecated in solidity version 0.4.22.

✓ *Recommendation*: Replace, for example, function “HTLCBase” by "constructor(...) { ... }".

✗ This contracts contains “now” (alias for “block.timestamp”) thus miners can perform some manipulation. In this case miner manipulation risk is really low.

✓ *Recommendation*: Consider the potential risk and use “block.number” if necessary.

✗ For consistency in calculations, the “SafeMath” library should be used everywhere instead of “*”, “/” and “-”.

✔ *Recommendation*: Update “SafeMath” library and use for all arithmetic actions.

✗ Coding style of the project has several observations.

✓ *Recommendation*: Improve code quality. Even better, use a coding style from solidity doc.

5 Conclusion

Serious vulnerabilities were not detected. However this contract has minor issues that may affect certain functions or cause inconvenience when using it. However, they will not prevent the contract from functioning as intended; the smart contract has been tested.

6 Tools Used

The audit of the contract code was conducted using:

- the Remix IDE (http://remix.ethereum.org);

The Solidity compiler version used was 0.4.24 (the most recent stable version).