verichains

SECURITY AUDIT OF

WORLD OF DEFISH MARKETPLACE SMART CONTRACT

Public Report

Mar 16, 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 16, 2022. We would like to thank the World of Defish 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 World of Defish Marketplace 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 some vulnerable issues in the smart contracts code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About World of Defish Marketplace Smart Contract

World of Defish is a decentralized NFT gaming universe running on BSC where players immerse themselves in the delightful underwater universe.

With the care for best gaming quality in mind, World of Defish offers players the astonishing NFT world where they can set off their journey to the seven seas to look for the desired NFT fish.

Brave fishermen can upgrade their equipment and skills to improve fishing production. If the Fishermen don't have too much time on their hands, buying territories to earn passive income would surely be the way to go. Competitions and trading are also available to those who want to maximize their hard-earned profit. World of Defish has got you all covered!

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the World of Defish Marketplace Smart Contract.

It was conducted on commit 611002fa4dc6f4f2b2cee10b9fecb6f1938b6758 from git repository *https://github.com/wodtech/contracts/*.

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

SHA256 Sum	File
31dfacbbccce836d5820ca7984ec860154812877b79f162b58ebb0e057805711	**ItemsMarketplace.sol**
9c7dea211ec16f1caef33223430a0d58cf716d5b20fa8734eb93ec81a3f5ebe6	**MarketplaceAuctions.sol**
0687cbba898622b049bd0913087b2e893da3d72d3af13c4b86404b8497c583f4	**MarketplaceBase.sol**
a5f25ed8ccada66a7f8d77887e6be00908ec96747fb901055430889e1531605b	**MarketplaceOrders.sol**
c79ed215906253ab24238b42127297c372a094896b1ebce24f13decad4e97b49	**ZonesMarketplace.sol**
dafcedfc979b6a7c7882e3ab04ac0e6eda2c5082f1788e2520e7637ecf5b4513	**Roles.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 World of Defish Marketplace Smart Contract was written in Solidity language, with the required version to be 0.8.4. Almost all source codes in the World of Defish Marketplace Smart Contract imported OpenZeppelin contracts.

The source codes consist of two main contracts: Marketplace Orders and Marketplace Auctions. Marketplace Orders is the contract where users can buy/sell NFT and Marketplace Auctions help users buying and selling NFT by offering them up for bids, taking bids, and then selling the item to the highest bidder.

The contracts also includes a roles contract which supports RBAC (Role Based Access Control) to restricting system access to authorized users.

2.2. Findings

During the audit process, the audit team found some vulnerability issues in the given version of World of Defish Marketplace Smart Contract.

World of Defish fixed all the issues according to Verichain's private report in commit e6cde32a396203ab3b316e43f984ac8457991455.

2.2.1. MarketplaceBase.sol - Gas cost for _createOrder is increased along with number of orders MEDIUM

_createOrder function uses isTokenFree modifier which loop over array orders, as orders length is only increased, gas cost for each transaction is increased along and users will have to pay higher and higher fee every order count.

```
modifier isTokenFree(uint _token_id) {
    bool orderExists = false;
    for (uint i = 0; i < orders.length; i++) {
        if (orders[i].token_id == _token_id && orders[i].closed == false)…
   {
            orderExists = true;
            break;
        }
    }
    require(orderExists == false, 'Order for this token already exists');
    _;
}
```

RECOMMENDATION

Maintain a mapping for token_id => orderId to check whether the token is free or not.

UPDATES

- *Dec 16, 2021*: This issue has been acknowledged and fixed by the World of Defish team.

2.2.2. MarketplaceAuctions.sol - Wrong implement in finishAuction LOW

Consider the implementation of finishAuction:

```
function finishAuction(uint256 _orderId)
    external
    isAuction(_orderId)
    isOrderOpen(_orderId)
    isOrderOwner(_orderId)
{
    uint256 _betId;
    uint256 biggestPrice;
    for (uint i = 0; i < bets.length; i++) {
        if (bets[i].orderId == _orderId) {
            if (bets[i].price > biggestPrice) {
                if (_betId > 0) {
                    _closeBet(_betId);
                }

                _betId = i;
                biggestPrice = bets[i].price;
            } else {
                _closeBet(i);
            }
        }
    }
}
```

This implementation loop over bets, find the largest bet price, close any bets for the same order that is smaller. Variable _betId is used to store the last largest bet so far, initialized by 0.

Let's say bets[0] and bets[1] are for the same order, with the corresponding price of 100 and 200. After the first loop, _betId will be set to 0 (actually it doesnt change as it's already 0). After the second loop, _betId will be set to 1 **without closing bet 0**. So user that placed bet 0 will lose his fund.

UPDATES

- *Dec 14, 2021*: This issue has been acknowledged and fixed by the World of Defish team.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Dec 14, 2021*	Private Report	Verichains Lab
1.1	*Dec 16, 2021*	Public Report	Verichains Lab
1.2	*Mar 16, 2022*	Public Report	Verichains Lab

Table 2. Report versions history