verichains

SECURITY AUDIT OF

FOOTBALL BATTLE SMART CONTRACTS

Public Report

May 30, 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 May 30, 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. It was conducted on commit 9125d65166480edf1f6999d3797261251d7e3bf8 from git repository *https://github.com/klabs-hoa/crowdhero_football_battle_chain/tree/main/contracts*.

The latest version of the following files were made available in the course of the review:

SHA256 Sum	File
431d3f8d418c41506d1eedef5a155af73717b261ffb9bacd8e821b32266f9e8d	**FBLToken.sol**
97fee729b311645ad454bb28917e04210336691e6220e74c804bfdd3ef307768	**FBattleNFT.sol**

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 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, anyone may call the mint public function to create new tokens.

This table lists some properties of the audited Football Battle Smart Contracts (as of the report writing time).

PROPERTY	VALUE
Name	FootballBattle
Symbol	FBL
Decimals	18

Table 2. The Football Battle Smart Contracts 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.

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 - Users may burn any NFTs CRITICAL

The contract implements the burn public function without a modifier. Therefore, any user may call this function to destroy any nfts.

```
function burn( uint256 id) external {
        _burn(id);
    }
```

Snippet 1. The `burn` public function in FBattleNFT contract

RECOMMENDATION

The Football Battle team should add the modifier to the function which supports verifying the caller.

UPDATES

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

2.2.2. FBLToken.sol - Users may burn tokens of anyone CRITICAL

The contract implements the burn public function without a modifier. Therefore, any user may call this function to remove tokens from anyone.

```
function burn(address account, uint256 amount) public {
    _burn(account, amount);
}
```

Snippet 2. The `burn` public function in FBLToken contract

RECOMMENDATION

The Football Battle team should add the modifier to the function which supports verifying the caller.

UPDATES

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

2.2.3. FBattleNFT.sol - Users may call testSetOperator function to create new operator HIGH

The Operator role in the contract is an important role which has plenty of permission to update the state of the contract. But the contract implements the testSetOperator public function which allows anyone to approve/unapprove operator role.

The testSetOperator public function may be a convenient feature on the testnet, but it may be a terrible issue if the contract includes it on the mainnet.

```
/** for test */
    function testSetOperator(address opr_, bool val_) public {
    _operators[opr_] = val_;
}
```

Snippet 3. The `testSetOperator` public function in FBattleNFT contract

UPDATES

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

2.2.4. 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 withdraw logic, the creator doesn't receive anything while the projects[pId_].uIncome value is changed.

```
function withdraw(uint pId_) external {
        require(projects[pId_].creator == msg.sender, "only for creator");
        uint256 vAmount                     = projects[pId_].uIncome;
        projects[pId_].uIncome          = 0;
        _cryptoTransfer(msg.sender, projects[pId_].crypto, vAmount);
    }

    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 4. 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.

2.2.5. FBattleNFT.sol - Emit wrong event with amount = 0 in opMintProject function LOW

The opMintProject function is used for minting NFTs for a bunch of users and creating the event. There are some require statements that verify the input data. But they miss the case that the amount_ value equals 0.

If the amount_ equals 0 and tos_ is an empty array, the transaction may still expose the MintProject event with the last tokenID.

If the Football Battle team uses the data which was exposed from the MintProject event, it may cause some problems.

```
function opMintProject(uint pId_, address[] memory tos_, uint256 index_, …
  uint256 amount_) external payable chkOperator {
        require( tos_.length <= projects[pId_].uLimit, "invalid token num…
  ber");
        require( amount_  == projects[pId_].price * tos_.length,  "Amount…
   sent is not correct");
        _cryptoTransferFrom(msg.sender, address(this), projects[pId_].cry…
  pto, amount_);

        for(uint256 vI = 0; vI < tos_.length; vI++) {
            _mint(tos_[vI], tokenIdCurrent);
            tokenIdCurrent++;
            Info memory vInfo;
            vInfo.proId     =   pId_;
            vInfo.index     =   index_ + vI;
            infos.push(vInfo);
        }
        projects[pId_].uLimit      -= tos_.length;
        projects[pId_].uIdCurrent  += tos_.length;
        if(amount_ > 0) {
            uint256 vFee           =  projects[pId_].fee * tos_.length;
            projects[pId_].uTax    += vFee;
```

```
            projects[pId_].uIncome += amount_ - vFee;
        }
        emit MintProject(pId_, index_, tokenIdCurrent-1, tos_);
    }

    function _cryptoTransferFrom(address from_, address to_, address cryp…
  to_, uint256 amount_) internal returns (uint256) {
        if(amount_ == 0) return 0;
        if(crypto_ == address(0)) {
            require( msg.value == amount_, "ivd amount");
            return 1;
        }
        IERC20(crypto_).transferFrom(from_, to_, amount_);
        return 2;
    }
```

RECOMMENDATION

The Football Battle team should add a require statement that verifies the amount_ value.

UPDATES

- *May 30, 2022*: This issue has been acknowledged and fixed by the Football Battle.

2.2.6. FBattleNFT.sol - The ownerTokens function call may be failed if the tokenIdCurrent value is a big number INFORMATIVE

The ownerToken function uses a for-loop statement that fetches through all tokens minted. If the number of tokens minted is a big number, the transaction using it may run out of gas and failed.

```
function ownerTokens(address own_) external view returns(uint[][] memory)…
   {
        require(balanceOf(own_) > 0, "none NFT");
        uint[][] memory vTkns = new uint[][](balanceOf(own_));
        uint vTo;
        for(uint256 vI = 0; vI <= tokenIdCurrent; vI++) {
           if(ownerOf(vI) == own_) {
                vTkns[vTo]      = new uint[](3);
                vTkns[vTo][0]   = vI;
                vTkns[vTo][1]   = infos[vI].proId;
                vTkns[vTo][2]   = projects[infos[vI].proId].fundId;
                vTo++;
```

```
            }
            if(vTo == balanceOf(own_)) break;
        }
        return vTkns;
    }
```

RECOMMENDATION

The Football Battle should consider using the ERC721Enumerable contract which was designed by OpenZeppelin.

UPDATES

- *May 30, 2022*: This issue has been acknowledged and fixed by the Football Battle.

APPENDIX

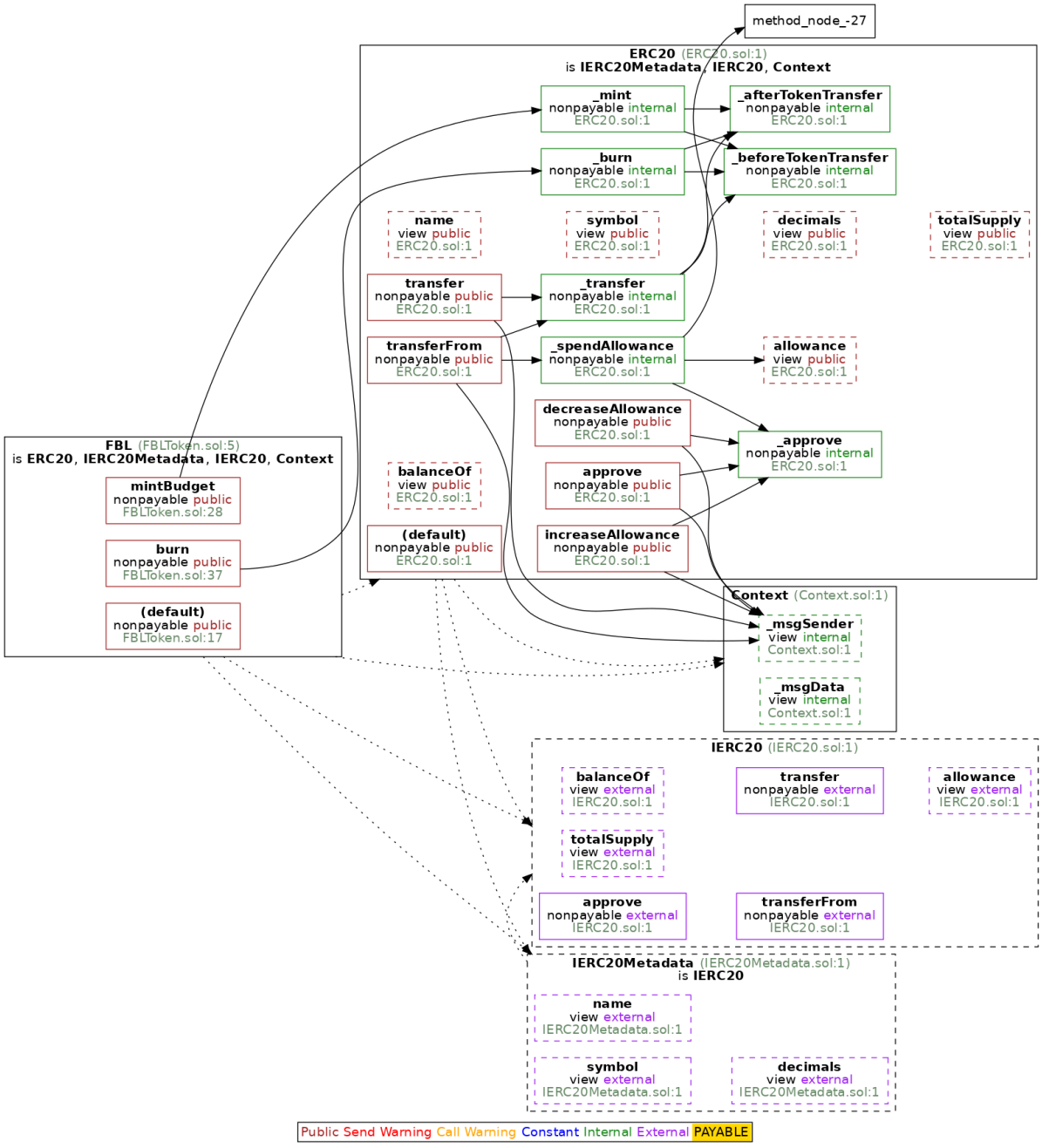

Image 1. FBLToken smart contract call graph

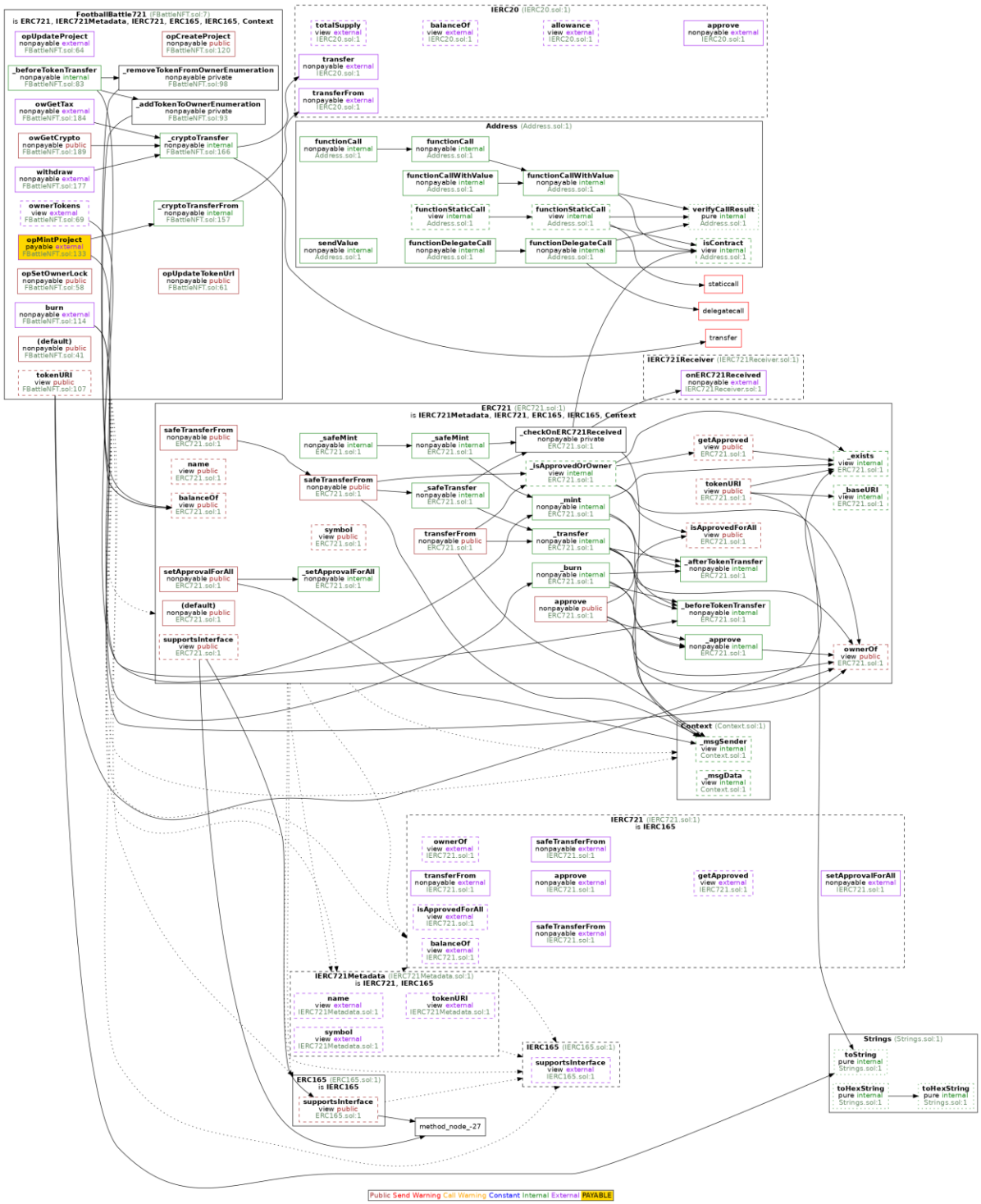

Image 2. FBattleNFT smart contract call graph

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*May 24, 2022*	Private Report	Verichains Lab
1.1	*May 25, 2022*	Private Report	Verichains Lab
1.2	*May 30, 2022*	Public Report	Verichains Lab

Table 3. Report versions history