verichains

SECURITY AUDIT OF

PIGU LAND SMART CONTRACTS

Public Report

Jan 13, 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 Jan 13, 2022. We would like to thank the Pigu Land 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 Pigu Land 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. Pigu Land has acknowledged and fixed those issues.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Pigu Land Smart Contracts

Pigu Land is a blockchain-based game where all characters are penguins, living in a kingdom called Zeal. Each penguin is an unique NFT that is truly owned by the player.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the smart contracts of Pigu Land name. It was conducted on the source code provided by the Pigu Land team.

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 initial review was conducted on Jan 03, 2021 and a total effort of 3 working days was dedicated to identifying and documenting security issues in the code base of Pigu Land contracts.

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

FILE	SHA256 SUM
PreSale.sol	7037788ec5d6d99cc7e0b180defeecc31e7e51fcb3c27c22345b17df88ca23ae
FishToken.sol	b139c4c33c6f763fd6d06c2ff089b87308c3c6696decacfd0912fabddccea1af
PiguToken.sol	3037b990fb3699aca78947481ced338954f914aec05079082e52f16dab141c0f
TimeLockedWallet.sol	85d616fb85d360ed7070f68457fdd55a8566ee9001630a6982c2523b80d45324
TimeLockedWalletFactory.sol	55b3fee6192b7f09f616a16c989d28b41db980767b129a81fce40dea07feff21

2.2. Contract codes

The Pigu Land Smart Contracts was written in Solidity language, with the required version to be 0.8.0.

2.2.1. FishToken contract

FishToken is an ERC20 token contract. The contract inherits the ERC20 contract of Openzepplin. The contract implements mint public function which allows the owner of the contract mint new token. The totalSupply of the contract can be changed by this function. The owner also burn the token of any address through the burn public function.

2.2.2. PiguToken contract

PiguToken is an ERC20 token contract. As the FishToken contract, the contract inherits the ERC20 contract of Openzepplin. The contract implements mint public function which allows

the owner of the contract mint new token. The totalSupply of the contract can be changed by this function. The owner also burn the token of any address through burn public function.

2.2.3. PreSale contract

The PreSale contract supports the users to buy tokens. Currently, the buy function only saves the state of the tokens that the users bought. The buy function doesn't include any transfer method to trade the tokens.

2.2.4. TimeLockedWalletFactory contract

TimeLockedWalletFactory contract supports the Pigu Land team to manage TheLockedWallets releasing the tokens that investors bought. For each investor, this contract creates a new TimeLockedWallet which holds all info of this investor.

2.2.5. TimeLockedWallet contract

TimeLockedWallet is the contract that holds the tokens of the investor. The contract supply info about vesting and allow investor claiming tokens. There are several TimeLockedWallet contracts working at the same time. Each TimeLockedWallet contract supports a specific investor which is set by the creator.

2.3. Findings

During the audit process, the audit team found some vulnerability issues in the given version of Pigu Land Smart Contracts.

2.3.1. PreSale.sol - The logic of buy function isn't clear CRITICAL

In the buy function, we found that the function just updates the state of tokens. It does not like a trading logic. There are some limitations of tokens that users can buy. But the users just call buy with the maximum amount and the state of variables will be updated. It doesn't have restrictions like the value of native token user must send,...

The amount value of users in the addressToTokensHolding variable is added twice after the first time. It seems to be incorrect. It may be correct if it updates data into the addressToTokensBought value.

In addition, the code at the line 46 is redundant because it is covered by the line 120.

```
44  function buy(uint amount) public
45      {
46          require (addressToTokensHolding[msg.sender].amount < addressT…
    oAllocations[msg.sender], "Reached maximum allocation");
47          require(availableTokens > 0, "No more tokens to buy");
```

```
48
49          require (amount <= addressToAllocations[msg.sender] - addres…
    sToTokensHolding[msg.sender].amount, "Cannot buy more than alloca…
    tion");
50          require (amount <= availableTokens, "Cannot buy this amount o…
    f tokens");
51
52          require (amount > 0, "cannot buy 0 tokens");
53
54          //bool txh = token.transferFrom(owner(), msg.sender, amount);
55          //require (txh);
56
57          if (addressToTokensHolding[msg.sender].amount == 0)
58          {
59              TokenHoldings memory entry;
60              entry.owner = msg.sender;
61              entry.amount = amount;
62              addressToTokensBought.push(entry);
63          }
64          else
65          {
66              addressToTokensHolding[msg.sender].amount += amount;
67          }
68
69          addressToTokensHolding[msg.sender].amount += amount;
70          availableTokens -= amount;
71          boughtTokens += amount;
72      }
```

Snippet 1. Unclear logic in the `buy` function

RECOMMENDATION

We suggest updating the logic of this function.

UPDATES

- *Jan 13,2022*: This issue has been acknowledged and fixed by the Pigu Land team.

2.3.2. PiguToken, FishToken - The owner of contract may control totalSupply MEDIUM

Currently, the contract uses a centralized mechanism to allow the owner to control the totalSupply value. The owner can call mint public function to mint an unlimited amount of the tokens and burn the tokens of any address wallet.

Any compromise to the owner account may allow the hacker to take advantage of this.

RECOMMENDATION

We strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices.

UPDATES

- *Jan 13,2022*: This issue has been acknowledged by the Pigu Land team

2.3.3. TimeLockedWallet.sol - Unsafe using transfer method through IERC20 interface LOW

In the withdrawTokens function, the contract uses transfer method to call the function from the token contract. With the FishToken and PiguToken contracts in the audit scope, it doesn't have any problems but the contract doesn't point exactly what token was used in the contract. So we can't ensure that the transfer 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 createTimeLockedWallet logic, the owner doesn't receive anything while the totalClaimedTokens is updated.

```
53  function withdrawTokens(address _tokenContract)  public onlyOwner non…
    Reentrant
54      {
55          uint claimableTokens = getClaimableTokens();
56
57          require(claimableTokens > 0, 'TimeLockedwallet: No claimable t…
    okens');
58
59          IERC20 token = IERC20(_tokenContract);
60
61          token.transfer(owner, claimableTokens);
62          totalClaimedTokens += claimableTokens;
63
64          emit WithdrewTokens(_tokenContract, msg.sender, claimableToken…
    s);
65      }
```

Snippet 2. TimeLockedWallet.sol Unsafe using `transfer` method in `withdrawTokens` function

RECOMMENDATION

We suggest using SafeERC20 library for IERC20 and changing all transfer, transferFrom method used in the contract to safeTransfer, safeTransferFrom which is declared in SafeERC20 library to ensure that there is no issue when transferring tokens.

UPDATES

- *Jan 13,2022*: This issue has been acknowledged and fixed by the Pigu Land team.

2.3.4. TimeLockedWalletFactory.sol - Unsafe using transferFrom method through IERC20 interface LOW

In the createTimeLockedWallet function, the contract use transferFrom method to call function from the token contract. With the FishToken and PiguToken contracts in the audit scope, it doesn't have any problems but the contract doesn't point exactly what token was used in the contract. So we can't ensure that the transfer 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 createTimeLockedWallet logic, the walletAdr doesn't receive anything which causes an issue in that LockedWallet.

```
58  function createTimeLockedWallet(address _owner, uint _tokenAmount) on…
    lyOwner payable public returns(address walletAdr)
59      {
60          TimeLockedWallet wallet = new TimeLockedWallet(address(this) …
    , msg.sender, _owner, _tokenAmount);
61          walletAdr = address(wallet);
62
63          if(msg.sender != _owner)
64          {
65              wallets[_owner].push(walletAdr);
66          }
67
68          //Transfer Tokens
69          token.approve(address(this), _tokenAmount);
70          token.transferFrom(address(this), walletAdr, _tokenAmount);
71
72          emit WalletCreated(walletAdr, _owner, _tokenAmount);
73      }
```

Snippet 3. TimeLockedWalletFactory.sol Unsafe using `transfer` method in `createTimeLockedWallet` function

RECOMMENDATION

We suggest using SafeERC20 library for IERC20 and changing all transfer, transferFrom method used in the contract to safeTransfer, safeTransferFrom which is declared in SafeERC20 library to ensure that there is no issue when transferring tokens.

UPDATES

- *Jan 13,2022*: This issue has been acknowledged and fixed by the Pigu Land team.

2.4. Additional notes and recommendations

2.4.1. Unuse SafeMath in contracts INFORMATIVE

In the head of all files that in the audit scope, the contracts imported SafeMath library but it doesn't use inside the contract. In addtion, the SafeMath checking overflow is unnessary because solidity 0.8.0+ already do that by default.

RECOMMENDATION

We suggest removing this library for readability.

UPDATES

- *Jan 13,2022*: This issue has been acknowledged and fixed by the Pigu Land team.

2.4.2. Missing checks length of array functions INFORMATIVE

There are some functions in the audit scope that use array variables but noncheck the length of this array variable. Therefore, the function can access an index which not inbound of the array causes an error in the function.

For instance with the _balances variable in batchMint in PiguToken.sol file.

```
function batchMint(address[] memory _recipients, uint256[] memory _balanc…
  es) public payable onlyOwner{
        uint256 total = 0;
        for (uint i = 0; i < _recipients.length; i++) {
            transferFrom(owner(), _recipients[i], _balances[i]);
            total += _balances[i];
        }
    }
```

Snippet 4. Missing check array length in `_balances` function

The batchMint function in FishToken.sol files, batchSetTokensAllocation function in PreSale.sol files and batchCreateTimeLockedWallet function in TimeLockedWalletFactory.sol have the same issue.

RECOMMENDATION

We suggest adding a require statement to check the length of array variables in those functions like the below code:

```
function batchMint(address[] memory _recipients, uint256[] memory _balanc…
  es) public payable onlyOwner{
        uint256 total = 0;
        require(_balances.length == _recipients.length, "batchMint: array…
   length is invalid");
        for (uint i = 0; i < _recipients.length; i++) {
            transferFrom(owner(), _recipients[i], _balances[i]);
            total += _balances[i];
        }
    }
```

Snippet 5. Recommend fixing in `_balances` function

UPDATES

- *Jan 13,2022*: This issue has been acknowledged and fixed by the Pigu Land team.

APPENDIX

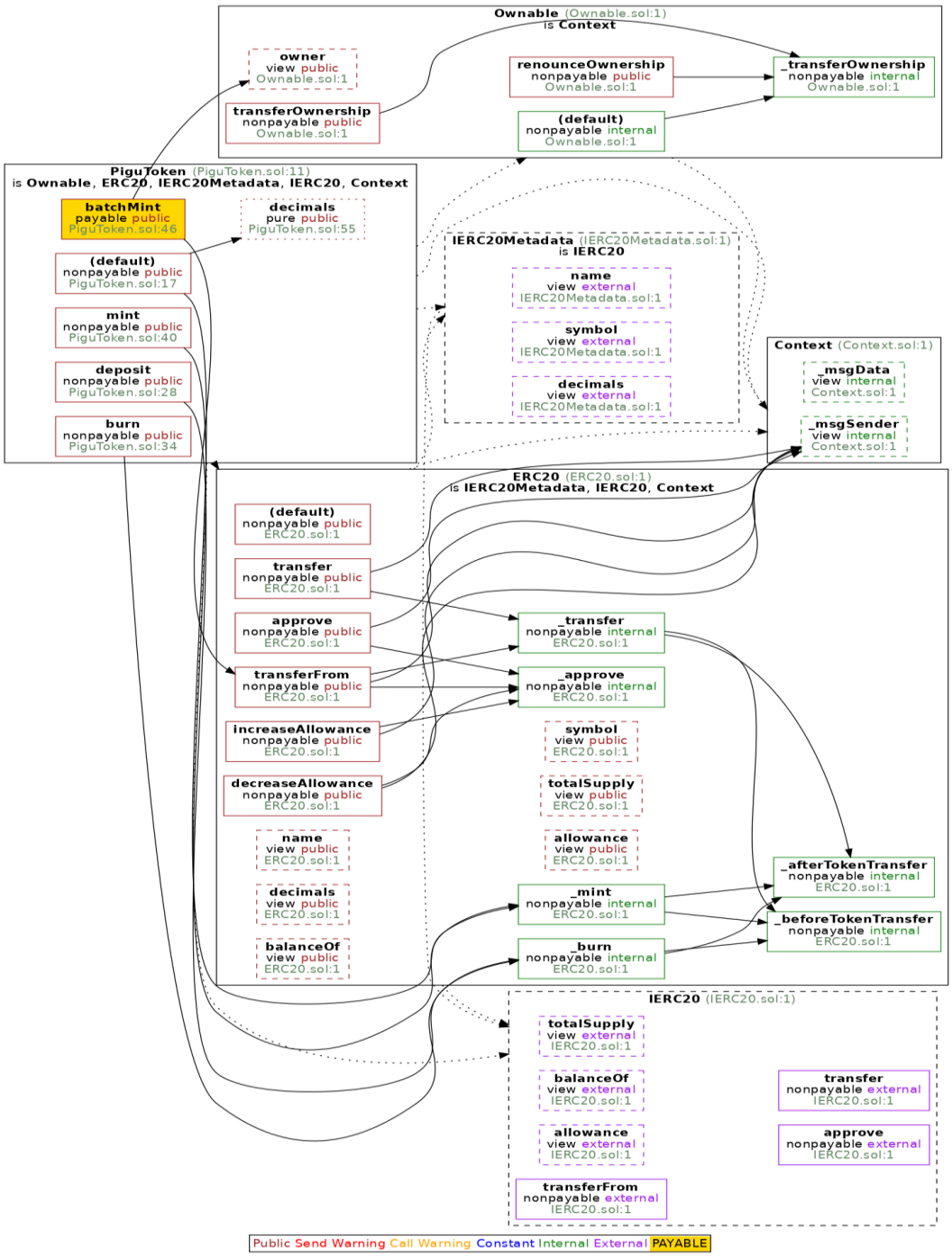

Image 1. PiguToken call graph

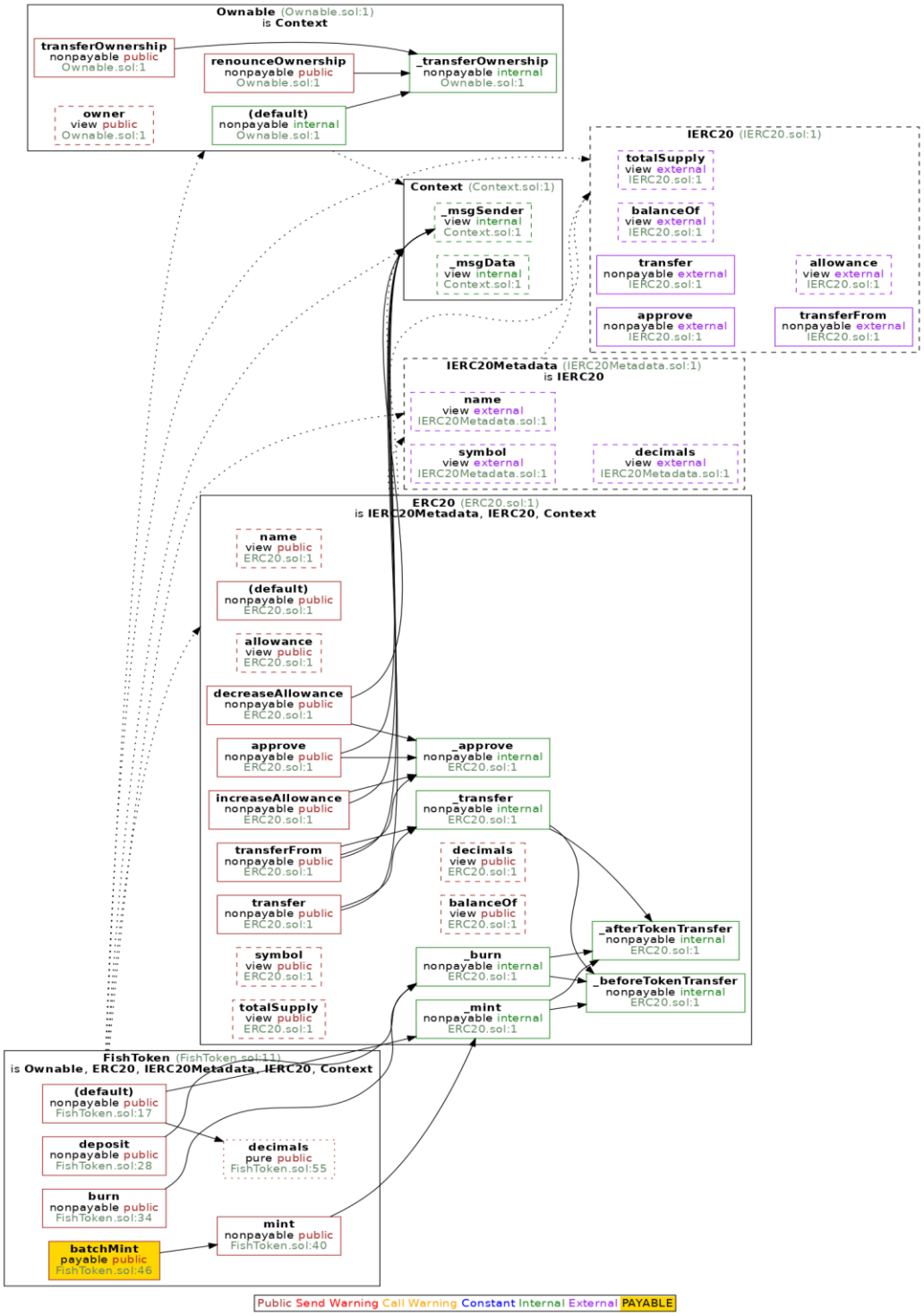

Image 2. FishToken call graph

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Jan 06, 2022*	Private Report	Verichains Lab
1.1	*Jan 13, 2022*	Public Report	Verichains Lab

Table 2. Report versions history