SECURITY AUDIT OF

ARKARUS TOKEN SMART CONTRACTS

Public Report

Dec 24, 2021

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network.
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
Solc	A compiler for Solidity.
ERC20	ERC20 (BEP20 in Binance Smart Chain or *x*RP20 in other chains) tokens are blockchain-based assets that have value and can be sent and received. The primary difference with the primary coin is that instead of running on their own blockchain, ERC20 tokens are issued on a network that supports smart contracts such as Ethereum or Binance Smart Chain.

EXECUTIVE SUMMARY

This Security Audit Report prepared by Verichains Lab on Dec 24, 2021. We would like to thank the Arkarus for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the Arkarus Token Smart Contracts. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified no vulnerable issues in the smart contracts code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Arkarus Token Smart Contracts

Arkarus is a galaxy filled with robots battling for resources that can be exchanged into AKS tokens. AKS tokens will be listed on Binance Smart Chain (BEP20) and every in-game action will be off-chain gas-free so that players do not have to worry about the gas fees...because gas is a valuable resource in every galaxy after all!

Arkarus Token is an ERC20 token that Arkarus players can use in the game.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the smart contracts of Arkarus Token. It was conducted on commit 480131d90ada7e6daa4e335671b16f9abff84cf5 from git repository *https://github.com/daizeta/Arkarus/*.

1.3. Audit methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

- Integer Overflow and Underflow
- Timestamp Dependence
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)
- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

Please note that security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a 100% secure smart contract. However, auditing allows discovering vulnerabilities that were unobserved, overlooked during development and areas where additional security measures are necessary.

2. AUDIT RESULT

2.1. Overview

The initial review was conducted in Dec 2021 and a total effort of 3 working days was dedicated to identifying and documenting security issues in the code base of Arkarus Token.

2.2. Contract codes

The Arkarus Token Smart Contracts was written in Solidity language, with the required version to be 0.8.0.

2.2.1. Arkarus token contract

The Arkarus token is an ERC20 token contract. The Arkarus contract implements all functions that define in IBEP20 interface. The contract also implements mint public function which allows owner contract can mint unlimited tokens. Therefore, the totalSupply value can be changed by this function.s

Table 2 lists some properties of the audited Arkarus Token Smart Contracts (as of the report writing time).

PROPERTY	VALUE
Name	Arkarus
Symbol	AKS
Decimals	18
Total Supply	300,000,000 (x10^{18}) Note: the number of decimals is 18, so the total representation token will be 300,000,000 or 300 million.

Table 2. The Arkarus Token Smart Contracts properties

2.3. Findings

During the audit process, the audit team found no vulnerability in the given version of Arkarus Token Smart Contracts.

2.4. Additional notes and recommendations

2.4.1. Unnecessary usage of SafeMath library in Solidity 0.8.0+ INFORMATIVE

All safe math usage in the contract are for overflow checking, solidity 0.8.0+ already do that by default. We suggest using normal operators for readability and gas saving.

RECOMMENDATION

We suggest changing all methods from SafeMath library to normal arithmetic operator in the contract and remove this library from the source code.

UPDATES

- *Dec 24, 2021*: This issue has been acknowledged and fixed by the Arkarus team. All methods from SafeMath library to normal arithmetic operator with some require statements for checking.

With currently changing, we found that the Arkarus team wants to use require statements before the normal arithmetic operator to get messages when reversion. So the normal arithmetic operator after those require statements can be put into unchecked block to optimize the gas fee.

For an instance in _transfer function:

```
369  function _transfer(address sender, address recipient, uint256 amount…
     ) internal {
370      require(sender != address(0), "BEP20: transfer from the zero add…
     ress");
371      require(recipient != address(0), "BEP20: transfer to the zero ad…
     dress");
372
373      require(amount <= _balances[sender], "BEP20: transfer amount exc…
     eeds balance");
374      _balances[sender] = _balances[sender] - amount;
375      _balances[recipient] = _balances[recipient] + amount;
376      emit Transfer(sender, recipient, amount);
377    }
```

Snippet 1. The `_transfer` can fixing to optimize gas fee

In the _transfer function, the value of_balance[sender] was ensured that it is larger than or equal to the amount. So we can put the code at line 374 into the unchecked block to avoid this statement being checked overflow by default.

The recommended fixing for this:

```
function _transfer(address sender, address recipient, uint256 amount) int…
  ernal {
    require(sender != address(0), "BEP20: transfer from the zero address"…
  );
    require(recipient != address(0), "BEP20: transfer to the zero address…
  ");

    require(amount <= _balances[sender], "BEP20: transfer amount exceeds …
  balance");
    unchecked{
        _balances[sender] = _balances[sender] - amount;
    }
    _balances[recipient] = _balances[recipient] + amount;
    emit Transfer(sender, recipient, amount);
  }
```

Snippet 2. Recommend fixing in `_transfer` function

We can do the same thing in transferFrom, decreaseAllowance, _burn, _burnFrom functions. The unchecked block only affects the statements that are syntactically inside the block, functions called from within an unchecked block do not inherit the property.

APPENDIX

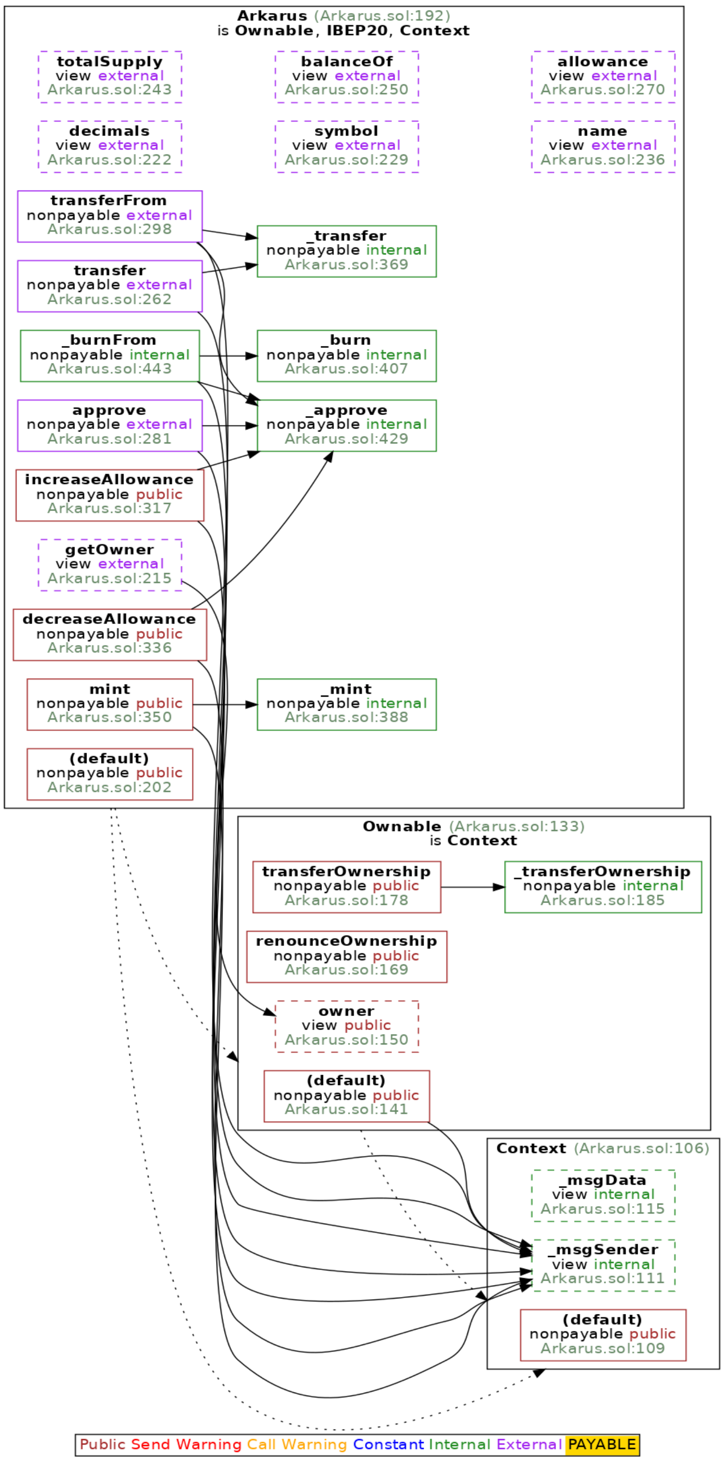
Arkarus (Arkarus.sol:192)
is Ownable, IBEP20, Context
totalSupply view external Arkarus.sol:243
balanceOf view external Arkarus.sol:250
allowance view external Arkarus.sol:270
decimals view external Arkarus.sol:222
symbol view external Arkarus.sol:229
name view external Arkarus.sol:236
transferFrom nonpayable external Arkarus.sol:298
_transfer nonpayable internal Arkarus.sol:369
transfer nonpayable external Arkarus.sol:262
_burnFrom nonpayable internal Arkarus.sol:443
_burn nonpayable internal Arkarus.sol:407
approve nonpayable external Arkarus.sol:281
_approve nonpayable internal Arkarus.sol:429
increaseAllowance nonpayable public Arkarus.sol:317
getOwner view external Arkarus.sol:215
decreaseAllowance nonpayable public Arkarus.sol:336
mint nonpayable public Arkarus.sol:350
_mint nonpayable internal Arkarus.sol:388
(default) nonpayable public Arkarus.sol:202
Ownable (Arkarus.sol:133)
is Context
transferOwnership nonpayable public Arkarus.sol:178
_transferOwnership nonpayable internal Arkarus.sol:185
renounceOwnership nonpayable public Arkarus.sol:169
owner view public Arkarus.sol:150
(default) nonpayable public Arkarus.sol:141
Context (Arkarus.sol:106)
_msgData view internal Arkarus.sol:115
_msgSender view internal Arkarus.sol:111
(default) nonpayable public Arkarus.sol:109
Public Send Warning Call Warning Constant Internal External PAYABLE

Image 1. Arkarus Token Smart Contracts call graph

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Dec 16,2020*	Public Report	Verichains Lab
1.1	*Dec 24,2020*	Public Report	Verichains Lab

Table 3. Report versions history