SECURITY AUDIT OF

CRYPTOPIECE SMART CONTRACT

Public Report

Dec 18, 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 18, 2021. We would like to thank the CryptoPiece 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 CryptoPiece Smart Contract. 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 CryptoPiece Smart Contract

CryptoPiece is a NFT game that focuses on gamers, made by gamers and for gamers. We want to have our players not only feeling relax, enjoying their favorite game, but also earning at the same time. Through our unique play to earn system, players can gather their crew by recruiting mercenary through a Mercenary Contract in Merc Centre. They then can sail their ship and go on an exciting adventure. By capturing criminals in Wanted list by the Government, players can earn Belly – ingame currency – and level up their Merc.

Belly Token is an ERC20 token that Belly 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 Belly Token. It was conducted on commit 0f61bc1d3aa60d82826fa95989c2c37b9ce86f60 from git repository *https://github.com/Cryptopiece/smartcontracts/tree/master/contracts/token*.

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 Belly Token.

2.2. Contract codes

The CryptoPiece Smart Contract was written in Solidity language, with the required version in the range from 0.8.0 to 0.8.9.

2.2.1. Belly token contract

The Belly token is an ERC20 token contract. The contract inherits ERC20Burnable contracts so the contracts have burn function which allows the users toburn their token and cause decreasing the totalSupply of the contract. In addition, the contract also inherits the burnFrom function. So an allowance account can call this function to remove the owner balances.

Table 2 lists some properties of the audited CryptoPiece Smart Contract (as of the report writing time).

PROPERTY	VALUE
Name	Belly
Symbol	Belly
Decimals	18
Total Supply	1,000,000,000 (x10^{18}) Note: the number of decimals is 18, so the total representation token will be 1,000,000,000 or 1 billion.

Table 2. The CryptoPiece Smart Contract properties

2.3. Findings

During the audit process, the audit team found no vulnerability in the given version of CryptoPiece Smart Contract.

2.4. Additional notes and recommendations

2.4.1. Useless code in contructor INFORMATIVE

In the constructor, the transferOwnership statement for msg.sende is useless. Because the contract inherits Ownable which has done it in the Ownable constructor.

```
    constructor() {
        _mint(msg.sender,1000000000*10**uint256(18));
        transferOwnership(msg.sender);
    }
```

RECOMMENDATION

We suggest removing the transferOwnership statement.

UPDATES

- *Dec 18,2021*: This issue has been acknowledged and fixed by the CryptoPiece team int commit e6db4646d305b98ed665e21165602e0c681b6572. The statements was removed.

2.4.2. Useless _beforeTokenTransfer internal function INFORMATIVE

The contract overrides _beforeTokenTransfer internal function but it doesn't do anything except call the _beforeTokenTransfer parent function. It is useless.

```
    function _beforeTokenTransfer(address from, address to, uint256 amount)
internal override(ERC20) {
        super._beforeTokenTransfer(from, to, amount);
    }
```

RECOMMENDATION

We suggest removing this function for readability.

UPDATES

- *Dec 18,2021*: This issue has been acknowledged and fixed by the CryptoPiece team int commit e6db4646d305b98ed665e21165602e0c681b6572. The function was removed.

2.4.3. The range of solidity version too wide and old INFORMATIVE

In the head of the source code, the file defines the version solidity that was used in the contract between 0.7.5 to 0.8.9. We suggest changing the lower bound to 0.8.0.

After setting the lower bound of the solidity version to 0.8.0. We suggest removing the SafeMath library from the contract and changing all methods of SafeMath to normal operators. Because all SafeMath usage in the contract is for overflow checking, solidity 0.8.0+ already do that by default, the only usage of SafeMath now is to have a custom revert message which isn't the case in the auditing contracts.

UPDATES

- *Dec 18,2021*: This issue has been acknowledged and fixed by the CryptoPiece team int commit e6db4646d305b98ed665e21165602e0c681b6572.

2.4.4. Redundant importing contracts INFORMATIVE

In the head of the source code, the source code imported ERC20Capped, SafeERC20 and ERC20 contract but they aren't used anywhere. We suggest removing these import statements for readability.

UPDATES

- *Dec 18,2021*: This issue has been acknowledged by the CryptoPiece team int commit e6db4646d305b98ed665e21165602e0c681b6572.

APPENDIX

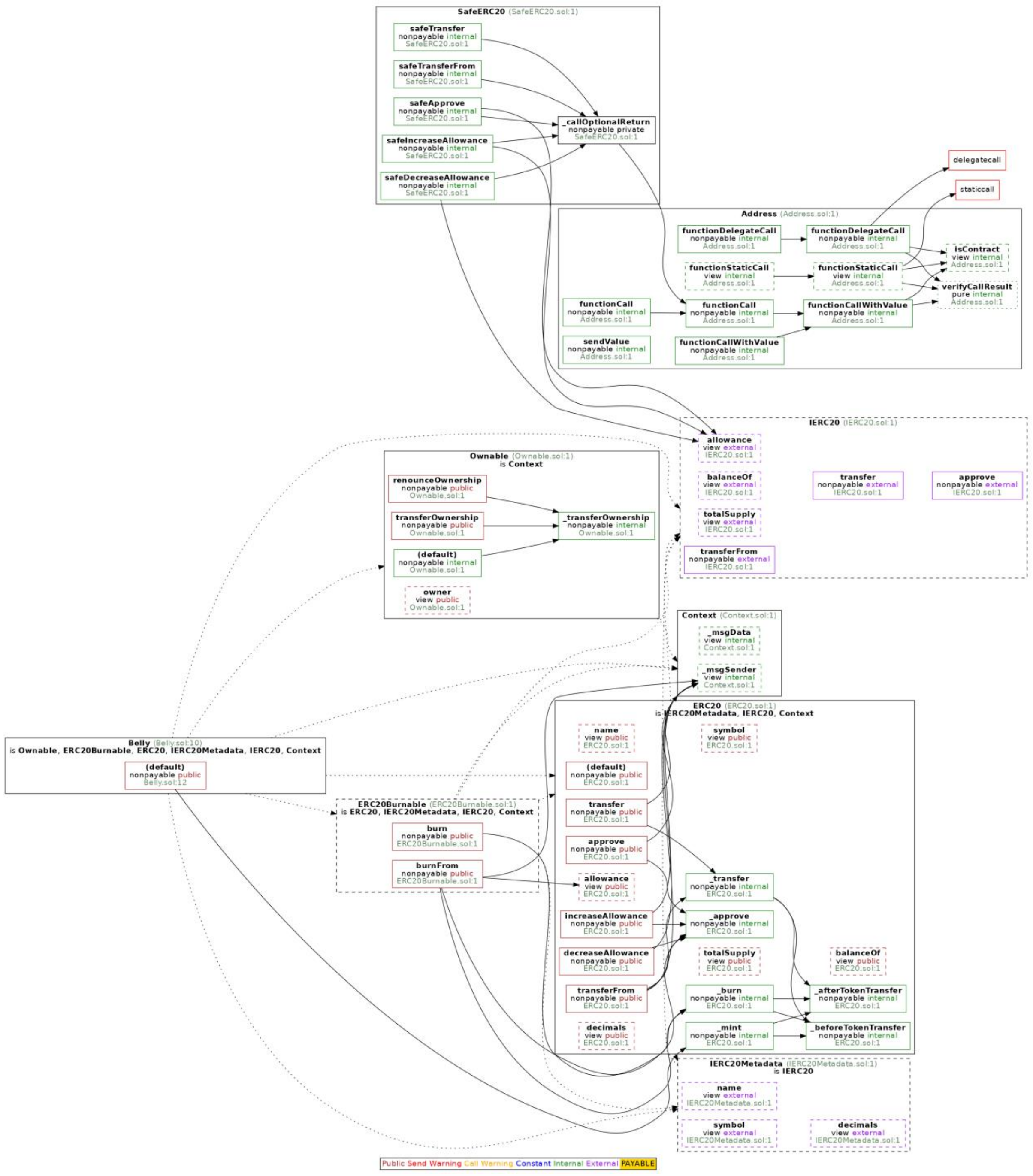
SafeERC20 (SafeERC20.sol:1)
safeTransfer nonpayable internal SafeERC20.sol:1
safeTransferFrom nonpayable internal SafeERC20.sol:1
safeApprove nonpayable internal SafeERC20.sol:1
safeIncreaseAllowance nonpayable internal SafeERC20.sol:1
safeDecreaseAllowance nonpayable internal SafeERC20.sol:1
_callOptionalReturn nonpayable private SafeERC20.sol:1
delegatecall
staticcall
Address (Address.sol:1)
functionDelegateCall nonpayable internal Address.sol:1
functionStaticCall view internal Address.sol:1
isContract view internal Address.sol:1
verifyCallResult pure internal Address.sol:1
functionCall nonpayable internal Address.sol:1
functionCallWithValue nonpayable internal Address.sol:1
sendValue nonpayable internal Address.sol:1
IERC20 (IERC20.sol:1)
allowance view external IERC20.sol:1
balanceOf view external IERC20.sol:1
transfer nonpayable external IERC20.sol:1
approve nonpayable external IERC20.sol:1
totalSupply view external IERC20.sol:1
transferFrom nonpayable external IERC20.sol:1
Ownable (Ownable.sol:1) is Context
renounceOwnership nonpayable public Ownable.sol:1
transferOwnership nonpayable public Ownable.sol:1
(default) nonpayable internal Ownable.sol:1
_transferOwnership nonpayable internal Ownable.sol:1
owner view public Ownable.sol:1
Context (Context.sol:1)
_msgData view internal Context.sol:1
_msgSender view internal Context.sol:1
ERC20 (ERC20.sol:1) is IERC20Metadata, IERC20, Context
name view public ERC20.sol:1
symbol view public ERC20.sol:1
(default) nonpayable public ERC20.sol:1
transfer nonpayable public ERC20.sol:1
approve nonpayable public ERC20.sol:1
allowance view public ERC20.sol:1
increaseAllowance nonpayable public ERC20.sol:1
decreaseAllowance nonpayable public ERC20.sol:1
transferFrom nonpayable public ERC20.sol:1
decimals view public ERC20.sol:1
_transfer nonpayable internal ERC20.sol:1
_approve nonpayable internal ERC20.sol:1
totalSupply view public ERC20.sol:1
balanceOf view public ERC20.sol:1
_burn nonpayable internal ERC20.sol:1
_mint nonpayable internal ERC20.sol:1
_afterTokenTransfer nonpayable internal ERC20.sol:1
_beforeTokenTransfer nonpayable internal ERC20.sol:1
Belly (Belly.sol:10) is Ownable, ERC20Burnable, ERC20, IERC20Metadata, IERC20, Context
(default) nonpayable public Belly.sol:12
ERC20Burnable (ERC20Burnable.sol:1) is ERC20, IERC20Metadata, IERC20, Context
burn nonpayable public ERC20Burnable.sol:1
burnFrom nonpayable public ERC20Burnable.sol:1
IERC20Metadata (IERC20Metadata.sol:1) is IERC20
name view external IERC20Metadata.sol:1
symbol view external IERC20Metadata.sol:1
decimals view external IERC20Metadata.sol:1
Public Send Warning Call Warning Constant Internal External PAYABLE

Image 1. CryptoPiece Smart Contract call graph

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Dec 18,2020*	Public Report	Verichains Lab

Table 3. Report versions history