verichains

SECURITY AUDIT OF

PEOPLE'S TELCO

people's telco

Public Report

May 08, 2023

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.
ERC721	The ERC-721 introduces a standard for NFT, in other words, this type of Token is unique and can have different value than another Token from the same Smart Contract, maybe due to its age, rarity or even something else like its visual

EXECUTIVE SUMMARY

This Security Audit Report was prepared by Verichains Lab on May 08, 2023. We would like to thank the PeopleTelco 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 People's Telco. 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.

In a contract ERC721TradeOperator which provide features to trade ERC721 tokens, the functions `makeOffer` and `acceptOffer` are vulnerable to a logic and frontrunning attack.

The following table summarizes the main issues identified and the action items to be taken to mitigate them. It allows users to own names intended for People's Telco users. Each name instends to be assigned to any eSims as part of its main utility. Although this may look similar to an ENS naming service, this service isn't a domain based based one and all names have no expiry. A name can be mapped into multiple eSim numbers but it is only limited to 1 number per country specifically. It is also able to map out other data through the use of multiple `FieldNameResolver` where each one can serve to store and resolve a specific field such as email, homepage, etc.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About People's Telco

People’s Telco is the first global web3 telco that allows you to own your mobile identity and be connected anytime and anywhere. With one application, users get seamless access to the the best global data plans, telco services, and web3 digital assets. People’s Telco is backed by global telco and web3 leaders.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the People's Telco.

It was conducted on commit `f974ee78d10e4e0826d44a2ed40f173c9f732887` from git repository *https://bitbucket.org/eden-holdings/pt-solidity-contracts*.

The patched version is commit `db1b2d88e3b7519ce859261926919650578edc77`.

1.3. Audit methodology

Our security audit process includes four steps:

- Mechanism Design is reviewed to look for any potential problems.
- Source codes are scanned/tested for commonly known and more specific vulnerabilities using public and our in-house security analysis tool.
- Manual audit of the codes for security issues. The source code is manually analyzed to look for any potential problems.
- Set up a testing environment to debug/analyze found issues and verifies our attack PoCs.

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 functioning; creates a critical risk to the application; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the application with high impact; needs to be fixed with high priority.

SEVERITY LEVEL	DESCRIPTION
MEDIUM	A vulnerability that could affect the desired outcome of executing the application 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 application. 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

People's Telco contains 2 main contracts: `ERC721TradeOperator`, `PeoplesTelcoNS` and some contracts provide service resolver for name and sim.

2.1.1. ERC721TradeOperator contract

This is the marketplace contract in the People's Telco, which extends `ReentrancyGuard`, `IERC721Receiver` and `Ownable` contracts. With `Ownable`, by default, contract owner is contract deployer, but he can transfer ownership to another address at any time. This contract is the place where users can buy/sell NFT. Users can list their NFTs for sale.

2.1.2. PeoplesTelcoNS contract

This is an ERC721 based contract in the People's Telco, which extends `EIP712MetaTxBase`, `IPeoplesTelcoRegistry`, `Ownable` contract. With `Ownable`, by default, contract owner is contract deployer, but he can transfer ownership to another address at any time.

2.2. Findings

This section contains a detailed analysis of all the vulnerabilities that were discovered by the audit team during the audit process.

2.2.1. ERC721TradeOperator contract - Sellers can receive payment for their orders that is less than they had expected - CRITICAL

The contract is designed to allow sellers set their NFTs for sales and buyers will make an offer to purchase the NFT. The buyer can update if they are not satisfied with the offer and set another offer with a lower price.

A case that is not a valid use case is that the buyer will make an offer if they see the seller's NFTs for sale. While the buyer is waiting for the seller manually accept the offer (call a function `acceptOffer`), the buyer will run a transaction that call function `makeOffer` again to update a cheaper price. To perform this case, the buyer just runs a transaction with a higher gas price than the seller (as known as frontrunning).

Finally, The buyer will purchase the order at a price lower. Following the vulnerable code:

```
function acceptOffers(address nftContract, uint256[] calldata ids, address[] calldata
buyers)
    isAllowedContract(nftContract)
    nonReentrant
    external {

    uint256 offerCount = ids.length;
    mapping(uint256 => EnumerableMap.AddressToUintMap) storage nftOffers =
offers[nftContract];

    for(uint256 i = 0; i < offerCount; i++){
        uint256 id = ids[i];
        address buyer = buyers[i];
        uint256 amount = nftOffers[id].get(buyer);

        __offerIsValid(nftContract, id, buyer, amount);
        _transactNft(nftContract, id, payable(_msgSender()), buyer, amount);
    }
}
```

A reason of the bug is that the function `acceptOffers` get the offer amount from the mapping `nftOffers` by id and buyer's address without checking if the offer is the latest offer. This allows the buyer to update their offer with a lower price and the seller will accept the offer with the lower price.

RECOMMENDATION

We advise to add a check to ensure that the offer is the latest offer. Amount of the best offer should be passed as a parameter of the function `acceptOffers` instead of getting from the mapping `nftOffers`.

UPDATES

- *Apr 28, 2023*: The team has fixed the bug by adding a check to ensure that the offer is the latest offer. Price of the offers are passed as a parameter of the function `acceptOffers`.

2.2.2. ERC721TradeOperator contract - Buyer cannot update their offer if this offer is the lowest offer - HIGH

A function `_insertOffer` of a contract ERC721TradeOperator inserts a new offer into the list of offers for a specific NFT. When offer's length is greater than bidderLimit (default is 100), the function will remove the lowest offer from the list of offers.

However, the function does check if the buyer who make the lowest offer and a length of offers is over the limit. This disallows the buyer to update their offer. The case is not a valid use case. Following the code:

```
function _insertOffer(address nftContract, uint256 id, uint256 amount, address sender)
internal {
        mapping(uint256 => EnumerableMap.AddressToUintMap ) storage nftOffers =
offers[nftContract];
        uint256 userBalance = userBalances.contains(sender) ? userBalances.get(sender) : 0;

        bool eligible  = true;

        uint256 currBids = nftOffers[id].length();
        uint256 lowest = amount;
        address replace = sender;
        if(currBids >= bidderLimit){
            // find an offer that can be replaced
            for(uint256 j = 0; j < currBids; j++){
                (address bidder, uint256 bid) = nftOffers[id].at(j);
                if(bid < lowest){
                    replace = bidder;
                    lowest = bid;
                }
            }
            if(replace == sender){
                eligible = false;
            }
        }
        // if userBalance is still enough for the rest, place offer
        if( ((userBalance + 1) > amount) && eligible){
            // take out the low bid
            // so we can replace it soon
            nftOffers[id].remove(replace);
            // post offer
            nftOffers[id].set(sender, amount);
            emit OfferPlaced(nftContract, id, sender, amount);
        } else {
            // ignore if fund is insufficient
            // or not eligible due to low offer
        }
    }
```

RECOMMENDATION

We advise to change a replace variable to the zero address and add a check to ensure that the replace variable is not the zero address. This allows the buyer to update their offer if their offer is the lowest offer.

```
function _insertOffer(address nftContract, uint256 id, uint256 amount, address sender)
internal {
        // ...
        address replace = address(0);

        if(currBids >= bidderLimit){
            // ...

            if(replace == address(0)){
                eligible = false;
            }
        }
```

UPDATES

- *Apr 28, 2023*: The team has fixed the bug by changing a replace variable to the zero address and add a check to ensure that the replace variable is not the zero address.

2.2.3. ERC721TradeOperator contract - Bypass check user's balance before make offers - MEDIUM

A function `makeOffer` of a contract ERC721TradeOperator has a guard condition that checks if the user has enough balance to make offers.However, the check using a loop each amount of offers to check user's balance. If the user has enough balance to make the offer with the largest amount, then the user will pass the check. This allows the user to bypass the check and make offers with a total value greater than their balance.

```
function makeOffer(address nftContract, uint256[] calldata ids, uint256[] calldata amounts)
        isEnabled isAllowed isAllowedContract(nftContract) nonReentrant payable external {

    require(ids.length == amounts.length, "ids and prices mismatch");
    uint256 offerCount = ids.length;

    // ...
    uint256 userBalance = userBalances.contains(sender) ? userBalances.get(sender) : 0;
    for(uint256 i = 0; i < offerCount; i++){
        if(userBalance < amounts[i]) revert OfferBudgetFailed(userBalance, amounts[i]);
    }
}
```

RECOMMENDATION

Add a check to ensure that the user has enough balance to make offers.

```
function makeOffer(address nftContract, uint256[] calldata ids, uint256[] calldata amounts)
        isEnabled isAllowed isAllowedContract(nftContract) nonReentrant payable external {

    require(ids.length == amounts.length, "ids and prices mismatch");
    uint256 offerCount = ids.length;

    // ...
    uint256 userBalance = userBalances.contains(sender) ? userBalances.get(sender) : 0;
    uint totalOffersAmount = 0;
    for(uint256 i = 0; i < offerCount; i++){
        totalOffersAmount = totalOffersAmount.add(amounts[i]);
    }
    if(userBalance < totalOffersAmount) revert OfferBudgetFailed(userBalance,
totalOffersAmount);

}
```

UPDATES

- *Apr 28, 2023*: The team has been acknowledged the bug.

2.2.4. ERC721TradeOperator contract - Owner of a NFT can make a offer for the their own NFT - LOW

A function `makeOffer` of a contract ERC721TradeOperator allows potential buyers to make an offer to purchase an NFT from the owner. It provides an opportunity for negotiation between the buyer and the seller to reach a mutually acceptable price for the NFT.

The function lacks a check that the owner of the NFT is not the same as the buyer. This allows the owner of the NFT to make an offer to themselves, which is not a valid use case.

RECOMMENDATION

Add a check to ensure that the owner of the NFT is not the same as the buyer.

```
function makeOffer(address nftContract, uint256[] calldata ids, uint256[] calldata amounts)
        isEnabled isAllowed isAllowedContract(nftContract) nonReentrant payable external {

    require(ids.length == amounts.length, "ids and prices mismatch");

    uint256 offerCount = ids.length;

    address sender = _msgSender(); // move to the top of the function

    // check if token ids exists
    for(uint256 i = 0; i < offerCount; i++){
        require(IERC721(nftContract).ownerOf(ids[i]) != address(0), "Token ID fail");
        require(IERC721(nftContract).ownerOf(ids[i]) != sender, "Buyer is owner"); // add
this line
    }
    //...
}
```

UPDATES

- *Apr 28, 2023*: The team has been acknowledged and fixed by the PeopleTelco team.

2.2.5. SafeMath unnecessary in Solidity version ^0.8. INFORMATIVE

The SafeMath libraries were necessary in previous versions of Solidity to ensure that arithmetic operations would not result in overflow or underflow. However, Solidity version ^0.8 now includes built-in checks to prevent these issues.The addition of built-in arithmetic overflow and underflow protection in Solidity version ^0.8 means that the use of SafeMath is redundant.

RECOMMENDATION

Remove the SafeMath and SafeMath64 libraries and use the built-in arithmetic overflow and underflow protection in Solidity version ^0.8.

UPDATES

- *Apr 28, 2023*: The team has been acknowledged and fixed by the PeopleTelco team.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Apr 21, 2023*	Private Report	Verichains Lab
1.1	*Apr 28, 2023*	Private Report	Verichains Lab
1.2	*May 08, 2023*	Public Report	Verichains Lab

Table 2. Report versions history