Cyberscope

Audit Report

Tectum Enumeration Token

March 2023

SHA256 525f930c7764a8c4dd1af1dc116c221604ca16489a739ab95b8ac57e1b916b06

Audited by © cyberscope

Table of Contents

Review

Audit Updates

Initial Audit	09 Mar 2023

Source Files

Filename	SHA256
TETtoken5c_7-3-23.sol	525f930c7764a8c4dd1af1dc116c221604ca16489a739ab95b8ac57e1b916b06

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	OCTD	Transfers Contract's Tokens	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	ULTW	Transfers Liquidity to Team Wallet	Passed
●	MT	Mints Tokens	Unresolved
●	BT	Burns Tokens	Unresolved
●	BC	Blacklists Addresses	Passed

MT - Mints Tokens

Criticality	Critical
Location	TETtoken5c_7-3-23.sol#L289,311
Status	Unresolved

Description

The contract owner has the authority to mint tokens. The owner may take advantage of it by calling the `TETwrite` or `issue` function. As a result, the contract tokens will be highly inflated.

```
function TETwrite(uint newTS) public onlyOwner returns (uint) {
  require( newTS <= MAX_TET);
  require( newTS != _totalSupply);

  if (newTS > _totalSupply) {
    balances[owner] += ( newTS - _totalSupply );
    _totalSupply =  newTS ;
  } else {
    require ( (balances[owner] - ( _totalSupply - newTS )) >= 0);
    balances[owner] -= ( _totalSupply - newTS );
    _totalSupply =  newTS ;
  }

  return _totalSupply ;
  TETwriteEvent( _totalSupply );
}

function issue(uint amount) public onlyOwner {
  require( _totalSupply + amount <= MAX_TET);
  require(_totalSupply + amount > _totalSupply);
  require(balances[owner] + amount > balances[owner]);

  balances[owner] += amount;
  _totalSupply += amount;
  Issue(amount);
}
```

Recommendation

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

BT - Burns Tokens

Criticality	Critical
Location	TETtoken5c_7-3-23.sol#L289,328
Status	Unresolved

Description

The contract owner has the authority to burn tokens from a specific address. The owner may take advantage of it by calling the `TETwrite` or `redeem` function. As a result, the targeted address will lose the corresponding tokens.

```
function TETwrite(uint newTS) public onlyOwner returns (uint) {
  require( newTS <= MAX_TET);
  require( newTS != _totalSupply);

  if (newTS > _totalSupply) {
    balances[owner] += ( newTS - _totalSupply );
    _totalSupply =  newTS ;
  } else {
    require ( (balances[owner] - ( _totalSupply - newTS )) >= 0);
    balances[owner] -= ( _totalSupply - newTS );
    _totalSupply =  newTS ;
  }

  return _totalSupply ;
  TETwriteEvent( _totalSupply );
}

function redeem(uint amount) public onlyOwner {
  require(_totalSupply >= amount);
  require(balances[owner] >= amount);

  _totalSupply -= amount;
  balances[owner] -= amount;
  Redeem(amount);
}
```

Recommendation

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	CUOO	Calculations Underflow Or Overflow	Unresolved
●	RII	Redundant Inheritance Issue	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L08	Tautology or Contradiction	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L23	ERC20 Interface Misuse	Unresolved

CUOO - Calculations Underflow Or Overflow

Criticality	Critical
Location	TETtoken5c_7-3-23.sol
Status	Unresolved

Description

There are arithmetic calculations that are not properly checked and validated to prevent overflow and underflow errors. Overflow and underflow errors in Solidity code can lead to loss of funds or access to user data.

For instance, if the owner has zero balance `(balances[owner] - ( _totalSupply - newTS ))`, the corresponding calculation will always be greater or equal to zero.

```
balances[owner] += (newTS - _totalSupply);
require((balances[owner] - (_totalSupply - newTS)) >= 0);
balances[owner] -= (_totalSupply - newTS);
require(_totalSupply + amount <= MAX_TET);
require(_totalSupply + amount > _totalSupply);
require(balances[owner] + amount > balances[owner]);
balances[owner] += amount;
_totalSupply -= amount;
balances[owner] -= amount;
```

Recommendation

It is recommended that apply proper checks and validations in the Solidity code to prevent overflow and underflow errors. This may include using safe math libraries, which provide additional checks and validations to prevent overflow and underflow errors in arithmetic operations.

RII - Redundant Inheritance Issue

Criticality	Minor / Informative
Location	TETtoken5c_7-3-23.sol#L188,228
Status	Unresolved

Description

The contract includes the Pausable and BlackList contracts, but it does not make use of their functionalities.

```
contract Pausable is Ownable { ... }
contract BlackList is Ownable, BasicToken {
```

Recommendation

The inheritance of unused contracts in the contract implementation should be avoided. It is recommended to remove unsused contracts.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	TETtoken5c_7-3-23.sol#L53,72,112,123,143,159,177,231,241,246,284,289
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
address _owner
uint public _totalSupply
uint _value
address _to
address _from
address _spender
address _maker
address _evilUser
address _clearedUser

function TETRead() public constant returns (uint) {
            return _totalSupply;
        }

...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.

Find more information on the Solidity documentation https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L08 - Tautology or Contradiction

Criticality	Minor / Informative
Location	TETtoken5c_7-3-23.sol#L298
Status	Unresolved

Description

A tautology is a logical statement that is always true, regardless of the values of its variables. A contradiction is a logical statement that is always false, regardless of the values of its variables.

Using tautologies or contradictions can lead to unintended behavior and can make the code harder to understand and maintain. It is generally considered good practice to avoid tautologies and contradictions in the code.

```
require ( (balances[owner] - ( _totalSupply - newTS )) >= 0)
```

Recommendation

The team is advised to carefully consider the logical conditions is using in the code and ensure that it is well-defined and make sense in the context of the smart contract.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	TETtoken5c_7-3-23.sol#L2
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.4.17;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

L23 - ERC20 Interface Misuse

Criticality	Minor / Informative
Location	TETtoken5c_7-3-23.sol#L75,85,86,112,143,159
Status	Unresolved

Description

The ERC20 is a standard interface for tokens on the blockchain. It defines a set of functions and events that a contract must implement in order to be considered an ERC20 token. According to the ERC20 interface, the transfer function returns a bool value, which indicates the success or failure of the transfer. If the transfer is successful, the function returns true. If the transfer fails, the function returns false. The contract implements the transfer function without the return value.

```
function transfer(address to, uint value) public;
function transferFrom(address from, address to, uint value) public;
function approve(address spender, uint value) public;

function transfer(address _to, uint _value) public onlyPayloadSize(2 *
32) {
        balances[msg.sender] = balances[msg.sender].sub(_value);
        balances[_to] = balances[_to].add(_value);
        Transfer(msg.sender, _to, _value);
    }

...
```

Recommendation

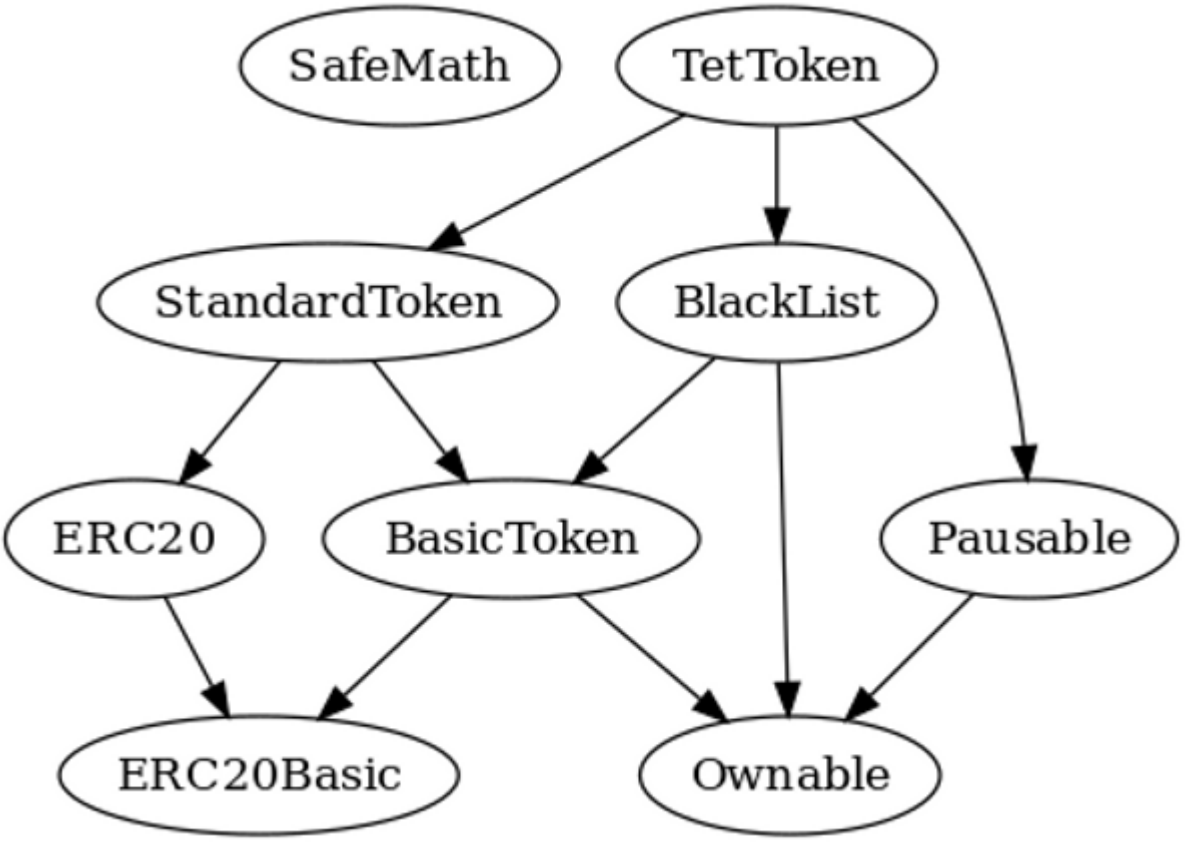
The incorrect implementation of the ERC20 interface could potentially lead to problems when interacting with the contract, as other contracts or applications that expect the ERC20 interface may not behave as expected.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
SafeMath	Library			
	mul	Internal		
	div	Internal		
	sub	Internal		
	add	Internal		
Ownable	Implementation			
		Public	✓	-
	changeOwner	Public	✓	onlyOwner
	confirmOwner	Public	✓	-
ERC20Basic	Implementation			
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
ERC20	Implementation	ERC20Basic		
	allowance	Public		-
	transferFrom	Public	✓	-
	approve	Public	✓	-
BasicToken	Implementation	Ownable, ERC20Basic		

	transfer	Public	✓	onlyPayloadSize
	balanceOf	Public		-
StandardToken	Implementation	BasicToken, ERC20		
	transferFrom	Public	✓	onlyPayloadSize
	approve	Public	✓	onlyPayloadSize
	allowance	Public		-
Pausable	Implementation	Ownable		
	pause	Public	✓	onlyOwner whenNotPaused
	unpause	Public	✓	onlyOwner whenPaused
BlackList	Implementation	Ownable, BasicToken		
	getBlackListStatus	External		-
	getOwner	External		-
	addBlackList	Public	✓	onlyOwner
	removeBlackList	Public	✓	onlyOwner
TetToken	Implementation	Pausable, StandardToken, BlackList		
		Public	✓	-
	totalSupply	Public		-
	TETRead	Public		-
	TETwrite	Public	✓	onlyOwner
	issue	Public	✓	onlyOwner
	redeem	Public	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
TetToken
<Constructor>
totalSupply
TETRead
TETwrite
issue
redeem
BlackList
getBlackListStatus
getOwner
addBlackList
removeBlackList
Pausable
whenNotPaused
whenPaused
pause
unpause
StandardToken
transferFrom
approve
allowance
allowance
sub
BasicToken
onlyPayloadSize
transfer
balanceOf
ERC20
allowance
transferFrom
approve
ERC20Basic
totalSupply
balanceOf
transfer
Ownable
<Constructor>
onlyOwner
changeOwner
confirmOwner
SafeMath (lib)
mul
div
sub
add

Summary

Tectum Enumeration Token contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. There are some functions that can be abused by the owner like mint tokens and burn tokens from any address. if the contract owner abuses the mint functionality, then the contract will be highly inflated. if the contract owner abuses the burn functionality, then the users could lost their tokens. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io