SECURITY AUDIT OF

CRYPTOPLANES SMART CONTRACT

Public Report

Dec 03, 2021

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 Dec 03, 2021. We would like to thank the CryptoPlanes 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 CryptoPlanes 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 application.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY **5**

1.1. About CryptoPlanes Smart Contract **5**

1.2. Audit scope **5**

1.3. Audit methodology **5**

1.4. Disclaimer **7**

2. AUDIT RESULT **8**

2.1. Overview **8**

2.2. Contract codes **8**

2.2.1. Initializable abstract contract 8

2.2.2. ContextUpgradeable abstract contract 8

2.2.3. IERC20MetadataUpgradeable interface 8

2.2.4. IERC20Upgradeable interface 9

2.2.5. PausableUpgradeable abstract contract 9

2.2.6. OwnableUpgradeable abstract contract 9

2.2.7. Ownable abstract contract 9

2.2.8. ERC20Upgradeable contract 9

2.2.9. CPAN contract 9

2.2.10. Limiter contract 9

2.3. Findings **9**

2.3.1. CPAN_final.sol - Unused onSellSuccess on selling tokens LOW 9

2.3.2. CPAN_final.sol - Hardcoded pancakeswap v2 router address INFORMATIVE 12

2.3.3. CPAN_final.sol - Unchecked zero address INFORMATIVE 14

3. VERSION HISTORY **15**

1. MANAGEMENT SUMMARY

1.1. About CryptoPlanes Smart Contract

CryptoPlanes is an expansion of the CryptoCity metaverse. The place where the famous NFT racing game CryptoCars was founded.

With this expansion, CryptoPlanes brings new experiences, beautiful NFT Planes, and exciting game modes.

1.2. Audit scope

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

The audited contract is the CryptoPlanes Smart Contract that deployed on Binance Smart Chain Mainnet at address 0xc0e36a4c8bf041f0e54b82e2e32aa4c6c207505e (behind proxy at address 0x04260673729c5f2b9894a467736f3d85f8d34fc8). The details of the deployed smart contract are listed in Table 1.

FIELD	VALUE
Contract Name	CPAN
Contract Implement Address	0xc0e36a4c8bf041f0e54b82e2e32aa4c6c207505e
Contract Proxy Address	0x04260673729c5f2b9894a467736f3d85f8d34fc8
Compiler Version	v0.8.4+commit.c7e474f2
Optimization Enabled	No with 200 runs
Explorer	*https://bscscan.com/address/0xc0e36a4c8bf041f0e54b82e2e32aa4c6c207505e*

Table 1. 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 2. 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

This table lists some properties of the audited CryptoPlanes Smart Contract (as of the report writing time).

PROPERTY	VALUE
Name	CryptoPlanes
Symbol	CPAN
Decimals	18
Total Supply	100,000,000 (x10^{18}) Note: the number of decimals is 18, so the total representation token will be 100,000,000 or 100 million.

Table 3. The CryptoPlanes Smart Contract properties

2.2. Contract codes

The CryptoPlanes Smart Contract was written in Solidity language, with the required version to be 0.8.2.

The source codes consist of three contracts, five abstract contracts and two interfaces. Almost all source codes in the CryptoPlanes Smart Contract imported OpenZeppelin contracts.

2.2.1. Initializable abstract contract

This is a base contract to aid in writing upgradeable contracts, or any kind of contract that will be deployed behind a proxy. The source code is referenced from OpenZeppelin's implementation.

2.2.2. ContextUpgradeable abstract contract

Provides information about the current execution context, including the sender of the transaction and its data. The source code is referenced from OpenZeppelin's implementation.

2.2.3. IERC20MetadataUpgradeable interface

Interface for the optional metadata functions from the ERC20 standard. The source code is referenced from OpenZeppelin's implementation.

2.2.4. IERC20Upgradeable interface

Interface of the ERC20 standard as defined in the EIP. The source code is referenced from OpenZeppelin’s implementation.

2.2.5. PausableUpgradeable abstract contract

Contract module allows children to implement an emergency stop mechanism that can be triggered by an authorized account. The source code is referenced from OpenZeppelin’s implementation.

2.2.6. OwnableUpgradeable abstract contract

Contract module which provides a basic access control mechanism, where there is an account (an owner) that can be granted exclusive access to specific functions. The source code is referenced from OpenZeppelin’s implementation.

2.2.7. Ownable abstract contract

Contract module which provides a basic access control mechanism, where there is an account (an owner) that can be granted exclusive access to specific functions. The source code is referenced from OpenZeppelin’s implementation.

2.2.8. ERC20Upgradeable contract

This is the contract implement ERC20 token with upgradable ability, adding transfer fee feature.

2.2.9. CPAN contract

This is the main contract in the CryptoPlanes Smart Contract, which is inherited from ERC20Upgradeable contract.

2.2.10. Limiter contract

This contract implement buy/sell limit for CPAN contract.

2.3. Findings

The CryptoPlanes Smart Contract was written in Solidity language, with the required version to be ^0.8.2. The source code was written based on OpenZeppelin's library.

2.3.1. CPAN_final.sol - Unused onSellSuccess on selling tokens LOW

When buying/selling token with pancakeswap, the contract use limiter to limit amount of tokens which one can buy/sell.

When buying, transfer function uses _limiter.onBuySuccess callback to increase _totalBuy of an address but when that address sells tokens with transferFrom, _limiter.onSellSuccess callback doesn't fire so _totalBuy doesn't decrease and break the logic (_limitBuy) of the limiter.

```
function transferFrom(
    address sender,
    address recipient,
    uint256 amount
) public virtual override returns (bool) {
    require(!_isLockTransferForAntiBot || sender == _lpCreatorAddress, "C…
  PAN: Antibot revert");
    if (isAddressLP(recipient)) {
        // SELL
        uint256 _fee = 0;
        if (sender != _lpCreatorAddress) {
            require(_limiter.isValidSell(amount), "CPAN: Sell limit reach…
  ");
            _fee = calculateFee(amount, _sellFee);
        }
        return _transferFrom(sender, recipient, amount, _fee, _feeReceive…
  Address);
    }
    return _transferFrom(sender, recipient, amount, 0, address(0));
}
function transfer(address recipient, uint256 amount) public virtual overr…
  ide returns (bool) {
    if (isAddressLP(_msgSender()) || isAddressLP(recipient)) {
        // BUY
        require(!_isLockTransferForAntiBot || recipient == _lpCreatorAddr…
  ess, "CPAN: Antibot revert");
        uint256 _fee = 0;
        if (recipient != _lpCreatorAddress && recipient != 0x10ED43C71871…
  4eb63d5aA57B78B54704E256024E) {
            require(_limiter.isValidBuy(recipient, amount), "CPAN: Buy li…
  mit reach");
            _fee = calculateFee(amount, _buyFee);
            if (isAddressLP(recipient)) {
                _fee = calculateFee(amount, _sellFee);
            }
        }
        _transfer(_msgSender(), recipient, amount, _fee, _feeReceiveAddre…
  ss);
```

```
        if (recipient != _lpCreatorAddress) {
            _limiter.onBuySuccess(recipient, amount);
        }
    } else {
        require(recipient != address(this), "CPAN: Failed to transfer");
        _transfer(_msgSender(), recipient, amount, 0, address(0));
    }
    return true;
}
function onBuySuccess(address _sender, uint256 _amount) public {
    _totalBuy[_sender] += _amount;
}
function onSellSuccess(address _sender, uint256 _amount) public {
    uint256 value = _totalBuy[_sender];
    if (value < _amount) {
        _totalBuy[_sender] = 0;
    } else {
        _totalBuy[_sender] = (value - _amount);
    }
}
function isValidBuy(address _sender, uint256 _amount) public view returns…
  (bool) {
    if (_amount > _maxBuy) {
        return false;
    }
    uint256 totalBuyOfAddress = _totalBuy[_sender];
    if (totalBuyOfAddress + _amount > _limitBuy) {
        return false;
    }
    return true;
}
```

RECOMMENDATION

Consider call the onSellSuccess when users sell tokens.

```
function transferFrom(
    address sender,
    address recipient,
    uint256 amount
) public virtual override returns (bool) {
    require(!_isLockTransferForAntiBot || sender == _lpCreatorAddress, "C…
  PAN: Antibot revert");
```

```
    if (isAddressLP(recipient)) {
        // SELL
        uint256 _fee = 0;
        if (sender != _lpCreatorAddress) {
            require(_limiter.isValidSell(amount), "CPAN: Sell limit reach…
  ");
            _fee = calculateFee(amount, _sellFee);
        }
        _limiter.onSellSuccess(sender, amount); // add this
        return _transferFrom(sender, recipient, amount, _fee, _feeReceive…
  Address);
    }
    return _transferFrom(sender, recipient, amount, 0, address(0));
}
```

UPDATES

- *Dec 03, 2021*: This issue has been acknowledged by the CryptoPlanes team.

2.3.2. CPAN_final.sol - Hardcoded pancakeswap v2 router address INFORMATIVE

The pancakeswap v2 router address is hardcoded in the transfer function which could confuse readers.

```
function transfer(address recipient, uint256 amount) public virtual overr…
  ide returns (bool) {
    if (isAddressLP(_msgSender()) || isAddressLP(recipient)) {
        // BUY
        require(!_isLockTransferForAntiBot || recipient == _lpCreatorAddr…
  ess, "CPAN: Antibot revert");
        uint256 _fee = 0;
        if (recipient != _lpCreatorAddress && recipient != 0x10ED43C71871…
  4eb63d5aA57B78B54704E256024E) {
            require(_limiter.isValidBuy(recipient, amount), "CPAN: Buy li…
  mit reach");
            _fee = calculateFee(amount, _buyFee);
            if (isAddressLP(recipient)) {
                _fee = calculateFee(amount, _sellFee);
            }
        }
        _transfer(_msgSender(), recipient, amount, _fee, _feeReceiveAddre…
  ss);
        if (recipient != _lpCreatorAddress) {
            _limiter.onBuySuccess(recipient, amount);
        }
    } else {
        require(recipient != address(this), "CPAN: Failed to transfer");
        _transfer(_msgSender(), recipient, amount, 0, address(0));
    }
    return true;
}
```

RECOMMENDATION

Consider use a constant variable to store the address.

```
address constant pancakeSwapRouterV2 = 0x10ED43C718714eb63d5aA57B78B54704…
  E256024E; // add this

function transfer(address recipient, uint256 amount) public virtual overr…
  ide returns (bool) {
    if (isAddressLP(_msgSender()) || isAddressLP(recipient)) {
        // BUY
        require(!_isLockTransferForAntiBot || recipient == _lpCreatorAddr…
  ess, "CPAN: Antibot revert");
        uint256 _fee = 0;
        if (recipient != _lpCreatorAddress && recipient != pancakeSwapRou…
  terV2) {
            require(_limiter.isValidBuy(recipient, amount), "CPAN: Buy li…
  mit reach");
            _fee = calculateFee(amount, _buyFee);
            if (isAddressLP(recipient)) {
                _fee = calculateFee(amount, _sellFee);
            }
        }
        _transfer(_msgSender(), recipient, amount, _fee, _feeReceiveAddre…
  ss);
        if (recipient != _lpCreatorAddress) {
            _limiter.onBuySuccess(recipient, amount);
        }
    } else {
        require(recipient != address(this), "CPAN: Failed to transfer");
        _transfer(_msgSender(), recipient, amount, 0, address(0));
    }
    return true;
}
```

UPDATES

- *Dec 03, 2021*: This issue has been acknowledged by the CryptoPlanes team.

2.3.3. CPAN_final.sol - Unchecked zero address INFORMATIVE

The contract check for zero address in setFeeReceiveAddress but doesn't check in other functions.

```
function setRewardAddress(address newAddress) public onlyOwner {
    _rewardAddress = newAddress;
}
function setFeeReceiveAddress(address addr) public onlyOwner {
    require(addr != address(0), "ERC20: Invalid receive address");
    _feeReceiveAddress = addr;
}
function setLPCreatorAddress(address _addr) external onlyOwner {
    _lpCreatorAddress = _addr;
}
```

RECOMMENDATION

Consider check for zero address in all the function which set address variable.

```
function setRewardAddress(address newAddress) public onlyOwner {
    require(newAddress != address(0), "ERC20: Invalid reward address"); /…
  / add this
    _rewardAddress = newAddress;
}
function setFeeReceiveAddress(address addr) public onlyOwner {
    require(addr != address(0), "ERC20: Invalid receive address");
    _feeReceiveAddress = addr;
}
function setLPCreatorAddress(address _addr) external onlyOwner {
    require(addr != address(0), "ERC20: Invalid LP creator address"); // …
  add this
    _lpCreatorAddress = _addr;
}
```

UPDATES

- *Dec 03, 2021*: This issue has been acknowledged by the CryptoPlanes team.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Nov 30, 2021*	Public Report	Verichains Lab
1.1	*Dec 03, 2021*	Public Report	Verichains Lab

Table 4. Report versions history