SMART CONTRACT SECURITY AUDIT REPORT

For TRON PHANTA BEAR

26 May 2022

lunaray.co

Table of Contents

1. Overview.....4
2. Background.....5
   2.1 Project Description.....5
   2.2 Audit Range.....6
3. Project contract details.....7
   3.1 Contract Overview.....7
   3.2 Contract details.....8
4. Audit details.....12
   4.1 Findings Summary.....12
   4.2 Risk distribution.....13
   4.3 Risk audit details.....14
      4.3.1. Variables are updated.....14
      4.3.2 Floating Point and Numeric Precision.....14
      4.3.3 Default Visibility.....15
      4.3.4 tx.origin authentication.....15
      4.3.5 Faulty constructor.....16
      4.3.6 Unverified return value.....16
      4.3.7 Insecure random numbers.....17
      4.3.8 Timestamp Dependency.....17
      4.3.9 Transaction order dependency.....18
      4.3.10 Delegatecall.....18
      4.3.11 Call.....19
      4.3.12 Denial of Service.....19
      4.3.13 Logic Design Flaw.....20
      4.3.14 Fake recharge vulnerability.....20
      4.3.15 Short Address Attack Vulnerability.....21
      4.3.16 Uninitialized storage pointer.....21
      4.3.17 Frozen Account bypass.....22
      4.3.18 Uninitialized.....22

4.3.19 Reentry Attack......22
5. Security Audit Tool......23

1. Overview

On May 23, 2022, the security team of Lunaray Technology received the security audit request of the **TRON PHANTA BEAR project**. The team completed the audit of the **TRON PHANTA BEAR smart contract** on May 26, 2022. During the audit process, the security audit experts of Lunaray Technology and the TRON PHANTA BEAR project interface Personnel communicate and maintain symmetry of information, conduct security audits under controllable operational risks, and avoid risks to project generation and operations during the testing process.

Through communicat and feedback with TRON PHANTA BEAR project party, it is confirmed that the loopholes and risks found in the audit process have been repaired or within the acceptable range. The result of this TRON PHANTA BEAR smart contract security audit: **Passed**

Audit Report Hash:

FA96D3E34EE45946B8ABC43300D0B601D902422ED852AEF36BFA6C5FCA35DB15

2. Background

2.1 Project Description

Project name	TRON PHANTA BEAR
Contract type	NFT
Code language	Solidity
Public chain	TRON
Website	https://tronphantabear.com/
Contract file	TRC721Token.sol

2.2 Audit Range

The project smart contract is in the TRON network address：

Name	address
TRC721Token.sol	TACqoNBnAMZmze97KdBibRoQGkS3St875X

3. Project contract details

3.1 Contract Overview

TRC721 Contract

Implementation of TRC721 interface with TRC165, issuance of non-homogenized token standard, including the logic of the base method to ensure proper functioning of NFT.

TRC721Metadata

Users can query the name of the smart contract and details of the asset represented by the NFT.

TRC721Enumerable Contract

Allow a user's smart contract to publish its full list of NFTs and make them visible.

3.2 Contract details

TRC721 Contract

Name	Parameter	Attributes
balanceOf	address owner	public
ownerOf	uint256 tokenId	public
approve	address to, uint256 tokenId	public
getApproved	uint256 tokenId	public
setApprovalForAll	address to, bool approved	public
isApprovedForAll	address owner, address operator	public
transferFrom	address from, address to, uint256 tokenId	public
safeTransferFrom	address from, address to, uint256 tokenId	public
safeTransferFrom	address from, address to, uint256 tokenId, bytes memory _data	public
_safeTransferFrom	address from, address to, uint256 tokenId, bytes memory _data	internal
_exists	uint256 tokenId	internal
_isApprovedOrOwner	address spender, uint256 tokenId	internal
_safeMint	address to, uint256 tokenId	internal
_safeMint	address to, uint256 tokenId, bytes memory _data	internal
_mint	address to, uint256 tokenId	internal
_burn	address owner, uint256 tokenId	internal
_burn	uint256 tokenId	internal
_transferFrom	address from, address to, uint256 tokenId	internal

_checkOnTRC721 Received	address from, address to, uint256 tokenId, bytes memory _data	internal
_clearApproval	uint256 tokenId	private

MinterRole Contract

Name	Parameter	Attributes
isMinter	address account	public
addMinter	address account	onlyMinter
renounceMinter	none	public
_addMinter	address account	internal
_removeMinter	address account	internal

TRC165 Contract

Name	Parameter	Attributes
supportsInterface	bytes4 interfaceId	external
_registerInterface	bytes4 interfaceId	internal

TRC721MetadataMintable Contract

Name	Parameter	Attributes
mintWithTokenURI	address to, uint256 tokenId, string memory tokenURI	onlyMinter

TRC721Enumerable Contract

Name	Parameter	Attributes
tokenOfOwnerByIndex	address owner, uint256 index	public
totalSupply	none	public
tokenByIndex	uint256 index	public
_transferFrom	address from, address to, uint256 tokenId	internal
_mint	address to, uint256 tokenId	internal
_burn	address owner, uint256 tokenId	internal
_tokensOfOwner	address owner	internal
_addTokenToOwnerEnumeration	address to, uint256 tokenId	private
_addTokenToAllTokens Enumeration	uint256 tokenId	private
_removeTokenFromOwner Enumeration	address from, uint256 tokenId	private
_removeTokenFromAllTokens Enumeration	uint256 tokenId	private

TRC721Metadata Contract

Name	Parameter	Attributes
name	none	external
symbol	none	external
tokenURI	uint256 tokenId	external
_setTokenURI	uint256 tokenId, string memory _tokenURI	internal
_setBaseURI	string memory baseURI	internal
baseURI	none	external
_burn	address owner, uint256 tokenId	internal

TRC721Mintable Contract

Name	Parameter	Attributes
mint	address to, uint256 tokenId	onlyMinter
safeMint	address to, uint256 tokenId	onlyMinter
safeMint	address to, uint256 tokenId, bytes memory _data	onlyMinter

4. Audit details

4.1 Findings Summary

Severity	Found	Resolved	Acknowledged
High	0	0	0
Medium	0	0	0
Low	0	0	0
Info	0	0	0

4.2 Risk distribution

Name	Risk level	Repair status
Variables are updated	No	normal
Floating Point and Numeric Precision	No	normal
Default visibility	No	normal
tx.origin authentication	No	normal
Faulty constructor	No	normal
Unverified return value	No	normal
Insecure random numbers	No	normal
Timestamp Dependent	No	normal
Transaction order dependency	No	normal
Delegatecall	No	normal
Call	No	normal
Denial of Service	No	normal
Logical Design Flaw	No	normal
Fake recharge vulnerability	No	normal
Short address attack Vulnerability	No	normal
Uninitialized storage pointer	No	normal
Frozen account bypass	No	normal
Uninitialized	No	normal
Reentry attack	No	normal

4.3 Risk audit details

4.3.1. Variables are updated

- **Risk description**

When there is a contract logic to obtain rewards or transfer funds, the coder mistakenly updates the value of the variable that sends the funds, so that the user can use the value of the variable that is not updated to obtain funds, thus affecting the normal operation of the project.

- **Audit Results : Passed**

4.3.2 Floating Point and Numeric Precision

- **Risk Description**

In Solidity, the floating-point type is not supported, and the fixed-length floating-point type is not fully supported. The result of the division operation will be rounded off, and if there is a decimal number, the part after the decimal point will be discarded and only the integer part will be taken, for example, dividing 5 pass 2 directly will result in 2. If the result of the operation is less than 1 in the token operation, for example, 4.9 tokens will be approximately equal to 4, bringing a certain degree of The tokens are not only the tokens of the same size, but also the tokens of the same size. Due to the economic properties of tokens, the loss of precision is equivalent to the loss of assets, so this is a cumulative problem in tokens that are frequently traded.

- **Audit Results : Passed**

4.3.3 Default Visibility

- **Risk description**

In Solidity, the visibility of contract functions is public pass default. therefore, functions that do not specify any visibility can be called externally pass the user. This can lead to serious vulnerabilities when developers incorrectly ignore visibility specifiers for functions that should be private, or visibility specifiers that can only be called from within the contract itself. One of the first hacks on Parity's multi-signature wallet was the failure to set the visibility of a function, which defaults to public, leading to the theft of a large amount of money.

- **Audit Results : Passed**

4.3.4 tx.origin authentication

- **Risk Description**

tx.origin is a global variable in Solidity that traverses the entire call stack and returns the address of the account that originally sent the call (or transaction). Using this variable for authentication in a smart contract can make the contract vulnerable to phishing-like attacks.

- **Audit Results : Passed**

4.3.5 Faulty constructor

- **Risk description**

Prior to version 0.4.22 in solidity smart contracts, all contracts and constructors had the same name. When writing a contract, if the constructor name and the contract name are not the same, the contract will add a default constructor and the constructor you set up will be treated as a normal function, resulting in your original contract settings not being executed as expected, which can lead to terrible consequences, especially if the constructor is performing a privileged operation.

- **Audit Results : Passed**

4.3.6 Unverified return value

- **Risk description**

Three methods exist in Solidity for sending tokens to an address: transfer(), send(), call.value(). The difference between them is that the transfer function throws an exception throw when sending fails, rolls back the transaction state, and costs 2300gas; the send function returns false when sending fails and costs 2300gas; the call.value method returns false when sending fails and costs all gas to call, which will lead to the risk of reentrant attacks. If the send or call.value method is used in the contract code to send tokens without checking the return value of the method, if an error occurs, the contract will continue to execute the code later, which will lead to the thought result.

- **Audit Results : Passed**

4.3.7 Insecure random numbers

- **Risk Description**

All transactions on the blockchain are deterministic state transition operations with no uncertainty, which ultimately means that there is no source of entropy or randomness within the blockchain ecosystem. Therefore, there is no random number function like rand() in Solidity. Many developers use future block variables such as block hashes, timestamps, block highs and lows or Gas caps to generate random numbers. These quantities are controlled pass the miners who mine them and are therefore not truly random, so using past or present block variables to generate random numbers could lead to a destructive vulnerability.

- **Audit Results : Passed**

4.3.8 Timestamp Dependency

- **Risk description**

In blockchains, data block timestamps (block.timestamp) are used in a variety of applications, such as functions for random numbers, locking funds for a period of time, and conditional statements for various time-related state changes. Miners have the ability to adjust the timestamp as needed, for example block.timestamp or the alias now can be manipulated pass the miner. This can lead to serious vulnerabilities if the wrong block timestamp is used in a smart contract. This may not be necessary if the contract is not particularly concerned with miner manipulation of block timestamps, but care should be taken when developing the contract.

- **Audit Results : Passed**

4.3.9 Transaction order dependency

- **Risk description**

In a blockchain, the miner chooses which transactions from that pool will be included in the block, which is usually determined pass the gasPrice transaction, and the miner will choose the transaction with the highest transaction fee to pack into the block. Since the information about the transactions in the block is publicly available, an attacker can watch the transaction pool for transactions that may contain problematic solutions, modify or revoke the attacker's privileges or change the state of the contract to the attacker's detriment. The attacker can then take data from this transaction and create a higher-level transaction gasPrice and include its transactions in a block before the original, which will preempt the original transaction solution.

- **Audit Results : Passed**

4.3.10 Delegatecall

- **Risk Description**

In Solidity, the delegatecall function is the standard message call method, but the code in the target address runs in the context of the calling contract, i.e., keeping msg.sender and msg.value unchanged. This feature supports implementation libraries, where developers can create reusable code for future contracts. The code in the library itself can be secure and bug-free, but when run in another application's environment, new vulnerabilities may arise, so using the delegatecall function may lead to unexpected code execution.

- **Audit Results : Passed**

4.3.11 Call

- **Risk Description**

The call function is similar to the delegatecall function in that it is an underlying function provided pass Solidity, a smart contract writing language, to interact with external contracts or libraries, but when the call function method is used to handle an external Standard Message Call to a contract, the code runs in the environment of the external contract/function The call function is used to interact with an external contract or library. The use of such functions requires a determination of the security of the call parameters, and caution is recommended. An attacker could easily borrow the identity of the current contract to perform other malicious operations, leading to serious vulnerabilities.

- **Audit Results : Passed**

4.3.12 Denial of Service

- **Risk Description**

Denial of service attacks have a broad category of causes and are designed to keep the user from making the contract work properly for a period of time or permanently in certain situations, including malicious behavior while acting as the recipient of a transaction, artificially increasing the gas required to compute a function causing gas exhaustion (such as controlling the size of variables in a for loop), misuse of access control to access the private component of the contract, in which the Owners with privileges are modified, progress state based on external calls, use of obfuscation and oversight, etc. can lead to denial of service attacks.

- **Audit Results : Passed**

4.3.13 Logic Design Flaw

- **Risk Description**

In smart contracts, developers design special features for their contracts intended to stabilize the market value of tokens or the life of the project and increase the highlight of the project, however, the more complex the system, the more likely it is to have the possibility of errors. It is in these logic and functions that a minor mistake can lead to serious depasstions from the whole logic and expectations, leaving fatal hidden dangers, such as errors in logic judgment, functional implementation and design and so on.

- **Audit Results : Passed**

4.3.14 Fake recharge vulnerability

- **Risk Description**

The success or failure (true or false) status of a token transaction depends on whether an exception is thrown during the execution of the transaction (e.g., using mechanisms such as require/assert/revert/throw). When a user calls the transfer function of a token contract to transfer funds, if the transfer function runs normally without throwing an exception, the transaction will be successful or not, and the status of the transaction will be true. When balances[msg.sender] < _value goes to the else logic and returns false, no exception is thrown, but the transaction acknowledgement is successful, then we believe that a mild if/else judgment is an undisciplined way of coding in sensitive function scenarios like transfer, which will lead to Fake top-up vulnerability in centralized exchanges, centralized wallets, and token contracts.

- **Audit Results : Passed**

4.3.15 Short Address Attack Vulnerability

- **Risk Description**

In Solidity smart contracts, when passing parameters to a smart contract, the parameters are encoded according to the ABI specification. the EVM runs the attacker to send encoded parameters that are shorter than the expected parameter length. For example, when transferring money on an exchange or wallet, you need to send the transfer address address and the transfer amount value. The attacker could send a 19-passte address instead of the standard 20-passte address, in which case the EVM would fill in the 0 at the end of the encoded parameter to make up the expected length, which would result in an overflow of the final transfer amount parameter value, thus changing the original transfer amount.

- **Audit Results : Passed**

4.3.16 Uninitialized storage pointer

- **Risk description**

EVM uses both storage and memory to store variables. Local variables within functions are stored in storage or memory pass default, depending on their type. uninitialized local storage variables could point to other unexpected storage variables in the contract, leading to intentional or unintentional vulnerabilities.

- **Audit Results : Passed**

4.3.17 Frozen Account bypass

- **Risk Description**

In the transfer operation code in the contract, detect the risk that the logical functionality to check the freeze status of the transfer account exists in the contract code and can be passpassed if the transfer account has been frozen.

- **Audit Results : Passed**

4.3.18 Uninitialized

- **Risk description**

The initialize function in the contract can be called pass another attacker before the owner, thus initializing the administrator address.

- **Audit Results : Passed**

4.3.19 Reentry Attack

- **Risk Description**

An attacker constructs a contract containing malicious code at an external address in the Fallback function When the contract sends tokens to this address, it will call the malicious code. The call.value() function in Solidity will consume all the gas he receives when it is used to send tokens, so a re-entry attack will occur when the call to the call.value() function to send tokens occurs before the actual reduction of the sender's account balance. The re-entry vulnerability led to the famous The DAO attack.

- **Audit Results : Passed**

5. Security Audit Tool

Tool name	Tool Features
Oyente	Can be used to detect common bugs in smart contracts
securify	Common types of smart contracts that can be verified
MAIAN	Multiple smart contract vulnerabilities can be found and classified
Lunaray Toolkit	self-developed toolkit

Disclaimer:

Lunaray Technology only issues a report and assumes corresponding responsibilities for the facts that occurred or existed before the issuance of this report, Since the facts that occurred after the issuance of the report cannot determine the security status of the smart contract, it is not responsible for this.

Lunaray Technology conducts security audits on the security audit items in the project agreement, and is not responsible for the project background and other circumstances, The subsequent on-chain deployment and operation methods of the project party are beyond the scope of this audit.

This report only conducts a security audit based on the information provided by the information provider to Lunaray at the time the report is issued, If the information of this project is concealed or the situation reflected is inconsistent with the actual situation, Lunaray Technology shall not be liable for any losses and adverse effects caused thereby.

There are risks in the market, and investment needs to be cautious. This report only conducts security audits and results announcements on smart contract codes, and does not make investment recommendations and basis.

LUNARAY

BLOCKCHAINSECURITY

https://lunaray.co

https://github.com/lunaraySec

https://twitter.com/lunaray_Sec

http://t.me/lunaraySec