SECURITY AUDIT OF

POLYFLIP SMART CONTRACTS

Public Report

Mar 28, 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 Mar 28, 2022. We would like to thank the Polyflip 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 Polyflip 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 contract code, along with some recommendations. Polyflip team has resolved and updated most of the issues following our recommendations.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Polyflip Smart Contracts

Polyflip is a decentralized betting platform powered by Polygon Network and Chainlink. Unlike traditional casinos that operate in black boxes, Polyflip runs on smart contracts that are fair, transparent and immutable. The platform provides a low-cost and fast gaming experience through the combination of both traditional core game and blockchain mechanics.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the Polyflip Smart Contracts. It was conducted on the source code provided by the Polyflip team.

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

SHA256 Sum	File
50f22c5bbc909544e7882ccbe734a071fc1c4aca88690ee46a199fad5c9dfb0b	**./House/House.sol**
3d48cdc1244dece01c44d0edecfc15a50166ce7f2b2a0ef4075b148510a2ea8e	**./Games/Coinflip/Coinflip.sol**
2471602781f82993df87c12a114424b2dd865f026e7e5318d954c909964f8e23	**./Games/Coinflip/Manager.sol**
33cc0a403bec5e46a9954cb9244e74464b27edf2b0e48a9622770cba856b42e0	**./Interfaces/IGame.sol**
b74a41df628674179fdcdfbca2b25e793de8a006063f5e6a974ba731880a926d	**./Interfaces/IVRFManager.sol**
d9685661545af399e0ec35e95bef86592a1116a3f874e24c3f1fcb08ceba73f4	**./Interfaces/IHouse.sol**
a1849fe35693ff80d376bc2b89232be9a84f276786eaafd3b888f675d7a8de96	**./VRFManager/VRFManager.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 Polyflip Smart Contracts was written in Solidity language, with the required version to be 0.8.11. The source code was written based on OpenZeppelin's library.

These contracts are used to build a betting platform that supports multiple coin-flip games at the same time. In each game, the user will have to predict the outcome of the corresponding coins in exact order. A small fee will be taken from users' betting amount (2% by default), part of that fee amount (75% by default) will be distributed to all Polyflip NFT holders as a special reward. Note that these ratios can be changed by the contract owner at any time.

These games use random numbers generated by ChainLink VRF V1 so that the randomness result can not be manipulated by the contract owner or anyone else.

2.2. Findings

During the audit process, the audit team found some vulnerabilities in the given version of Polyflip Smart Contracts. Polyflip team has fixed the code according to Verichain's draft report. This section contains a detailed analysis of all the vulnerabilities that were discovered by the audit team during the audit process.

2.2.1. VRFManager.sol - Front-running when settle bets CRITICAL

When the fulfillRandomness transaction is submitted by ChainLink, the attacker who is listening for pending transactions in mempool, can read the randomness result and submit a new placeBet transaction with a crafted betChoice (calculated based on the value of randomness) and a higher gas price to be executed before the fulfillRandomness transaction. Since the Coinflip contract allows users to place bets when the randomness request was made, and the randomness result is used to settle for all pending bets (including new bets), so this issue can be exploited to place a bet and win immediately.

```
// VRFManager.sol
function fulfillRandomness(bytes32, uint randomness) internal override {
    uint256[] memory expandedValues = expand(randomness, maxBetsPerGame);

    for (uint i = 0; i < games.length; i++) {
        if (gasleft() <= 150000) {
            return;
        }
        if (address(games[i]) != address(0)) {
            games[i].settleBet(expandedValues);
        }
    }
}

// Coinflip.sol
function settleBet(uint256[] memory expandedValues) external isVRFManager…
   {
    uint _pedingBetsLenght = pendingBetsLength();
    if (_pedingBetsLenght == 0) {
        return;
    }

    uint i;
    for (i = 0; i < _pedingBetsLenght && i < expandedValues.length; i++) …
  {
        if (gasleft() <= 100000) {
            if (i == 0) {
                return;
            }
            break;
        }
        _settleBet(pendingBets[i], expandedValues[i]);
    }

    uint[] memory newArray = new uint[](_pedingBetsLenght - i);
    for (uint j = 0; i < _pedingBetsLenght; i++) {
        newArray[j] = pendingBets[i];
        j++;
    }
    pendingBets = newArray;
}
```

RECOMMENDATION

We should map the id of the randomness request with a list of pending bets, and only settle these bets when the corresponding randomness value is returned. With this approach, the randomness result will not be expanded as outcomes for new pending bets.

UPDATES

- *March 28, 2022*: This issue has been acknowledged and fixed by the Polyflip team.

2.2.2. VRFManager.sol - Duplicated random values for all games LOW

In the fulfillRandomness function of the VRFManager contract, the randomness value is used to expand as outcomes for requested games. However, the expandedValues is reused to settle for all games leading to duplicated random values for all games.

```
function fulfillRandomness(bytes32, uint randomness) internal override {
    uint256[] memory expandedValues = expand(randomness, maxBetsPerGame);

    for (uint i = 0; i < games.length; i++) {
        if (gasleft() <= 150000) {
            return;
        }
        if (address(games[i]) != address(0)) {
            games[i].settleBet(expandedValues);
        }
    }
}
```

RECOMMENDATION

We should generate distinguish random values for each game, so the code can be fixed as below:

```
function fulfillRandomness(bytes32, uint randomness) internal override {
    uint256[] memory seeds = expand(randomness, games.length);

    for (uint i = 0; i < games.length; i++) {
        if (gasleft() <= 150000) {
            return;
        }
        if (address(games[i]) != address(0)) {
            uint256[] memory expandedValues = expand(seeds[i], maxBetsPer…
  Game);
            games[i].settleBet(expandedValues);
        }
    }
}
```

UPDATES

- *March 28, 2022*: This issue has been acknowledged by the Polyflip team. However, they think it's not a problem for the platform if the games will receive the same random numbers because each game will have different logic, so they can save some gas by generating only the necessary random numbers.

2.2.3. VRFManager.sol - Empty game should be removed from array LOW

In the removeGame function, when the game is removed, the corresponding game item is just been deleted and the games array size will keep increasing over time and lead to gas-consuming.

```
function removeGame(address toRemove) external onlyOwner {
    addressGame[toRemove] = false;
    for (uint i = 0; i < games.length; i++) {
        if (address(games[i]) == toRemove) {
            delete games[i];
            return;
        }
    }
}
```

RECOMMENDATION

The code can be fixed as below:

```
function removeGame(address toRemove) external onlyOwner {
    addressGame[toRemove] = false;
    for (uint i = 0; i < games.length; i++) {
        if (address(games[i]) == toRemove) {
            games[i] = games[games.length - 1];
            games.pop();
            return;
        }
    }
}
```

UPDATES

- *March 28, 2022*: This issue has been acknowledged and fixed by the Polyflip team.

2.3. Additional notes and recommendations

2.3.1. Typos INFORMATIVE

There are some typos in these contracts.

- _pedingBetsLenght in Coinflip.sol

- bettableAmount in Coinflip.sol
- amountToBettableAmountConverter in Manager.sol
- maxCoinsBettable in Manager.sol
- setMaxCoinsBettable in Manager.sol

UPDATES

- *March 28, 2022*: Some typos has been fixed by the Polyflip team.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Mar 28, 2022*	Public Report	Verichains Lab

Table 2. Report versions history