CryptoSec

SECURITY FOR BLOCKCHAIN

REPORT
XYZ TOKEN SMART CONTRACTS AUDIT RESULTS
FOR ABC INC

Change history

Version	Date	Author	Changes
1.0	27.09.2017	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

CryptoSec
SECURITY FOR BLOCKCHAIN

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 ABC INC (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

4.1 Vulnerability table for all smart contracts

Contract name	Vulnerability level			
	Critical	High	Medium	Low
TokenBasic.sol	-	-	1	4
TokenExtension1.sol	-	-	-	2
TokenExtension2.sol	-	-	-	1
TokenExtension3.sol	-	-	-	-
SafeMath.sol	-	-	-	-
Genaral comments	-	-	2	4
Totals	-	-	3	11

4.2 Evaluation of TokenBasic.sol

4.2.1 Medium Severity

- ✗ This file contains the "transfer" and “transferFrom” functions, which have 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

- ✗ The "sha3" hash function has been deprecated, and the "keccak256" hash function is used instead. However, the "sha3" function is currently an alias for the "keccak256" function.
- ✓ Recommendation: Replace the "sha3" hash function with the "keccak256" hash function.

- ✗ The "addressNotNull" modifier checks the address for a null value, but it compares the address data type with a null value. At this time, implicit type conversion is performed by the compiler, but we recommend doing this explicitly.
- ✓ Recommendation: Before 0, add an explicit conversion to the address type (address(0)).

- ✗ The "validRequirement" modifier checks two values and has three conditions. However, the "_ownersCount > 0" condition is not necessary, because the "_ownersCount > = _required" check is performed later, but the condition "_required > 1" is useful.
- ✓ Recommendation: Remove the unnecessary condition "_ownersCount > 0".

- ✗ 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.

4.3 Evaluation of TokenExtension1.sol

4.3.1 Low Severity

- ✗ This contract contains the "onlyPayloadSize" modifier, which protects against one of the known types of attacks. However, the use of this modifier is not rational in some cases.
- ✓ Recommendation: Consider performing this check at a higher abstraction level.

- ✗ This contract contains the "transfer" function, which has the number of transmitted tokens as one of the input parameters, but it does not check the balance at the sender's address to verify that enough tokens are available. The level of severity is low, since this test is in the SafeMath library, which is not apparent.
- ✓ Recommendation: Implement a balance check for the required number of tokens.

4.4 Evaluation of TokenExtension2.sol

4.4.1 Low Severity

- ✗ This contract contains timestamps and “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.

4.5 Evaluation of TokenExtension3.sol

This contract does not require changes.

4.6 Evaluation of SafeMath.sol

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

4.7 General Comments

4.7.1 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.

- ✗ Current code is written for old versions of solc. Use latest version of Solidity.
- ✓ Recommendation: Use solc version 0.4.20.

4.7.2 Low Severity

- ✗ For consistency in calculations, the SafeMath library should be used everywhere instead of "++" of "--".
- ✓ Recommendation: Use SafeMath library for all arithmetic actions.

- ✗ There are several "magic constants" in the contract code. See lines 143 in TokenExtension1.sol and line 225 in TokenExtension2.sol. Use of magic constants reduces code readability and makes it harder to understand code intention.
- ✓ Recommendation: Extract magic constants into contract constants.

- ✗ Code quality of the project is low, which made auditing it hard.
- ✓ Recommendation: improve code quality. Even better, use a coding style from solidity doc.

- ✗ Commented code adds clutter for the reader and creates unnecessary confusion.
- ✓ Recommendation: Remove all commented functions.

5 Conclusion

Serious vulnerabilities were not detected. Also this contract has minor recommendations that may affect certain functions or cause inconveniences when using the contract. However, they will not prevent the contract from functioning as intended.

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 0.4.20 (the most recent stable version).