MobiePay, LLC
Smart Contract Audit

Review and Remediation

Presented by:

Red Lion, LLC

POC: Scott Lyons

Auditor: Jonathan Scott

Presented to:

MobiPay, LLC

Jun 23, 2022

Table of Contents

Legal

Report Disclaimer Statement

This disclaimer governs the use of this report. Customer shall own all right, title, and interest in and to any written summaries, reports, analyses, and findings or other information or documentation prepared for Customer in connection with Red Lion, LLC ("Red Lion") consulting services to Customer. Red Lion specifically disclaim any and all liability for any damages whatsoever (whether foreseen or unforeseen, direct, indirect, consequential, incidental, special, exemplary or punitive) arising from or related to reliance by anyone on any guidance in this report or any contents thereof.

Copyrights & Trademarks

No claim is made to the exclusive right to use any trademarks or trade names found in this document. Red Lion disclaims responsibility for errors or omissions in typography or photography.

Red Lion Confidential

This document has been classified as Confidential. This is an internal Red Lion designation, which has the highest classification ranking for customer data. Stringent protection of this document is required by Red Lion's information classification policy and security controls.

This document should never be communicated from Red Lion to customers in an unencrypted format. Additionally, the information contained in this report is strictly prohibited from any type of release except to the customer.

Executive Summary

Redlion,LLc has been given the opportunity to review the smart contract source code of MobiePay. This report outlines our systematic approach to evaluate potential security issues in the smart contract implementation and deployment. Our aim is to expose possible semantic inconsistencies in the custom Solana code, and to provide suggestions or recommendations for improvements. After reviewing the given version of the smart contract code base committed to the private github repository May 12th, 2022 “on-chain-staking-contracts,” our results show that the given version can be further improved due to the presence of several issues. This document outlines the audit results.

About MobiPay

MobiePay is being built to be a universal payment ecosystem that allows users to instantly spend or exchange fiat and cryptocurrency directly from their mobile phone to merchants or other users. Mobie will reduce friction in the payment process and give users immediate cash back on purchases. Mobie will be a fast and convenient alternative to cash or plastic and will be integrated with over 500 brands at 225,000 retail locations and online sites. MobieCoin is designed as an integrated payment token that connects any fiat and cryptocurrency directly to the global retail marketplace. Mobie intends to bridge the gap between cryptocurrency and local currency enabling a revolutionary new payment standard.[1]

About Redlion

Red Lion, LLC offers a wide range of managed cybersecurity and compliance consulting services. We work alongside client security teams to identify gaps in security postures and minimize security threats. Boasting a resume of over 20 years of information security experience, when you work with Red Lion you'll have a highly experienced cyber security consultant by your side. Red Lion professionals are versatile and use real world experience to create custom solutions for organizations of all sizes.

Findings Overview

Phase 1 of our audit begins with studying the smart contract source code and noting coding comments for future reference. Coding comments are often left in standard libraries, and many times overlooked during development. There were many instances where standard library code comments warn developers to disable functions that could lead to a vulnerability.

Phase 2 of our audit involves creating sample tests that allow us to determine potential vulnerabilities such as re-entry, low-level calls, guard conditions and others.

The analysis of MobiePay's code revealed 15 findings of medium and 2 low severity:

Critical	High	Medium	Low	Total
0	0	15	2	17

Testing

Consideration #1
TokenStakingLP.sol
Check-effects-interaction: Potential violation of Checks-Effects-Interaction pattern in TokenStakingLP.deposit(uint256): Could potentially lead to re-entrancy vulnerability.
Code Lines: 108-126
Severity: Medium
Remediation Completed: ReentrancyGuard function added lib imported

```
function deposit(uint256 _amount) public whenNotPaused {
    // First, make sure staking is open, and the last reward block paid hasn't passed the
program last block
    require(programOpenBlock > 0, "Staking not open");
    require(block.number < programCloseBlock, "Staking closed");
    UserInfo storage user = userInfo[msg.sender];
    updatePool();
    uint256 pending =
user.amount.mul(accTokPerShare).div(ACC_TOKEN_PRECISION).sub(user.rewardDebt);
    if (pending > 0) {
      rewTok.mint(msg.sender, pending);
      emit Harvest(msg.sender, pending);
    }
    require(lastRewardBlock < programCloseBlock, "Staking not available");
    require(lpTok.allowance(msg.sender, address(this)) >= _amount, "No allowance");
    user.amount = user.amount.add(_amount);
    userTokStaked = userTokStaked.add(_amount); // Update the total tokens staked by
all users
    user.rewardDebt = user.amount.mul(accTokPerShare).div(ACC_TOKEN_PRECISION); //
Unearned rewards per share
    lpTok.safeTransferFrom(address(msg.sender), address(this), _amount); // Collect
tokens from user
    emit Deposit(msg.sender, _amount);
  }
```

Consideration #2

Code File: TokenStakingLP.sol
Check-effects-interaction: Potential violation of Checks-Effects-Interaction pattern in TokenStakingLP.withdraw(uint256 Could potentially lead to re-entrancy vulnerability.
Code Lines: 128-143
Severity: Medium
Remediation Completed: ReentrancyGuard function added lib imported

```
function withdraw(uint256 _amount) public whenNotPaused {
      UserInfo storage user = userInfo[msg.sender];
      require(user.amount >= _amount, "withdraw: not available");
      updatePool();
      // Harvest rewards
      uint256 pending =
user.amount.mul(accTokPerShare).div(ACC_TOKEN_PRECISION).sub(user.rewardDebt);
      if (pending > 0) {
         rewTok.mint(msg.sender, pending);
         emit Harvest(msg.sender, pending);
      }
      user.amount = user.amount.sub(_amount);
      userTokStaked = userTokStaked.sub(_amount); // Update the total tokens staked by all
users
      user.rewardDebt = user.amount.mul(accTokPerShare).div(ACC_TOKEN_PRECISION); //
Unearned rewards per share
      lpTok.safeTransfer(address(msg.sender), _amount);
      emit Withdraw(msg.sender, _amount);
  }
```

Consideration #3
Code File: TokenStakingLP.sol
Check-effects-interaction: Potential violation of Checks-Effects-Interaction pattern in TokenStakingLP.harvest(): Could potentially lead to re-entrancy vulnerability.
Code Lines: 159-168
Severity: Medium
Remediation Completed: ReentrancyGuard function added lib imported

```
  function harvest() public whenNotPaused {
      UserInfo storage user = userInfo[msg.sender];
      updatePool();
      uint256 pending =
user.amount.mul(accTokPerShare).div(ACC_TOKEN_PRECISION).sub(user.rewardDebt);
      require(pending > 0, "No rewards available.");
      rewTok.mint(msg.sender, pending);
      rewardsPaid = rewardsPaid.add(pending);
      user.rewardDebt = user.amount.mul(accTokPerShare).div(ACC_TOKEN_PRECISION);
      emit Harvest(msg.sender, pending);
  }
```

Consideration #4
Code File: TokenStakingLP.sol
Check-effects-interaction: Potential violation of Checks-Effects-Interaction pattern in TokenStakingLP.emergencyShutdown(): Could potentially lead to re-entrancy vulnerability.
Code Lines: 180-188
Severity: Medium
Remediation Completed: ReentrancyGuard function added lib imported

```
 function emergencyShutdown() public onlyOwner {
      lpTok.safeTransfer(address(msg.sender), lpTok.balanceOf(address(this)));
      userTokStaked = 0;
      accTokPerShare = 0;
      programOpenBlock = 0;
      programCloseBlock = 0;
      lastRewardBlock = 0;
      rewardsPaid = 0;
  }
```

Consideration #5
Code File: Address.sol
Low level calls: Use of "call": should be avoided whenever possible. It can lead to unexpected behavior if return values are not handled properly. Please use Direct Calls via specifying the called contract's interface.

Code Lines: 53-59
Severity: Medium
Remediation Completed: Address.sol was removed from TokenStaking.sol & TokenStakinLP.sol

```
function sendValue(address payable recipient, uint256 amount) internal {
    require(address(this).balance >= amount, "Address: insufficient balance");

    // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
    (bool success, ) = recipient.call{ value: amount }("");
    require(success, "Address: unable to send value, recipient may have reverted");
}
```

Consideration #6
Code File: Address.sol
Low level calls: Use of "call": should be avoided whenever possible. It can lead to unexpected behavior if return values arenot handled properly. Please use Direct Calls via specifying the called contract's interface.
Code Lines: 114-121
Severity: Medium
Remediation Completed: Address.sol was removed from TokenStaking.sol & TokenStakinLP.sol

```
function functionCallWithValue(address target, bytes memory data, uint256 value, string memory errorMessage) internal returns (bytes memory) {
    require(address(this).balance >= value, "Address: insufficient balance for call");
    require(isContract(target), "Address: call to non-contract");

    // solhint-disable-next-line avoid-low-level-calls
    (bool success, bytes memory returndata) = target.call{ value: value }(data);
    return _verifyCallResult(success, returndata, errorMessage);
}
```

Consideration #7
Code File: Address.sol

Low level calls: Use of "call": should be avoided whenever possible. It can lead to unexpected behavior if return values are not handled properly. Please use Direct Calls via specifying the called contract's interface.
Code Lines: 163-169
Severity: Medium
Remediation Completed: Address.sol was removed from TokenStaking.sol & TokenStakinLP.sol

```
function functionDelegateCall(address target, bytes memory data, string memory
errorMessage) internal returns (bytes memory) {
        require(isContract(target), "Address: delegate call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.delegatecall(data);
        return _verifyCallResult(success, returndata, errorMessage);
    }
```

Consideration #8
Code File: Address.sol
Inline assembly: The Contract uses inline assembly, this is only advised in rare cases. Additionally static analysis modules do not parse inline Assembly, this can lead to Incorrect analysis results.
Code Lines: 26-35
Severity: Medium
Remediation Completed: Address.sol was removed from TokenStaking.sol & TokenStakinLP.sol

```
function isContract(address account) internal view returns (bool) {
        // This method relies on extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.

        uint256 size;
        // solhint-disable-next-line no-inline-assembly
        assembly { size := extcodesize(account) }
        return size > 0;
```

```
  }
```

Consideration #9
Code File: Address.sol
Inline assembly: The Contract uses inline assembly, this is only advised in rare cases. Additionally static analysis modules do not parse inline Assembly, this can lead to Incorrect analysis results.
Code Lines: 171-189
Severity: Medium
Remediation Completed: Address.sol was removed from TokenStaking.sol & TokenStakinLP.sol

```
function _verifyCallResult(bool success, bytes memory returndata, string memory errorMessage) private pure returns(bytes memory) {
      if (success) {
         return returndata;
      } else {
         // Look for revert reason and bubble it up if present
         if (returndata.length > 0) {
            // The easiest way to bubble the revert reason is using memory via assembly

            // solhint-disable-next-line no-inline-assembly
            assembly {
               let returndata_size := mload(returndata)
               revert(add(32, returndata), returndata_size)
            }
         } else {
            revert(errorMessage);
         }
      }
   }
}
```

Consideration #10
Code File: TokenStaking.sol
Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in TokenStaking.deposit(uint256): Could potentially lead to re-entrancy vulnerability.
Code Lines: 101-114
Severity: Medium
Remediation Completed: ReentrancyGuard function added lib imported

```
 function deposit(uint256 _amount) public whenNotPaused {
      // First, make sure staking is open, and the last reward block paid hasn't passed the
program last block
      require(programOpenBlock > 0, "Staking not open");
      require(block.number < programCloseBlock, "Staking closed");
      UserInfo storage user = userInfo[msg.sender];
      updatePool();
      require(lastRewardBlock < programCloseBlock, "Staking not available");
      require(tok.allowance(msg.sender, address(this)) >= _amount, "No allowance");
      user.amount = user.amount.add(_amount);
      user.rewardDebt = user.amount.mul(accTokPerShare).div(ACC_TOKEN_PRECISION); //
Unearned rewards per share
      userTokStaked = userTokStaked.add(_amount); // Update the total tokens staked by
all users
      tok.transferFrom(address(msg.sender), address(this), _amount); // Collect tokens from
user
      emit Deposit(msg.sender, _amount);
   }
```

Consideration #11
Code File: TokenStaking.sol
Check-effects-interaction: Potential violation of Checks-Effects-Interaction pattern in TokenStaking.withdraw(uint256): Could potentially lead to re-entrancy vulnerability.
Code Lines: 116-125
Severity: Medium
Remediation Completed: ReentrancyGuard function added lib imported

```
function withdraw(uint256 _amount) public whenNotPaused {
      UserInfo storage user = userInfo[msg.sender];
      require(user.amount >= _amount, "withdraw: not available");
```

```
        updatePool();
        user.amount = user.amount.sub(_amount);
        userTokStaked = userTokStaked.sub(_amount); // Update the total tokens staked by all
users
        user.rewardDebt = user.amount.mul(accTokPerShare).div(ACC_TOKEN_PRECISION); //
Unearned rewards per share
        tok.transfer(address(msg.sender), _amount);
        emit Withdraw(msg.sender, _amount);
    }
```

Consideration #12
Code File: TokenStaking.sol
Check-effects-interaction: Potential violation of Checks-Effects-Interaction pattern in TokenStaking.withdraw(uint256): Could potentially lead to re-entrancy vulnerability.
Code Lines: 116-125
Severity: Medium
Remediation Completed: ReentrancyGuard function added lib imported

```
function harvest() public whenNotPaused {
        UserInfo storage user = userInfo[msg.sender];
        updatePool();
        uint256 pending =
user.amount.mul(accTokPerShare).div(ACC_TOKEN_PRECISION).sub(user.rewardDebt);
        require(pending > 0, "No rewards to reinvest.");
        tok.mint(msg.sender, pending);
        rewardsPaid = rewardsPaid.add(pending);
        user.rewardDebt = user.amount.mul(accTokPerShare).div(ACC_TOKEN_PRECISION);
        emit Harvest(msg.sender, pending);
    }
```

Consideration #13
Code File:TokenStaking.sol
Check-effects-interaction: Potential violation of Checks-Effects-Interaction pattern in TokenStaking.harvest(): Could potentially lead to re-entrancy vulnerability.
Code Lines: 141-150

Severity: Medium
Remediation Completed: ReentrancyGuard function added lib imported

```
function harvest() public whenNotPaused {
        UserInfo storage user = userInfo[msg.sender];
        updatePool();
        uint256 pending =
user.amount.mul(accTokPerShare).div(ACC_TOKEN_PRECISION).sub(user.rewardDebt);
        require(pending > 0, "No rewards to reinvest.");
        tok.mint(msg.sender, pending);
        rewardsPaid = rewardsPaid.add(pending);
        user.rewardDebt = user.amount.mul(accTokPerShare).div(ACC_TOKEN_PRECISION);
        emit Harvest(msg.sender, pending);
    }
```

Consideration #14
Code File: TokenStaking.sol
Check-effects-interaction: Potential violation of Checks-Effects-Interaction pattern in TokenStaking.compound(): Could potentially lead to re-entrancy vulnerability.
Code Lines: 152-166
Severity: Medium
Remediation Completed: ReentrancyGuard function added lib imported

```
function compound() public whenNotPaused {
        require(block.number < programCloseBlock, "Staking closed.");
        UserInfo storage user = userInfo[msg.sender];
        updatePool();
        require(lastRewardBlock < programCloseBlock, "Staking closed.");
        // Reinvest rewards
        uint256 pending =
user.amount.mul(accTokPerShare).div(ACC_TOKEN_PRECISION).sub(user.rewardDebt);
        require(pending > 0, "No rewards to reinvest.");
        tok.mint(address(this), pending); // Mint the rewards to this address.
        rewardsPaid = rewardsPaid.add(pending);
        userTokStaked = userTokStaked.add(pending);
        user.amount = user.amount.add(pending);
```

```
    user.rewardDebt = user.amount.mul(accTokPerShare).div(ACC_TOKEN_PRECISION);
    emit Compound(msg.sender, pending);
  }
```

Consideration #15
Code File: TokenStaking.sol
Check-effects-interaction: Potential violation of Checks-Effects-Interaction pattern in TokenStaking.emergencyShutdown(): Could potentially lead to re-entrancy vulnerability.
Code Lines: 178-186
Severity: Medium
Remediation Completed: ReentrancyGuard function added lib imported

```
function emergencyShutdown() public onlyOwner {
    tok.transfer(address(msg.sender), tok.balanceOf(address(this)));
    userTokStaked = 0;
    accTokPerShare = 0;
    programOpenBlock = 0;
    programCloseBlock = 0;
    lastRewardBlock = 0;
    rewardsPaid = 0;
  }
```

Consideration #16
Code File: Address.sol
Guard conditions: Use "assert(x)" if you never, ever want x to be false, in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to conditions such as an invalid input or a failing external component.
Code Lines: 53-59
Severity: Low
Remediation Completed: Address.sol was removed from TokenStaking.sol & TokenStakinLP.sol

```
function sendValue(address payable recipient, uint256 amount) internal {
    require(address(this).balance >= amount, "Address: insufficient balance");

    // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
    (bool success, ) = recipient.call{ value: amount }("");
    require(success, "Address: unable to send value, recipient may have reverted");
  }
```

Consideration #17
Code File: Address.sol
Guard conditions: Use "assert(x)" if you never ever want x to be false, in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to conditions such as an invalid input or a failing external component.
Code Lines: 114-21
Severity: Low
Remediation Completed: Address.sol was removed from TokenStaking.sol & TokenStakinLP.sol

```
function functionCallWithValue(address target, bytes memory data, uint256 value, string memory errorMessage) internal returns (bytes memory) {
    require(address(this).balance >= value, "Address: insufficient balance for call");
    require(isContract(target), "Address: call to non-contract");

    // solhint-disable-next-line avoid-low-level-calls
    (bool success, bytes memory returndata) = target.call{ value: value }(data);
    return _verifyCallResult(success, returndata, errorMessage);
  }
```

Supporting References

Re-Entrancy [3]

Any interaction from a contract (A) with another contract (B) and any transfer of Eth hands over control to that contract (B). This makes it possible for B to call back into A before this interaction is completed. To give an example, the following code contains a bug (it is just a snippet and not a complete contract):

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.6.0 <0.9.0;

// THIS CONTRACT CONTAINS A BUG - DO NOT USE
contract Fund {
    /// @dev Mapping of ether shares of the contract.
    mapping(address => uint) shares;
    /// Withdraw your share.
    function withdraw() public {
        if (payable(msg.sender).send(shares[msg.sender]))
            shares[msg.sender] = 0;
    }
}
```

Inline Assembly [4]

You can interleave Solidity statements with inline assembly in a language close to the language used for Ethereum virtual machines. This technique provides more fine-grained control, which is especially useful to enhance the language by writing libraries.

The language used for inline assembly in Solidity is called Yul and it is documented in its own section. This section will only cover how the inline assembly code can interface with the surrounding Solidity code.

Inline assembly is a way to access the Ethereum Virtual Machine at a low level. This bypasses several important safety features and checks of Solidity. You should only use it for tasks that need it, and only if you are confident with using it.

An inline assembly block is marked by assembly { ... }, where the code inside the curly braces is code in the Yul language.

The inline assembly code can access local Solidity variables as explained below.

Different inline assembly blocks share no namespace, i.e. it is not possible to call a Yul function or access a Yul variable defined in a different inline assembly block.

Low Level Calls [5]

If the return value of a low-level message call is not checked then the execution will resume even if the called contract throws an exception. If the call fails accidentally or an attacker forces the call to fail, then this may cause unexpected behavior in the subsequent program logic.

Guard Conditions [6]

The Solidity documentation suggests that require() "should be used to ensure valid conditions, such as inputs, or contract state variables [..], or to validate return values from calls to external contracts" and assert() "should only be used to test for internal errors, and to check invariants". Both methods evaluate the parameters passed to it as a boolean and throw an exception if it evaluates to false. The revert() throws in every case. It is therefore useful in complex situations, like if-else trees, where the evaluation of the condition can not be conducted in one line of code and the use of 'require()' would not be fitting.

References

[1] MobiPay White Paper

https://mobiepay.docsend.com/view/fcyi4yg6536jysyr

[2] Redlion, LLC About

https://redlion.io/about/

[3] Re-Entrancy

https://docs.soliditylang.org/en/v0.8.7/security-considerations.html#re-entrancy

[4] Assembly

https://docs.soliditylang.org/en/v0.8.7/assembly.html

[5] Low Level Calls

https://docs.guardrails.io/docs/vulnerabilities/solidity/use_of_low_level_call

[6] Guard Conditions

https://fravoll.github.io/solidity-patterns/guard_check.html