Kambria

Smart Contract Security Audit

Prepared by ShellBoxes

Nov 21st, 2022 - Nov 24th, 2022

Shellboxes.com

contact@shellboxes.com

Document Properties

Client	Kambria
Version	1.0
Classification	Public

Scope

Token	Contract Address
Kambria Token (KAT)	0xeF4656d34BDBF49d30078B5ed856681b45414817

Re-Audit

Token	Contract Address
Kambria Token (KAT)	0x378C907b04f3D41f96d821318B96CFe1c0f9ddAD

Contacts

COMPANY	EMAIL
ShellBoxes	contact@shellboxes.com

Contents

1 Introduction

Kambria engaged ShellBoxes to conduct a security assessment on the Kambria beginning on Nov 21st, 2022 and ending Nov 24th, 2022. In this report, we detail our methodical approach to evaluate potential security issues associated with the implementation of smart contracts, by exposing possible semantic discrepancies between the smart contract code and design document, and by recommending additional ideas to optimize the existing code. Our findings indicate that the current version of smart contracts can still be enhanced further due to the presence of many security and performance concerns.

This document summarizes the findings of our audit.

1.1 About Kambria

Kambria, an open innovation platform for Deep Tech.

Issuer	Kambria
Website	`https://kambria.io`
Type	Solidity Smart Contract
Documentation	KAT BEP20 Smart contract Documentation
Audit Method	Whitebox

1.2 Approach & Methodology

ShellBoxes used a combination of manual and automated security testing to achieve a balance between efficiency, timeliness, practicability, and correctness within the audit's scope. While manual testing is advised for identifying problems in logic, procedure, and implementation, automated testing techniques help to expand the coverage of smart contracts and can quickly detect code that does not comply with security best practices.

1.2.1 Risk Methodology

Vulnerabilities or bugs identified by ShellBoxes are ranked using a risk assessment technique that considers both the LIKELIHOOD and IMPACT of a security incident. This framework is effective at conveying the features and consequences of technological vulnerabilities.

Its quantitative paradigm enables repeatable and precise measurement, while also revealing the underlying susceptibility characteristics that were used to calculate the Risk scores. A risk level will be assigned to each vulnerability on a scale of 5 to 1, with 5 indicating the greatest possibility or impact.

- Likelihood quantifies the probability of a certain vulnerability being discovered and exploited in the untamed.
- Impact quantifies the technical and economic costs of a successful attack.
- Severity indicates the risk's overall criticality.

Probability and impact are classified into three categories: H, M, and L, which correspond to high, medium, and low, respectively. Severity is determined by probability and impact and is categorized into four levels, namely Critical, High, Medium, and Low.

Impact				
	High	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low
		High	Medium	Low

Likelihood

2 Findings Overview

2.1 Summary

The following is a synopsis of our conclusions from our analysis of the Kambria implementation. During the first part of our audit, we examine the smart contract source code and run the codebase via a static code analyzer. The objective here is to find known coding problems statically and then manually check (reject or confirm) issues highlighted by the tool. Additionally, we check business logics, system processes, and DeFi-related components manually to identify potential hazards and/or defects.

2.2 Key Findings

In general, this token contract is well-designed and constructed, but its implementation might be improved by addressing the discovered flaws, which include , 1 medium-severity, 3 low-severity vulnerabilities.

Vulnerabilities	Severity	Status
SHB.1. Power Centralization For The MINTER_ROLE User	MEDIUM	Fixed
SHB.2. Approve Race Condition	LOW	Fixed
SHB.3. The Owner Can Renounce Ownership	LOW	Fixed
SHB.4. Floating Pragma	LOW	Fixed

3 Finding Details

SHB.1 Power Centralization For The MINTER_ROLE User

- Severity : MEDIUM
- Status : Fixed
- Likelihood : 2
- Impact : 2

Description:

The mintTo function allows any MINTER_ROLE user to mint any amount of tokens to any receiver address. This represents a significant centralization where the MINTER_ROLE has too much power in the contract. Having this logic, the minter can increase the totalSupply of the token therefore decreasing its value. The same issue was found in the mint function.

Files Affected:

SHB.1.1: BEP20Token

```
884   function mint(uint256 amount) public onlyRole(MINTER_ROLE) returns (
          ↪ bool) {
885     _mint(_msgSender(), amount);
886     return true;
887   }
888
889   function mintTo(address receiver,uint256 amount) public onlyRole(
          ↪ MINTER_ROLE) returns (bool) {
890     _mint(receiver, amount);
891     return true;
892   }
```

Recommendation:

Consider having a limited maximum supply, or the MINTER_ROLE should be a DAO or a multisig wallet.

Updates

The Kambria team has resolved the issue by limiting the supply to 5 billion (5*1e9).

SHB.1.2: BEP20Token

```
706   function mint(uint256 amount) public onlyRole(MINTER_ROLE) returns (
          ↪ bool) {
707     require(_totalSupply+amount<=5*1e9, "Max total supply is 5
            bilion");
708     _mint(_msgSender(), amount);
709     return true;
710   }
711
712   function mintTo(address receiver,uint256 amount) public onlyRole(
          ↪ MINTER_ROLE) returns (bool) {
713     require(_totalSupply+amount<=5*1e9, "Max total supply is 5
            bilion");
714     _mint(receiver, amount);
715     return true;
716   }
```

SHB.2 Approve Race Condition

- Severity : LOW
- Status : Fixed
- Likelihood : 1
- Impact : 2

Description:

The standard ERC20 implementation contains a widely known race condition in its approve function.

Exploit Scenario:

A spender can witness the token owner broadcast a transaction altering their approval and quickly sign and broadcast a transaction using transferFrom to move the current approved amount from the owner's balance to the spender. If the spender's transaction is validated before the owner's, the spender will be able to get both approval amounts of both transactions.

Files Affected:

SHB.2.1: BEP20Token

```
817 function approve(address spender, uint256 amount) external returns (bool
        ↪ ) {
818 _approve(_msgSender(), spender, amount);
819 return true;
820 }
```

Recommendation:

We recommend using increaseAllowance and decreaseAllowance functions to modify the approval amount instead of using the approve function to do so.

Updates

The Kambria team resolved the issue by disabling the approve function and using the increaseAllowance and the decreaseAllowance functions.

SHB.2.2: BEP20Token

```
639 function approve(address spender, uint256 amount) external {
640  revert("The approve function is disabled, use the increaseAllowance and
        ↪  decreaseAllowance");
```

```
641 }
```

SHB.3 The Owner Can Renounce Ownership

- Severity : LOW
- Status : Fixed
- Likelihood : 1
- Impact : 2

Description:

Typically, the account that deploys the contract is also its owner. Consequently, the owner is able to engage in certain privileged activities in his own name. In smart contracts, the renounceOwnership function is used to renounce ownership, which means that if the contract's ownership has never been transferred, it will never have an owner, rendering some owner-exclusive functionality unavailable.

Files Affected:

SHB.3.1: BEP20Token

```
698 contract BEP20Token is Context, IBEP20, Ownable, AccessControl {
```

Recommendation:

We recommend that you prevent the owner from calling renounceOwnership without first transferring ownership to a different address. Additionally, if you decide to use a multi-signature wallet, then the execution of the renounceOwnership will require at least two or more users to be confirmed. Alternatively, you can disable the Renounce Ownership functionality by overriding it.

Updates

The Kambria team resolved the issue by removing the renounceOwnership function.

SHB.4 Floating Pragma

- Severity : LOW
- Status : Fixed
- Likelihood : 1
- Impact : 1

Description:

The contract makes use of the floating-point pragma 0.8.0. Contracts should be deployed using the same compiler version. Locking the pragma helps ensure that contracts will not be unintentionally deployed using another pragma, which in some cases may be an obsolete version that may introduce issues to the contract system.

Files Affected:

SHB.4.1: BEP20Token

```
5 // SPDX-License-Identifier: MIT
6 pragma solidity ^0.8.13;
```

Recommendation:

Consider locking the pragma version. It is advised that floating pragma should not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Updates

The Kambria team resolved the issue by locking the pragma version to 0.8.13.

4 Best Practices

BP.1 Remove SafeMath Library

Description:

The SafeMath library validates if an arithmetic operation would result in an integer overflow/underflow. If it would, the library throws an exception, effectively reverting the transaction.

Since Solidity 0.8, the overflow/underflow check is implemented.

You don't need the SafeMath library for a solidity compiler version 0.8.13, it's recommended to change all arithmetic operations in the contract :

- x.add(y) to x + y
- x.sub(y) to x - y
- x.mul(y) to x * y
- ...

Files Affected:

BP.1.1: BEP20Token

```
699     using SafeMath for uint256;
```

Status - Fixed

BP.2 Initialize State Token Attributes In The Contract Declaration

Description:

Try initializing the tokens "_name," "_symbol," and "_decimals" directly in the contract declaration as constant variables instead of initializing them in the constructor().

Files Affected:

BP.2.1: BEP20Token

```
711   constructor() {
712     _name = 'Kambria Token';
713     _symbol = 'KAT';
714     _decimals =18;
```

Status - Fixed

BP.3 Set The Admin Role In The Constructor

Description:

To set the ADMIN_ROLE as an admin role of MINTER_ROLE, you don't need to override the getRoleAdmin function;instead, use the admin role setter function setRoleAdmin(MINTER_ROLE,ADMIN_ROLE) in the constructor.

Files Affected:

BP.3.1: BEP20Token

```
737   function getRoleAdmin(bytes32 role) public view virtual override
        ↪ returns (bytes32) {
738     if(role == MINTER_ROLE){
739       return ADMIN_ROLE;
740     }
741     else{
742       return 0;
743     }
744   }
```

Status - Fixed

BP.4 Remove Unnecessary Functions

Description:

Use the grantRole function from AccessControl to assign a role to a specific account, and remove the grantMinterRole and grantAdminRole functions from the BEP20Token contract.

Files Affected:

BP.4.1: BEP20Token

```
722   function grantMinterRole(address account) public onlyRole(ADMIN_ROLE)
        ↪ returns (bool) {
723     _grantRole(MINTER_ROLE, account);
724     return true;
725   }
726  function grantAdminRole( address account) public virtual onlyOwner {
727       _grantRole(ADMIN_ROLE, account);
728   }
```

Status - Fixed

BP.5 Remove Zero Initialization

Files Affected:

BP.5.1: BEP20Token

```
715     _totalSupply = 0;
716     _balances[msg.sender] = _totalSupply;
```

Status - Fixed

5 Tests

Results:

-> Contract: BEP20Token (20 passing)

✓ Should return owner address

✓ Should return ADMIN_ROLE address

✓ Should return MINTER_ROLE address

✓ Should return allowance

✓ Should return decimals

✓ Should return token name

✓ Should return symbol

✓ Should return boolean

✓ Should return totalSupply

✓ Should return Boolean

✓ Should return balanceOf an account

✓ test mint 10000 amount

✓ test mintTo 10000 amount

✓ test decreaseAllowance

✓ test grantRole

✓ test renounceRole

✓ test revokeRole

✓ test increaseAllowance

✓ test transfer

✓ test burn

6 Conclusion

In this audit, we examined the design and implementation of Kambria contract and discovered several issues of varying severity. Kambria team addressed all the issues raised in the initial report and implemented the necessary fixes.

However Shellboxes' auditors advised Kambria Team to maintain a high level of vigilance and participate in bounty programs in order to avoid any future complications.

7 Disclaimer

Shellboxes reports should not be construed as "endorsements" or "disapprovals" of particular teams or projects. These reports do not reflect the economics or value of any "product" or "asset" produced by any team or project that engages Shellboxes to do a security evaluation, nor should they be regarded as such. Shellboxes Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the examined technology, nor do they provide any indication of the technology's proprietors, business model, business or legal compliance. Shellboxes Reports should not be used in any way to decide whether to invest in or take part in a certain project. These reports don't offer any kind of investing advice and shouldn't be used that way. Shellboxes Reports are the result of a thorough auditing process designed to assist our clients in improving the quality of their code while lowering the significant risk posed by blockchain technology. According to Shellboxes, each business and person is in charge of their own due diligence and ongoing security. Shellboxes does not guarantee the security or functionality of the technology we agree to research; instead, our purpose is to assist in limiting the attack vectors and the high degree of variation associated with using new and evolving technologies.

For a Contract Audit, contact us at contact@shellboxes.com