Smart Contract Security Audit Report

The SlowMist Security Team received the OneInch team's application for smart contract security audit of the OneInchExchange on Nov. 08, 2020. The following are the details and results of this smart contract security audit:

Project name :

OneInchExchange

File name and HASH(SHA256) :

OneInchExchange.sol:

d44acf84343b8a0dec2d5498226718617de04b7689afb6185687be78dc1286df

RevertReasonParser.sol:

b7836cb2e14cc88a8656ceb05ababc452376956c0e15f33ca1b4c50b8ea6cf3b

UniERC20.sol:

2c7ceb502077357a0f657217fa4e07d15bd875788af9faaaee3d523bfd852333

The audit items and results :

(Other unknown security vulnerabilities are not included in the audit responsibility scope)

No.	Audit Items	Audit Subclass	Audit Subclass Result
1	Overflow Audit	-	Passed
2	Race Conditions Audit	-	Passed
3	Authority Control Audit	Permission vulnerability audit	Passed
		Excessive authority audit	Passed
4	Safety Design Audit	Zeppelin module safe use	Passed
		Compiler version security	Passed
		Hard-coded address security	Passed
		Fallback function safe use	Passed
		Show coding security	Passed
		Function return value security	Passed
		Call function security	Passed
5	Denial of Service Audit	-	Passed

6	Gas Optimization Audit	-	Passed
7	Design Logic Audit	-	Low Risk
8	“False Deposit” vulnerability Audit	-	Passed
9	Malicious Event Log Audit	-	Passed
10	Scoping and Declarations Audit	-	Passed
11	Replay Attack Audit	ECDSA's Signature Replay Audit	Passed
12	Uninitialized Storage Pointers Audit	-	Passed
13	Arithmetic Accuracy Deviation Audit	-	Passed

Audit Result : **Passed**

Audit Number : 0X002011120001

Audit Date : Nov. 12, 2020

Audit Team : SlowMist Security Team

Summary: This is the DEX aggregator contract. The audit scope this time only include OneInchExchange contract, RevertReasonParser contract and UniERC20 contract. The contracts does not have the Overflow and the Race Conditions issue.

During the audit, the OneInch team confirmed that following issues are remain unchanged:

1. Users are at the risk of losing their tokens when they approve their tokens to OneInchCaller contract, which is used to forward arbitrary data to specific DEX

2. Users who use smart contract wallet may fail to call discountedSwap function after Istanbul

update.

After feeding back with the project party, the risk of issues listed above is within an acceptable range, following are the reasons:

1. For issue one, the OneInchExchangeCaller contract is not intented to have any ownership of any assets, it is helper contract to execute any possible calls.

2. For issue two, the OneInch team are not aware of any wallet implementations vulnerable to the transfer yet

The source code:

OneInchExchange.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.6.12;
pragma experimental ABIEncoderV2;

import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";
import "@openzeppelin/contracts/utils/Pausable.sol";
import "./interfaces/IChi.sol";
import "./interfaces/IERC20Permit.sol";
import "./interfaces/IOneInchCaller.sol";
import "./helpers/RevertReasonParser.sol";
import "./helpers/UniERC20.sol";


contract OneInchExchange is Ownable, Pausable {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;
    using UniERC20 for IERC20;

    uint256 private constant _PARTIAL_FILL = 0x01;
    uint256 private constant _REQUIRES_EXTRA_ETH = 0x02;
    uint256 private constant _SHOULD_CLAIM = 0x04;
    uint256 private constant _BURN_FROM_MSG_SENDER = 0x08;
    uint256 private constant _BURN_FROM_TX_ORIGIN = 0x10;
```

```
struct SwapDescription {
    IERC20 srcToken;
    IERC20 dstToken;
    address srcReceiver;
    address dstReceiver;
    uint256 amount;
    uint256 minReturnAmount;
    uint256 guaranteedAmount;
    uint256 flags;
    address referrer;
    bytes permit;
}

event Swapped(
    address indexed sender,
    IERC20 indexed srcToken,
    IERC20 indexed dstToken,
    address dstReceiver,
    uint256 amount,
    uint256 spentAmount,
    uint256 returnAmount,
    uint256 minReturnAmount,
    uint256 guaranteedAmount,
    address referrer
);

event Error(
    string reason
);

function discountedSwap(
    IOneInchCaller caller,
    SwapDescription calldata desc,
    bytes calldata data
)
    external
    payable
    returns (uint256 returnAmount)
{
    uint256 initialGas = gasleft();
```

```
        address chiSource = address(0);
        if (desc.flags & _BURN_FROM_MSG_SENDER != 0) {
            chiSource = msg.sender;
        } else if (desc.flags & _BURN_FROM_TX_ORIGIN != 0) {
            chiSource = tx.origin; // solhint-disable-line avoid-tx-origin
        } else {
            revert("Incorrect CHI burn flags");
        }

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returnData) = address(this).delegatecall(abi.encodeWithSelector(this.swap.selector,
caller, desc, data));
        if (success) {
            returnAmount = abi.decode(returnData, (uint256));
        } else {
            if (msg.value > 0) {
```

//SlowMist// After Istanbul update, users may fail to call discountedSwap function because of the transfer failure

```
                msg.sender.transfer(msg.value);
            }
            emit Error(RevertReasonParser.parse(returnData, "Swap failed: "));
        }

        (IChi chi, uint256 amount) = caller.calculateGas(initialGas.sub(gasleft()), desc.flags, msg.data.length);
        if (amount > 0) {
            chi.freeFromUpTo(chiSource, amount);
        }
    }

    function swap(
        IOneInchCaller caller,
        SwapDescription calldata desc,
        bytes calldata data
    )
        external
        payable
        whenNotPaused
        returns (uint256 returnAmount)
    {
        require(desc.minReturnAmount > 0, "Min return should not be 0");
```

```
        require(data.length > 0, "data should be not zero");

        uint256 flags = desc.flags;
        IERC20 srcToken = desc.srcToken;
        IERC20 dstToken = desc.dstToken;

        if (flags & _REQUIRES_EXTRA_ETH != 0) {
            require(msg.value > (srcToken.isETH() ? desc.amount : 0), "Invalid msg.value");
        } else {
            require(msg.value == (srcToken.isETH() ? desc.amount : 0), "Invalid msg.value");
        }

        if (flags & _SHOULD_CLAIM != 0) {
            require(!srcToken.isETH(), "Claim token is ETH");
            _claim(srcToken, desc.srcReceiver, desc.amount, desc.permit);
        }

        address dstReceiver = (desc.dstReceiver == address(0)) ? msg.sender : desc.dstReceiver;
        uint256 initialSrcBalance = (flags & _PARTIAL_FILL != 0) ? srcToken.uniBalanceOf(msg.sender) : 0;
        uint256 initialDstBalance = dstToken.uniBalanceOf(dstReceiver);

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory result) = address(caller).call{value:
msg.value}(abi.encodePacked(caller.callBytes.selector, data));
        if (!success) {
            revert(RevertReasonParser.parse(result, "OneInchCaller callBytes failed: "));
        }

        uint256 spentAmount = desc.amount;
        returnAmount = dstToken.uniBalanceOf(dstReceiver).sub(initialDstBalance);

        if (flags & _PARTIAL_FILL != 0) {
            spentAmount = initialSrcBalance.add(desc.amount).sub(srcToken.uniBalanceOf(msg.sender));
            require(returnAmount > 0, "Return amount is zero");
            require(returnAmount.mul(desc.amount) >= desc.minReturnAmount.mul(spentAmount), "Return amount is not
enough");
        } else {
            require(returnAmount >= desc.minReturnAmount, "Return amount is not enough");
        }

        _emitSwapped(desc, srcToken, dstToken, dstReceiver, spentAmount, returnAmount);
    }
```

```
    function _emitSwapped(
        SwapDescription calldata desc,
        IERC20 srcToken,
        IERC20 dstToken,
        address dstReceiver,
        uint256 spentAmount,
        uint256 returnAmount
     ) private {
        emit Swapped(
            msg.sender,
            srcToken,
            dstToken,
            dstReceiver,
            desc.amount,
            spentAmount,
            returnAmount,
            desc.minReturnAmount,
            desc.guaranteedAmount,
            desc.referrer
        );
    }

    function _claim(IERC20 token, address dst, uint256 amount, bytes calldata permit) private {
        // TODO: Is it safe to call permit on tokens without implemented permit? Fallback will be called. Is it bad for proxies?

        if (permit.length == 32 * 7) {
            // solhint-disable-next-line avoid-low-level-calls
            (bool success, bytes memory result) =
address(token).call(abi.encodeWithSelector(IERC20Permit.permit.selector, permit));
            if (!success) {
                string memory reason = RevertReasonParser.parse(result, "Permit call failed: ");
                if (token.allowance(msg.sender, address(this)) < amount) {
                    revert(reason);
                } else {
                    emit Error(reason);
                }
            }
        }

        token.safeTransferFrom(msg.sender, dst, amount);
    }
```

```
    //SlowMist// The owner can transfer any tokens to himself, users maybe worry about the function.
    function rescueFunds(IERC20 token, uint256 amount) external onlyOwner {
        token.uniTransfer(msg.sender, amount);
    }

    function pause() external onlyOwner {
        _pause();
    }
}
```

RevertReasonParser.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.6.12;



library RevertReasonParser {
    function parse(bytes memory data, string memory prefix) internal pure returns (string memory) {
        // https://solidity.readthedocs.io/en/latest/control-structures.html#revert
        // We assume that revert reason is abi-encoded as Error(string)

        // 68 = 4-byte selector 0x08c379a0 + 32 bytes offset + 32 bytes length
        if (data.length >= 68 && data[0] == "\x08" && data[1] == "\xc3" && data[2] == "\x79" && data[3] == "\xa0") {
            string memory reason;
            // solhint-disable no-inline-assembly
            assembly {
                // 68 = 32 bytes data length + 4-byte selector + 32 bytes offset
                reason := add(data, 68)
            }
            /*
                revert reason is padded up to 32 bytes with ABI encoder: Error(string)
                also sometimes there is extra 32 bytes of zeros padded in the end:
                https://github.com/ethereum/solidity/issues/10170
                because of that we can't check for equality and instead check
                that string length + extra 68 bytes is less than overall data length
            */
            require(data.length >= 68 + bytes(reason).length, "Invalid revert reason");
            return string(abi.encodePacked(prefix, "Error(", reason, ")"));
        }
```

```
        // 36 = 4-byte selector 0x4e487b71 + 32 bytes integer
        else if (data.length == 36 && data[0] == "\x4e" && data[1] == "\x48" && data[2] == "\x7b" && data[3] == "\x71") {
            uint256 code;
            // solhint-disable no-inline-assembly
            assembly {
                // 36 = 32 bytes data length + 4-byte selector
                code := mload(add(data, 36))
            }
            return string(abi.encodePacked(prefix, "Panic(", _toHex(code), ")"));
        }

        return string(abi.encodePacked(prefix, "Unknown()"));
    }

    function _toHex(uint256 value) private pure returns(string memory) {
        return _toHex(abi.encodePacked(value));
    }

    function _toHex(bytes memory data) private pure returns(string memory) {
        bytes memory alphabet = "0123456789abcdef";
        bytes memory str = new bytes(2 + data.length * 2);
        str[0] = "0";
        str[1] = "x";
        for (uint256 i = 0; i < data.length; i++) {
            str[2 * i + 2] = alphabet[uint8(data[i] >> 4)];
            str[2 * i + 3] = alphabet[uint8(data[i] & 0x0f)];
        }
        return string(str);
    }
}
```

UniERC20.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.6.12;

import "@openzeppelin/contracts/math/SafeMath.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";
```

```
library UniERC20 {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;

    IERC20 private constant _ETH_ADDRESS = IERC20(0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE);
    IERC20 private constant _ZERO_ADDRESS = IERC20(0);

    function isETH(IERC20 token) internal pure returns (bool) {
        return (token == _ZERO_ADDRESS || token == _ETH_ADDRESS);
    }

    function uniBalanceOf(IERC20 token, address account) internal view returns (uint256) {
        if (isETH(token)) {
            return account.balance;
        } else {
            return token.balanceOf(account);
        }
    }

    function uniTransfer(IERC20 token, address payable to, uint256 amount) internal {
        if (amount > 0) {
            if (isETH(token)) {
                to.transfer(amount);
            } else {
                token.safeTransfer(to, amount);
            }
        }
    }

    function uniApprove(IERC20 token, address to, uint256 amount) internal {
        require(!isETH(token), "Approve called on ETH");

        if (amount == 0) {
            token.safeApprove(to, 0);
        } else {
            uint256 allowance = token.allowance(address(this), to);
            if (allowance < amount) {
                if (allowance > 0) {
                    token.safeApprove(to, 0);
                }
                token.safeApprove(to, amount);
```

```
            }
        }
    }
}
```

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

Github
https://github.com/slowmist