GETTY/IO

Smart Contract Audit for:

TronWatch Market (TWM)

Version 1.8.0: April 29, 2019

Table of Contents

Getty/IO Inc. 2019
2711 Centerville Road, STE 400,
Wilmington, Delaware, DE, 19808

About US

Getty/IO is an innovative remote IT consulting firm that combines software and domain expertise to build auto scalable and high performance web and mobile products. We specialise in modern Javascript technologies, lean and highly scalable backends and blockchain technologies, building outstanding products for our customers world-wide.

Born in South America, GETTY has become the largest remote development firm in the region. We are a global company helping startups and enterprises from all around the world scale their development teams by providing them the top remote developers and consultants, having recently been awarded AWS Think Big award:

- Frontend and Backend Software Development
- Cloud, DevOps & Software Architecture - winners of
- Blockchain, Big Data & Data Science
- Smart Contract Development, Review & Audit

Our professionals are able to integrate to our Customer's teams and add value from day one. Agile methodologies, experience helping dozens of teams building products and constant coaching and mentoring ensure all our engagements help our customers become successful.

In our hands your entire process is safe, without hassle and as seamless as it can be. With the collaboration of our team of experts, you can expect to achieve much more. We are present in the United States, Canada, Brazil, Chile, Peru and Belgium.

TronWallet is a secure p2p crypto wallet and exchange for TRON, built by GETTY. For further information and to download, please visit https://www.tronwallet.me

Abstract

Getty/IO and TronWallet Teams have been appointed by the TronWatchMarket Team to carry out the audit of it's smart contracts. **TronWatch Market** is a Decentralized Exchange (DEX) which will allow users to trade any Tron token safely and securely.

Disclaimer

Our responsibility is to express an opinion on TWM smart contracts based on our audit. We perform the audit to obtain reasonable assurance about whether the smart contract is safe, resilient, if it works in accordance with the specifications. This audit does not give any warranties on the security of the code. One audit cannot be considered enough. Security audit is not an investment advice.

Methodology

Tests were conducted using the **Tron Testnet** and **Tron Mainnet** environments. A set of different techniques were used to achieve this objective, including:

Attack: A focused attempt to evaluate the quality, especially reliability, of a test object, attempting to force specific failures.

Black-box testing: Functional or non-functional tests, without reference to the internal structure of the component or system.

Boundary value analysis: A black-box test design technique in which test cases are designed based on threshold values.

Code coverage: The code analysis that determines which parts of the software were run (covered) by the test suite and which parts were not executed.

Complexity: Tests the degree to which a component or system has a design and / or internal structure that is difficult to understand, maintain and verify.

Configuration control: Test the configuration management of a smart contract, which consists of evaluation, coordination, approval or disapproval, and implementation of changes to configuration items after formal establishment of their configuration.

Automated Audit: An automated audit using open source static and non-static analysis tools.

Manual Audit: A manual audit on smart contracts, code compliance, tests and code coverage and elaborate the report with all the gathered information.

NOTE: This report is subject to updates to reflect issues that can be found during new audits tests or network updates.

1 Introduction

TronWatch Market is a decentralized exchange (DEX) that will allow users to exchange any Tron token safely. Getty/IO and TronWallet Teams have been appointed by the TronWatchMarket Team to carry out the audit of it's smart contracts. The audit was conducted from March 18, 2019 to April 26, 2019.

1.1 Audit Dashboard

Project	TronWatch Market
Auditors	Bruno Campos Diogenes Ianakiara David Ianakiara Marlon Gomes Calebe Santos Victor Pereira
Github	Link
Languages	Solidity, JavaScript
Networks	Tron Testnet / Tron Mainnet
Date	2019-04-01 to 2019-04-30

Issues Found

	Low	Mid	High	Critical
Open	**3**	**4**	**1**	**0**
Resolved	**3**	**4**	**0**	**0**

1.2 Smart Contract Audit Goals

The audit goal is to verify and analyze the smart contracts in four categories:

Smart Contract Security

A complete review to identify security vulnerabilities related issues within each TWM contract.

Smart Contract Code Quality

A complete review of the smart contract tests, and execution environments.

Smart Contract Correctness

A complete review of the smart contract source code and how it operates. Verify implementation with white paper.

Smart Contract Software Architecture

Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.

1.3 System Overview

Documentation

The following documentation was available to the audit team:

- The Whitepaper: high level information on the TronWatch Market contract system
- Live Beta version
- TWM smart contract code

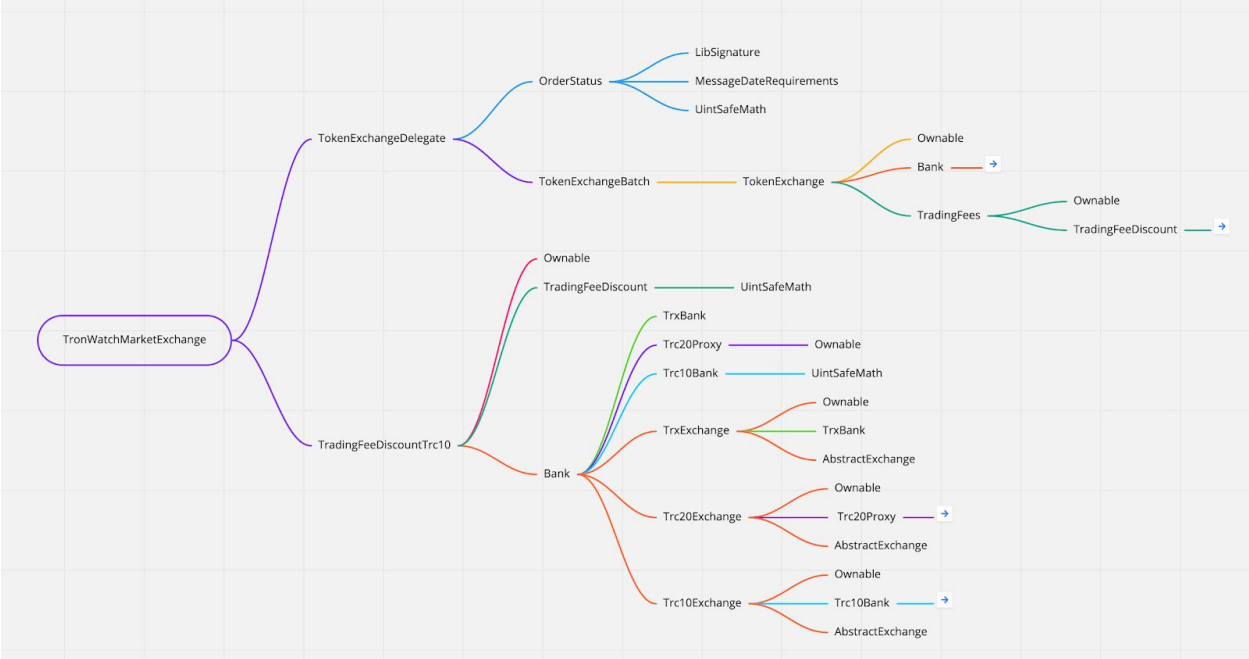
Scope

The audit focus was on the projects files, and test suite found in the following directories of the repository:

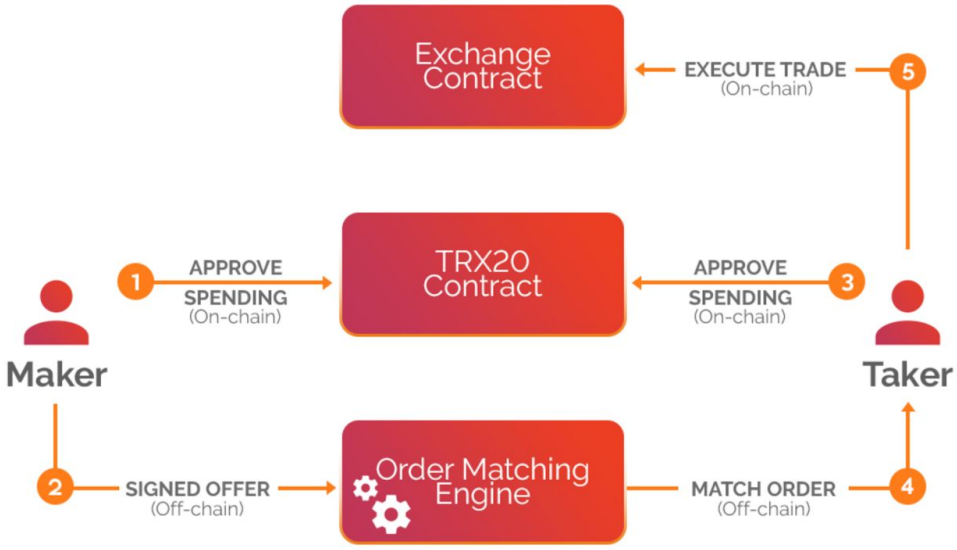
Directory	Commit hash	Commit Date
TronWatchMarket-Contracts/contracts/*.sol TronWatchMarket-Contracts/test/	1c5e722af951f512a8edadbaf3b15c43699cecfb	14th April 2019

Architecture

The call tree from the smart contracts is represented by the following graph. Here we can note that the **general** contracts are common to a lot of **core (trading** and **exchange)** contracts.



How it works ?



Maker Buy / Taker Sell Order for a TRX Pair

Operation description:

1. **On-chain**: Maker approves the Exchange contract to spend their TRX20 tokens
2. Off-chain: Maker signs a maker offer and shares it with the order matching engine
3. **On-chain**: Taker approves the Exchange contract to spend their TRX20 tokens
4. Off-chain: Taker selects one or more maker offers from the order matching engine
5. **On-chain**: The taker submits a transaction for the Exchange contract to execute the Trade

The on-chain/in-scope items can be divided into the following distinct parts:

- Trading: Contains the bulk of the business logic. It is the entry point for filling orders, canceling orders, calling transactions.
- Token: Manages transfers of TRC10 tokens, TWMToken.
- General: Contains the utils and common methods used by the **core** of TronWatch Market contracts like: LibSignature, Ownable, UintMath, UintSafeMath

- Exchange: Represent the direct call from Trading and all Exchange operations, it is the entity enabled to manage on a raw level.

1.4 Audit Key Observations

The first discoveries found were in the complex protocol, in it we can appreciate a high quality in the codes used. The following are noteworthy:

- The specifications documents are well written and denote exhaustive work. In order to have a good visualization of the system, the use of interaction diagrams was of great support.
- The comments of the code are adequate, specifically in the sections where the developer's understanding is necessary.
- The reentrancy cases were tested and they are being handled by the code to avoid risk to the business or even to their own contract, all cases were documented by the contracts' own code.
- We understand that a smart contract upgrade is not expected to happen. But, since it is a new software, it is necessary to evaluate how to upgrade in case of major changes on the network, user experience and evolution of the platform. A practical solution was proposed in 2016 and it solves the problems of migrating storage by separating the storage from the smart contract logic.
- Considering at the time of this audit the latest version of Tron's solidity is 0.4.25 Developer should make sure it is ready to use that without breaking any part of the code. This change will optimize trc10 transferToken function and will restrict tokenId range.

1.5 Recommendations

It is recommendable to address all issues uncovered during this audit process, regardless of the severity level. A common approach is to focus and address each issue based on severity level, starting with **Major** issues, then **Medium** and finally those issues with **Minor** severity.

Optional:

1. **Locked money:** Contracts programmed to receive tokens should implement a way to withdraw it, i.e., call transfer, send, or call.value at least once.
2. **Add a Circuit breaker**: If something goes wrong, the owner should be able to pause the contract.

2 Issue Overview

The following table contains an overview of all the issues discovered during the audit. The current (GitHub) status of the discovered problems was also added to the table.

Item	Title	Status	Severity
3.1	Compiler version not fixed	Resolved	Mid
3.2	Replace multiple return values with a struct	Resolved	Low
3.3	Costly loop	Resolved	Mid
3.4	Hardcoded address	Resolved	Mid
3.5	Private modifier	Resolved	Mid
3.6	Overpowered User	Resolved	High
3.7	Non-initialized return value	Resolved	Low
3.8	View-function should not change	Resolved	Low

	state		
3.9	Using assembly	Resolved	Low

3 Issues Detail

For each issue discovered, a detailed description, background and explanation is provided in order to issue a Remediation / Recommendation

3.1 Compiler version not fixed

Severity	Status	Link	Auditor Comment
Mid	Resolved	[Link Here]	Specify the exact compiler version to 0.4.25 on all TWM files.

Description

The last version of TRON Solidity is 0.4.25. Some source files enable contract to compiles with versions above the last version.

```
pragma solidity ^0.4.23;

import "../general/Ownable.sol";
import "../trading/Bank.sol";
import "../trading/TradingFees.sol";
```

TWM Sources

AbstractExchange.sol - ln.1	TWMToken.sol - ln.1
TokenExchange.sol - ln.1	Trc10Proxy.sol - ln.1
TokenExchangeBatch.sol - ln.1	Bank.sol - ln.1
TokenExchangeDelegate.sol - ln.1	MessageDateRequirements.sol - ln.1
Trc10Exchange.sol - ln.1	OfferStatus.sol - ln.1
Trc20Exchange.sol - ln.1	OrderStatus.sol - ln.1
TrxExchange.sol - ln.1	TradingFeeDiscount - ln.1
LibSignature.sol - ln.1	TradingFeeDiscountTrc10.sol - ln.1
Ownable.sol - ln.1	TradingFees.sol - ln.1
UintMath.sol - ln.1	Trc10Bank.sol - ln.1
UintSafeMath.sol - ln.1	Trc20Proxy.sol - ln.1
IERC20.sol - ln.1	ListingManager.sol - ln.1
TronWatchMarketExchange.sol - ln.1	Migrations.sol - ln.1
VoteForToken.sol - ln.1	TokenVesting.sol - ln.1

Recommendations

- It is recommended to follow the latter example, as future compiler versions may handle certain language constructions in a way the developer did not foresee.
- Use TRON version 0.4.25

References

- https://tool.smartdec.net/knowledge/SOLIDITY_PRAGMAS_VERSION

3.2 Replace multiple return values with a struct

Severity	Status	Link	Auditor Comment
Low	Resolved	[Link Here]	The use of ***struct*** is recommended for internal functions which can returns many parameters at once.

Description

We recommend the use of ***struct*** as possible for internal functions which can returns many parameters at once since It can improve code readability. In the following example the internal function **_calcTradeQuantity** returns several parameters:

```
function _calcTradeQuantity(
   ...
) internal view
returns (uint baseAmount, uint quoteAmount) { ... }
```

Source

AbstractExchange.sol - ln.59

AbstractExchange.sol - ln.117

AbstractExchange.sol - ln.188

References

- https://tool.smartdec.net/knowledge/SOLIDITY_SHOULD_RETURN_STRUCT

3.3 Costly Loop

Severity	Status	Link	Auditor Comment
Mid	Resolved	[Link Here]	Avoid loops with big or unknown number of steps.

Description

Costly loops can turn into a problem when the total number of energy consumed in a block is bigger the imposed limit by the TRON blockchain. Costly loop functions can exceeds the block energy limit, and transactions on this block can never be confirmed.

```
function _getTakerHash(
 ...
) internal view
returns (bytes32)
{
  ...
 for(uint i = 0; i < count; i++) {  ...  }
 return keccak256(...)  }
```

TWM Sources

TokenExchangeDelegate.sol - ln.155

LibSignature.sol - ln.63

Recommendations

Avoid loops with big or unknown number of steps.

References

- https://tool.smartdec.net/knowledge/SOLIDITY_GAS_LIMIT_IN_LOOPS

3.4 Hardcoded Address

Severity	Status	Link	Auditor Comment
Mid	Open	[Link Here]	Please check hardcoded address and it's usage.

Description

The contract contains unknown address. This address might be used for some malicious activity.

```
function _burnTrc10
...
{
  address(0x77944D19C052B73Ee2286823AA83F8138cb7032f) .transferToken(_amount, _token);
}
```

TWM Sources

Ownable.sol - ln.51

Trc10Proxy.sol - ln.44

TrcBank10.sol - ln.130

VoteForToken.sol - ln.20

Recommendations

We recommend to check the address. Also, it is required to check the code of the called contract for vulnerabilities. Make sure to double check the address on **Trc10Proxy.sol.**

References

- https://tool.smartdec.net/knowledge/SOLIDITY_ADDRESS_HARDCODED

3.5 Private modifier

Severity	Status	Link	Auditor Comment
Mid	Open	[Link Here]	Private modifier does not make a variable invisible.

Description

Private modifier in Solidity does not make a variable invisible.

```
uint private tronWatchMarketToken
```

TWM Sources

Ownable.sol - ln.19

TWMToken.sol - ln.6

MessageDateRequirements.sol - ln.16

OrderStatus.sol - ln.24

TradingFeeDiscount - ln. 23, 25, 27

TradingFees.sol - ln.44, 45, 46, 49

Trc10Bank.sol - ln.42

ListingManager.sol - ln.50-54

Recommendations

Private modifier does not make a variable invisible.

- 1. Solidity: Visibility and getters

References

- https://tool.smartdec.net/knowledge/SOLIDITY_PRIVATE_MODIFIER

3.6 Overpowered user

Severity	Status	Link	Auditor Comment
High	Open	[Link Here]	We recommend the use of MultiSig wallet for overpowered user.

Description

Functions on **TradingFees.sol** (**setTradingFeeDenominators**, **setTradingFeeDenominatorsForQuote**, **setTradingFeeDenominatorsForPair**) are callable only from one address if the private key of this address becomes compromised trading fees can be changed and may lead to undesirable consequences.

```
function setTradingFeeDenominatorsForPair(...) external
onlyOwner
isValidPrice(_makerFeeDenominator, _takerFeeDenominator) {  ...  }
```

TWM Sources

TradingFees.sol - ln.55-106

Recommendations

We recommend the use of MultiSig wallet for overpowered user.

References

- https://tool.smartdec.net/knowledge/SOLIDITY_OVERPOWERED_ROLE

3.7 Non-initialized return value

Severity	Status	Link	Auditor Comment
Low	Open	[Link Here]	Do not specify returns in functions when there's no need for a return statement.

Description

The functions **_getAllowance** and **_getTrc20BalanceOf** on **Trc20Proxy.sol** doesn't initialize the return value. As result default value will be returned.

```
function _getAllowance(
```

```
 address _account,
 IERC20 _token
) private view
returns (uint _allowance) { ... }
```

TWM Sources

Trc20Proxy.sol - ln.186-224

Trc20Proxy.sol - ln.139-176

Recommendations

Do not specify returns in functions when there's no need for a return statement.

References

- https://tool.smartdec.net/knowledge/SOLIDITY_FUNCTIONS_RETURNS_TYPE_AND_NO_RETURN

3.8 View-function should not change state

Severity	Status	Link	Auditor Comment
Low	Open	[Link Here]	Do not declare functions that change the state as view.

Description

The functions **_getAllowance** and **_getTrc20BalanceOf** on **Trc20Proxy.sol** can change the state as view.

```
let success := staticcall(
 // Gas: forward all gas
 gas,
 // Address: call the token contract
 _token,
 // Start of input
 add(callData, 32),
 // Size of input: selector (4), address (32), address (32)
 68, // same as mload(callData)
 // Start of output: write output over callData
 callData,
 // Size of output: uint
 32
)
```

The following statements are considered modifying the state:

1. Writing to state variables
2. Emitting events
3. Creating other contracts
4. Using **selfdestruct**
5. Sending token via calls
6. Calling any function not marked view or pure
7. Using low-level calls
8. Using inline assembly that contains certain opcodes

TWM Sources

Trc20Proxy.sol - ln.186-224

Trc20Proxy.sol - ln.139-176

Recommendations

This issue is being on Trc20Proxy.sol because In Solidity < 0.4.25, the view modifier was not enforced. This approach ensures calling allowance cannot maliciously modify state or attempt a reentrancy attack.

References

- https://tool.smartdec.net/knowledge/SOLIDITY_SHOULD_NOT_BE_VIEW

3.9 Using assembly

Severity	Status	Link	Auditor Comment
Low	Closed	[Link Here]	Avoid using assembly if possible to not discard safety features of

Description

Inline assembly is a way to access the TVM at a low level. This discards several important safety features of Solidity.

```
function _getAllowance(
 address _account,
 IERC20 _token
) private view
returns (uint _allowance)
{
   ...
 assembly { ... }
}
```

TWM Sources

Trc20Proxy.sol - ln.149
Trc20Proxy.sol - ln.197

Recommendations

Avoid using assembly if possible. On **Trc20Proxy.sol** case seems to be required. Please double check.

References

- https://tool.smartdec.net/knowledge/SOLIDITY_USING_INLINE_ASSEMBLY

4 Test Coverage

Testing is implemented using the TronWatch Market project's own JavaScript-based tooling. It has to be noted that the test coverage is between 85-90% for most contracts.

The 15-10% that is missing is about edge cases and how they can impact / be handled by the contracts.

The “purchaseTradingFeeDiscount” function has a lot of tests supporting the good cases, as we have seen on audit tests a user is able to send tier parameters and they are only handled by safeSub (is not a reentrancy door). We should see a scenario like that on tests to make sure safeSub changes won’t open a door to that exploit.

5 Appendix

5.1 Severity

5.1.1 - Low

Low level issues typically have very little impact on the smart contract. Exploitation of such vulnerabilities may require a lot of energy.

5.1.2 - Mid

Mid level issues do not represent actual bugs or security problems. These issues should be addressed unless there is a clear reason not to.

5.1.3 - High

High level issues are security vulnerabilities that may be very difficult to exploit. Exploitation could result in elevated privileges. These kind of issues are highly likely to cause problems with the operation of the smart contract.

5.1.4 - Critical

Critical level issues are security vulnerabilities. Exploitation of the vulnerability likely results in root-level compromise of smart contract and it’s users.

Attachments