verichains

SECURITY AUDIT OF

FOOTBALL BATTLE SMART CONTRACTS

Public Report

Jun 24, 2022

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 Jun 24, 2022. We would like to thank the Football Battle 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 Football Battle 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 some vulnerable issues in the smart contracts code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Football Battle Smart Contracts

Football Battle, as known as Soccer Battle (US), is a play-to-earn game with innovative Gameplay combining both Strategy and Action gameplay.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of Football Battle Smart Contracts.

There are 2 files in our audit scope. They are FBLToken.sol and FBattleNFT.sol files.

The last version of the FBLToken contract is deployed on Binance Smart Chain Mainnet at address 0x393C44A497706DDE15996BB0C7Bdf691A79De38a.

The details of the deployed smart contract are listed in Table 1.

FIELD	VALUE
Contract Name	FBL
Contract Address	0x393C44A497706DDE15996BB0C7Bdf691A79De38a
Compiler Version	v0.8.1+commit.df193b15
Optimization Enabled	Yes with 200 runs
Explorer	*https://bscscan.com/address/0x393C44A497706DDE15996BB0C7Bdf691A79De38a*

Table 1. The deployed smart contract details

The last version of the FBattleNFT contract is deployed on Binance Smart Chain Mainnet at address 0xf1f1dcb647558aee4e6a7001b56f3ec71c824ea1.

The details of the deployed smart contract are listed in Table 2.

FIELD	VALUE
Contract Name	FBPlayer721

FIELD	VALUE
Contract Address	0xf1f1dcb647558aee4e6a7001b56f3ec71c824ea1
Compiler Version	v0.8.1+commit.df193b15
Optimization Enabled	Yes with 200 runs
Explorer	*https://bscscan.com/address/0xf1f1dcb647558aee4e6a7001b56f3ec71c824ea1*

Table 2. The deployed smart contract details

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 3. 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 Football Battle Smart Contracts was written in Solidity language, with the required version to be ^0.8.1. The source code was written based on OpenZeppelin's library.

2.1.1. FBLToken contract

FBLToken contract only extends the ERC20 contract. The tokens were minted following the budgets which were set by the deployer in the constructor. With the data following the budgets, the budget receiver or the token owner may call the mint public function to create new tokens.

This table lists some properties of the audited FBLToken contract (as of the report writing time).

PROPERTY	VALUE
Name	FootballBattle
Symbol	FBL
Decimals	18

Table 4. The FBLToken properties

2.1.2. FBattleNFT contract

FBattleNFT contract only extends ERC721 contract. The contract implements the logic following the projects. For each NFT minted in a project, the contract will keep a tiny amount of the profit like the fee. The creators may call withdraw public function to receive profit from their projects.

The contract also implements the owGetCrypto public function which allows the owner of the contract to withdraw all tokens in the contract.

This table lists some properties of the audited FBattleNFT contract (as of the report writing time).

PROPERTY	VALUE
Name	FootballBattle Player

PROPERTY	VALUE
Symbol	FBP

Table 5. The FBattleNFT properties

2.2. Findings

During the audit process, the audit team found some vulnerability issues in the given version of Football Battle Smart Contracts.

2.2.1. FBattleNFT.sol - Unsafe using transfer and transferFrom method through IERC20 interface MEDIUM

There are some functions in the contract that use transfer, transferFrom methods to call functions from the token contract. The contract doesn't point out exactly which token contract is used. Therefore, we can't ensure that the transfer and transferFrom function of another token contract works exactly as expected.

For instance, the transfer function can return false with the function call failure instead of returning true or revert like ERC20 Oppenzepplin. With mintSerie logic, the contract doesn't receive anything while the projects[pId_].uIncome value is changed.

```
function mintSerie(uint pId_, uint256 number_, uint256 amount_) external …
  payable {
        require( amount_ > 0, "invalid amount");
        require( number_ > 0, "invalid receivers");
        require( number_ + projects[pId_].uCurrent <= projects[pId_].limi…
  t, "invalid token number");
        require( amount_  == projects[pId_].price * number_,  "Amount sen…
  t is not correct");
        _cryptoTransferFrom(msg.sender, address(this), projects[pId_].cry…
  pto, amount_);
        uint256 vCurrent = projects[pId_].uCurrent;

        for(uint256 vI = 0; vI < number_; vI++) {
            Info memory vInfo;
            vInfo.proId     =   pId_;
            vInfo.proIndex  =   vCurrent + vI;
            infos.push(vInfo);
            _addTokenToOwnerEnumeration(msg.sender, tokenIdCurrent);
            _mint(msg.sender, tokenIdCurrent);
            tokenIdCurrent++;
```

```
        }
        projects[pId_].uCurrent  += number_;
        projects[pId_].uIncome   += amount_;

        emit MintProject(pId_, number_, tokenIdCurrent-1, msg.sender);
    }

    function _cryptoTransfer(address to_,  address crypto_, uint256 amoun…
  t_) internal returns (uint256) {
        if(amount_ == 0) return 0;
        if(crypto_ == address(0)) {
            payable(to_).transfer( amount_);
            return 1;
        }
        IERC20(crypto_).transfer(to_, amount_);
        return 2;
    }
```

Snippet 1. Unsafe using `transfer` method in `withdraw` function in FBattleNFT contract

The transferFrom method used in the _cryptoTransferFrom function has the same problem.

RECOMMENDATION

We suggest using SafeERC20 library for IERC20 and changing all transfer, transferFrom methods used in the contract to safeTransfer, safeTransferFrom which are declared in SafeERC20 library to ensure that there is no issue when transferring tokens.

UPDATES

- *May 25, 2022*: This issue has been acknowledged by the Football Battle.

2.2.2. FBattleNFT.sol - The opMintProject misses adding the new tokens to the owner enumeration LOW

The opMintProject function allows the operator to mint new tokens but the function doesn't set the owner enumeration data of these tokens. If one of these tokens is the last token of the user, when the user calls transfer/burn any tokens, the infos of the tokenID=0 may be overwritten although the user doesn't own the tokenID=0.

```
function opMintProject(uint pId_, address to_, uint number_) external pay…
  able chkOperator {
        require( number_ > 0, "invalid receivers");
        require( number_ + projects[pId_].uCurrent <= projects[pId_].limi…
```

```
  t, "invalid token number");
        uint256 vCurrent = projects[pId_].uCurrent;
        for(uint256 vI = 0; vI < number_; vI++) {
            Info memory vInfo;
            vInfo.proId     =   pId_;
            vInfo.proIndex  =   vCurrent + vI;
            infos.push(vInfo);
            //Missing here
            _mint(to_, tokenIdCurrent);
            tokenIdCurrent++;
        }
        projects[pId_].uCurrent  += number_;

        emit MintProject(pId_, number_, tokenIdCurrent-1, to_);
    }
    function _removeTokenFromOwnerEnumeration(address from, uint256 token…
  Id) private {
        uint256 lastTokenIndex = balanceOf(from)-1;
        uint256 ownedposition = infos[tokenId].ownedPosition;
        infos[_ownedTokens[from][lastTokenIndex]].ownedPosition         = …
     ownedposition;  //The _ownedTokens[from][lastTokenIndex] value may be …
  0, if it isn't set in the opMintProject function
        _ownedTokens[from][ownedposition]   =  _ownedTokens[from][lastTok…
  enIndex];
    }
```

RECOMMENDATION

The opMintProject function should be changed like the below code:

```
function opMintProject(uint pId_, address to_, uint number_) external pay…
  able chkOperator {
        require( number_ > 0, "invalid receivers");
        require( number_ + projects[pId_].uCurrent <= projects[pId_].limi…
  t, "invalid token number");
        uint256 vCurrent = projects[pId_].uCurrent;
        for(uint256 vI = 0; vI < number_; vI++) {
            Info memory vInfo;
            vInfo.proId     =   pId_;
            vInfo.proIndex  =   vCurrent + vI;
            infos.push(vInfo);
            _addTokenToOwnerEnumeration(to, tokenIdCurrent);
            _mint(to_, tokenIdCurrent);
```


```
            tokenIdCurrent++;
        }
        projects[pId_].uCurrent  += number_;

        emit MintProject(pId_, number_, tokenIdCurrent-1, to_);
    }
```

UPDATES

- *Jun 23, 2022*: This issue has been acknowledged by the Football Battle.

APPENDIX

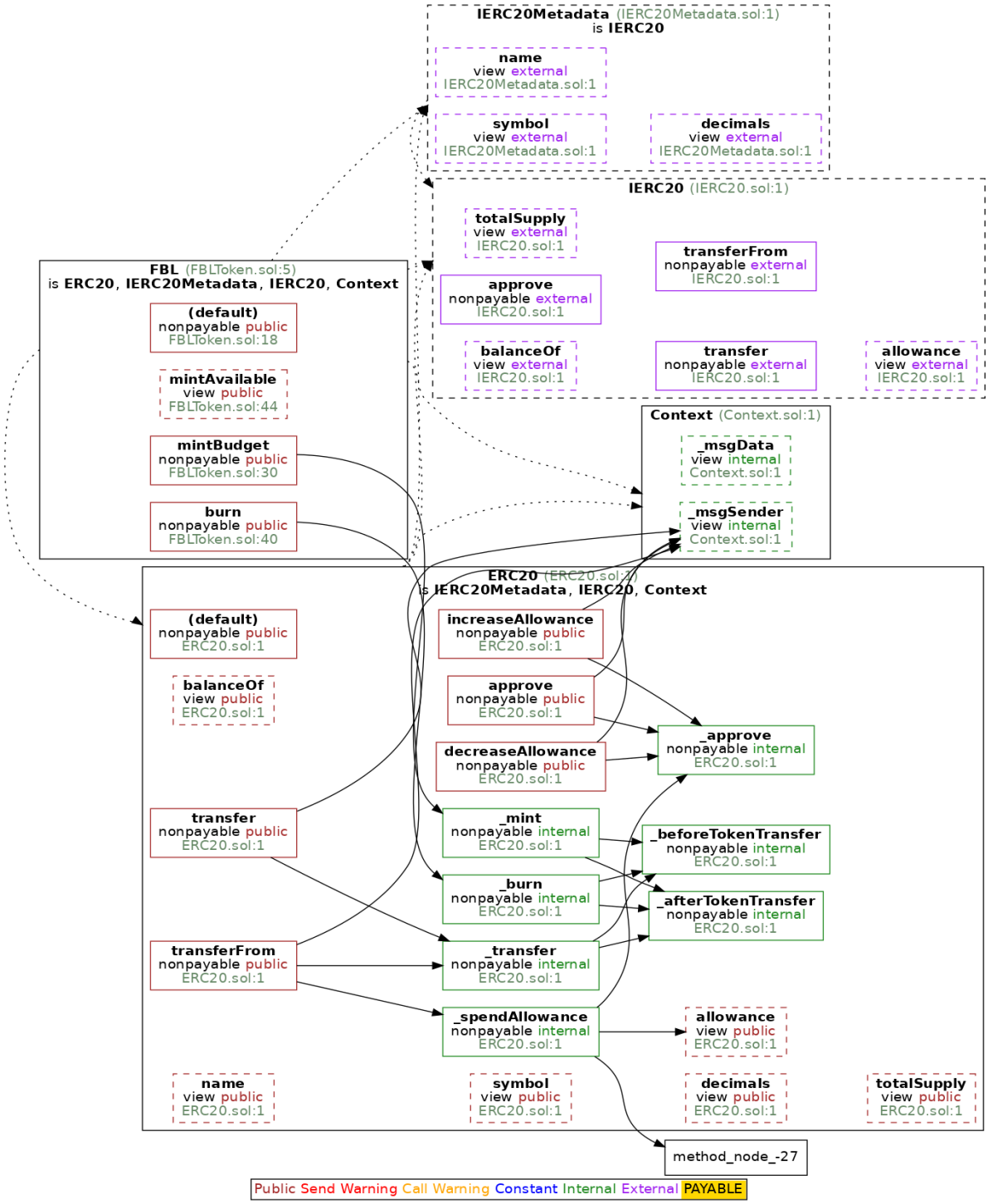

Image 1. FBLToken smart contract call graph

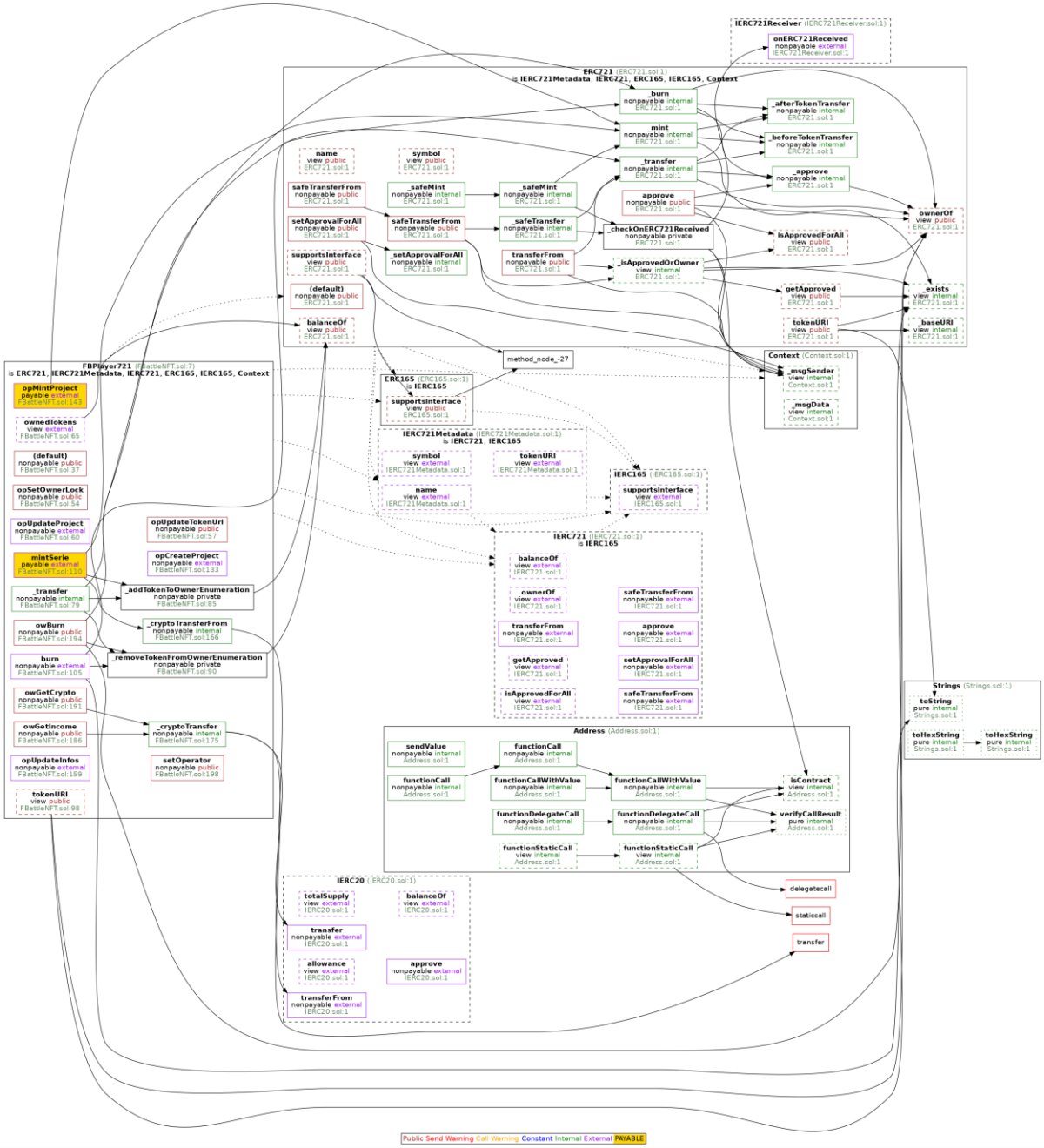

Image 2. FBattleNFT smart contract call graph

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Jun 24, 2022*	Public Report	Verichains Lab

Table 6. Report versions history