0x Guard

Smart contracts security assessment

Final report

Tariff: Standard

Cybercash

November 2021

0xguard.com

hello@0xguard.com

Contents

Introduction

The report was prepared for the Cybercash team.

Name	Cybercash
Audit date	2021-11-18 - 2021-11-23
Language	Solidity
Platform	Binance Smart Chain

Contracts checked

Name	Address
All contracts	
Buy contracts	
BuyFighter	0x24517Caa6dF157601D27abe8f9044F167F8119fd
BuyGenElem	0x851d54dBf01aC9cFC03C7808a1faa563de003fd2
BuyLevelUp	0x42D3F4aeD978535ddcb5a903eEb0f0Df9026f325
BuySyndicate	0xfa2b291F92f186DDD982273610c9C3b45b6f0f08
All NFT contracts	
CyberTradeVehicle	0x2a54D740b994226E2fab0e4669eC5f68e1F1eb99
CyberTradeItem	0xB00f7788a3b502534751F8902dc0e5A41B3eE29e
CyberTradeFighter	0x8d972272D7264c8787113478410D71c12ee27b50
BuyItem	0x7ff1401E2beB68E7A346E1eBa1aa4539085681AC
BuyVehicle	0xd6d46c89Cb415f92aF0bB033d31f4654e958A146

Procedure

We perform our audit according to the following procedure:

Automated analysis

- Scanning the project's smart contracts with several publicly available automated Solidity analysis tools
- Manual verification (reject or confirm) all the issues found by the tools

Manual audit

- Manually analyse smart contracts for security vulnerabilities
- Smart contracts' logic check

Known vulnerabilities checked

Title	Check result
Unencrypted Private Data On-Chain	passed
Code With No Effects	passed
Message call with hardcoded gas amount	passed
Typographical Error	passed
DoS With Block Gas Limit	passed
Presence of unused variables	passed
Incorrect Inheritance Order	passed
Requirement Violation	passed
Weak Sources of Randomness from Chain Attributes	passed
Shadowing State Variables	passed
Incorrect Constructor Name	passed
Block values as a proxy for time	passed
Authorization through tx.origin	passed
DoS with Failed Call	passed

Delegatecall to Untrusted Callee	passed
Use of Deprecated Solidity Functions	passed
Assert Violation	passed
State Variable Default Visibility	passed
Reentrancy	passed
Unprotected SELFDESTRUCT Instruction	passed
Unprotected Ether Withdrawal	passed
Unchecked Call Return Value	passed
Floating Pragma	passed
Outdated Compiler Version	passed
Integer Overflow and Underflow	passed
Function Default Visibility	passed

Classification of issue severity

High severity	High severity issues can cause a significant or full loss of funds, change of contract ownership, major interference with contract logic. Such issues require immediate attention.
Medium severity	Medium severity issues do not pose an immediate risk, but can be detrimental to the client's reputation if exploited. Medium severity issues may lead to a contract failure and can be fixed by modifying the contract state or redeployment. Such issues require attention.
Low severity	Low severity issues do not cause significant destruction to the contract's functionality. Such issues are recommended to be taken into consideration.

Issues

High severity issues

1. Function buyLevelUp accepts payment but does not update state (BuyLevelUp)

Function buyLevelUp accepts and distributes payment but does not update the state to record the purchase.

Medium severity issues

1. Lack of automated tests (All contracts)

A project of this size needs automatic testing with unit test to ensure that all functionality works as intended.

2. SafeMath is not used (All contracts)

The contracts are compiled with Solidity version 0.7 that has no build-in check for integer overlows.

Recommendation: We recommend compiling contracts with Soldity version >= 0.8 or using the SafeMath library for unit operations.

3. Function addAvatar() can be frontrunned (Buy contracts)

Function addAvatar() sets the price for the avatar, but the function buyNFT() does not check that was previously set. An attacker may monitor transactions and call buyNFT() function with 1 wei before the addAvatar transaction was mined.

Recommendation: Check that the price is > 0 in the buyNFT() function.

4. Input parameters not checked (BuyFighter)

The owner in the function addWallet() can set a percent for distribution that is more than 100. This will lead to a failure of the buyNFT() function.

```
    function addWallet(uint8 _percentage, address payable _wallet) external
onlyContractOwner {
        distributionWallets.push(_wallet);
        walletPercentage[_wallet] = _percentage;
    }
```

Recommendation: Check sum of the percents and do not allow it to exceed the value of 100.

Low severity issues

1. No require messages (All contracts)

The require() statements lack error messages.

Recommendation: Add messages to the require() statements describing errors to improve user experience.

2. Pragma version is not fixed (All contracts)

The contracts may be compiled with wide range of the Solidity versions and differ in behaviour.

Recommendation: We recommend fixing the pragma version.

3. No unpause events (Buy contracts)

Event is emitted only on pause.

Recommendation: Emit event on unpause.

4. Wrong operator in distribution condition (Buy contracts)

The function buyNFT() before transferring to a wallet checks that the receiver address is not a zero address or has a non-zero percentage for distribution.

```
if (_wallet != address(0) || walletPercentage[_wallet] > 0) {
```

```
     _wallet.transfer(uint256(priceFor[_name] * uint256(walletPercentage[_wallet]) /
100));
}
```

If the wallet percentage is above zero and the receiver address is a zero address, the check will pass.

Recommendation: Use && instead of ||

5. Contract not fails on accidential BNB transfers to it (Buy contracts)

The contract has functions for receiving BNB:

```
    receive() external payable {

    }

    fallback() external payable {

    }
```

It's better to fail if BNB was accidentally sent to the contract address.

Recommendation: Remove receive() and fallback() functions to fail on accidental transfers to the contract.

6. Contract accepts any amount bigger than the NFT price (Buy contracts)

The buyNFT() function will accept a bigger amount of tokens or BNB if user sent if by mistake.

```
function buyNFT(string memory _name) external payable ifNotPaused {
        ...
    require(msg.value > priceFor[_name]);
}
```

Recommendation: Require string equality for price and BNB/token value

7. Unnecessary modificator on the getWallet() function (Buy contracts)

A view-function getWallet() has onlyContractOwner modifier that checks that the caller is a contract owner. When the function is called from the node the called address is not passed for the function and it will always fail.

Recommendation: Remove unnecessary modifier.

8. RandomNumberGenerator out of scope of the audit (BuyFighter)

The contract uses RandomNumberGenerator that is out of the scope of the current audit.

9. Return value of token transfers not checked (BuyGenElem)

The function buyGenElem() does not check the return value of token transfer in token.transferFrom() and token.tranfer() functions.

Recommendation: Check token transfer result:

```
bool success = token.transferFrom(msg.sender, address(this), priceToken);
require(success, "token transfer failed");
```

10. Token price should be set in the constructor (BuySyndicate)

If function setPrices() is not called users may buy syndicates for 1 wei.

11. Function buySyndicate iterates only for 2 of 3 syndicates (BuySyndicate)

The contract declares 3 syndicates:

```
string[] syndicates = ["Korpo", "Hackhunters", "Midnight"];
```

but in the buySyndicate() function iteration is done only for 2:

```
        uint256 i;
        bool contains = false;
        for (i = 0; i < 2; i++) {
            if (keccak256(bytes(_syndicate)) == keccak256(bytes(syndicates[i]))) {
```

```
                contains = true;
            }
        }
```

12. Tokens accepts accidentially sent BNB (All NFT contracts)

The TokenBase contract that is the parent of all NFTs in the scope accepts BNB with receive() and fallback() functions.

Recommendation: We recommend making the contract fail if someone send BNB to it.

13. Token name and symbol can be changed (All NFT contracts)

The change of name of the token may confuse users.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability)set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without 0xGuard prior written consent.

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts 0xGuard to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

0x Guard