SECURITY AUDIT OF JOYSO SMART CONTRACT

PUBLIC REPORT

MARCH 31, 2018

✓erichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology >> Forward

EXECUTIVE SUMMARY

This Security Audit Report prepared by Verichains Lab on March 31, 2018. We would like to thank Joyso to trust Verichains Lab to audit 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 smart contracts. The scope of the audit is limited to the source code files provided to Verichains Lab on Feb 23, 2018. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

The assessment identified some security issues in Joyso smart contracts code and issues have been fixed. Overall, the code reviewed is of good quality, written with the awareness of smart contract development best practices.

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.

CONTENTS

ACRONYMS AND ABBREVIATIONS

Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
ETH (Ether)	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.
EVM	Ethereum Virtual Machine.

AUDIT OVERVIEW

ABOUT JOYSO

Joyso is a hybrid exchange for token trading, combining the advantages of a centralized exchange -fast and fully-featured- with the advantages of a decentralized exchange -improved security and privacy. This hybrid exchange (HEX) eliminates the need to trust a central exchange with user's private key or personal information, reducing to a historical minimum any opportunities for hacking.

Joyso is designed with centralized matching and decentralized fund moving. Matching is done off-chain by the matching server to provide fast processing and one-to-many matches. On-chain settlement utilizes smart contracts to guarantee the funds moving with no access to the user's assets.

The website of Joyso is at https://joyso.io/
White paper (EN) is at https://joyso.io/whitepaper.pdf

SCOPE OF THE AUDIT

This audit focused on identifying security flaws in code and the design of the smart contracts. The scope of the audit is limited to the source code files provided to Verichains Lab on Feb 23, 2018.

- **contracts.zip**
  300f582be0123dd11fb4e12193530f5d4ea7c9fc1a53c04351bb4e859524dd13
  (SHA256)

Source File	SHA256 Hash
Joyso.sol	70270201dfe5773dd5bb8c84ae59e5465b9c68973bf0e38e0b99da5d2b4535a0
JoysoDataDecoder.sol	4afd3bdf7d851a8d4c32bcd46f8da1fdd3563a51262785b266bc6cf17c435803
libs/BasicToken.sol	e5db571af050f94bede9432680adc9516165f85e45a0cc78f93af15564534e04
libs/ERC20.sol	115176a18bd1532f76b6a577069621f3867b83dfb5859b0ff1af578f0cea2098
libs/ERC20Basic.sol	1355547bf20e382a9f76af542ecbd86edf175abd1eee6175b42615ade4a85383
libs/Ownable.sol	fd49860e2f11bc70dc265e0d001048eeed921900ad2e4c9599fb4e9a8e3cae5c
libs/SafeMath.sol	596bb5e82ff5009c31049f0cc9e5b8b95172fc7b38da5335bbdd09585c692f9c
libs/StandardToken.sol	94e90d7ec0debecbc262138fe1890cd7d46cadbf984b5b41fbdf001b015ac662

After the reported issues were fixed by Joyso, we have reviewed the fixes on commit *1b7986a* from private GitHub repository of Joyso (https://github.com/ConsensusInnovation/joyso-contracts/)

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 in-house automated analysis tools.
- 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, depending on their criticality:

LOW	An issue that does not have a significant impact, can be considered as less important
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.

AUDIT RESULT

VULNERABILITIES FINDINGS

✓ FIXED HIGH
INCORRECT BITWISE DATA DECODER IN JOYSODATADECODER

Joyso contract has complex data encoding and decoding logics. Some data portions are signed and verified before being used. There are incorrect bitwise data decoder problems inside JoysoDataDecoder.sol library while extracts binary parts of uint256 buffer using division.

```
    function decodeOrderNonce (uint256 data) public pure returns (uint256 nonce) {
        nonce = data / 0x00000000ffffffffffffffffffffffffffffffffffffffffffffffffffffffff;
    }
```

```
> decodeOrderNonce(0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff)
Returns: 0x100000000
Correct: 0xffffffff
```

Similar issue found in various places in JoysoDataDecoder class

```
    function decodeOrderTakerFee (uint256 data) public pure returns (uint256 takerFee) {
        data = data & 0x00000000ffffffffffffffffffffffffffffffffffffffffffffffffffffffff;
        takerFee = data / 0x000000000000ffffffffffffffffffffffffffffffffffffffffffffffffffff;
    }

    function decodeOrderMakerFee (uint256 data) public pure returns (uint256 makerFee) {
        data = data & 0x000000000000ffffffffffffffffffffffffffffffffffffffffffffffffffff;
        makerFee = data / 0x0000000000000000ffffffffffffffffffffffffffffffffffffffffffffffff;
    }

    function decodeOrderJoyPrice (uint256 data) public pure returns (uint256 joyPrice) {
        data = data & 0x0000000000000000ffffffffffffffffffffffffffffffffffffffffffffffff;
        joyPrice = data / 0x00000000000000000000000ffffffffffffffffffffffffffffffffffffffff;
    }

    function decodeOrderTokenIdAndIsBuy (uint256 data) public pure returns (uint256 tokenId,
uint256 isBuy) {
```

```
        data = data & 0x0000000000000000000000000000000000000000000000000fffffffffffffff;
        uint256 tokenSellId = data /
0x0000000000000000000000000000000000000000000000000fffffffffff;
        data = data & 0x0000000000000000000000000000000000000000000000000000fffffffffff;
        uint256 tokenBuyId = data /
0x0000000000000000000000000000000000000000000000000000000ffffffff;
        isBuy = 0;
        tokenId = tokenSellId;
        if (tokenSellId == 0) {
            tokenId = tokenBuyId;
            isBuy = ORDER_ISBUY;
        }
    }

    function decodeWithdrawData (uint256 _data) public pure returns (uint256 paymentMethod,
uint256 tokenID, uint256 userID) {
        /**
            data3
            0x000181bfeb 0000000000000 1 1 00000000000000000000000000 0002 00000001
            [ 0.. 7] (uint256) nonce         --> use for random hash
            [23..23] (uint256) paymentMethod --> 0: ether, 1: Token, 2: joyToken
            [52..55] (uint256) tokenID
            [56..63] (address) userID
            */
        // Assume the _data is come after retriveV, which already eliminated the first two
bytes.
        paymentMethod = _data &
0x00000000000000000000000f0000000000000000000000000000000000000000;
        _data = _data & 0x0000000000000000000000000000000000000000000000000000fffffffffff;
        tokenID = _data / 0x0000000000000000000000000000000000000000000000000000000000000000ffffffff;
        userID = _data & 0x00000000000000000000000000000000000000000000000000000000ffffffff;
    }
```

Recommended fixes

```
nonce = data / (0x00000000ffffffffffffffffffffffffffffffffffffffffffffffffffffffff + 1);
or
takerFee = data / 0x0000000000010000000000000000000000000000000000000000000000000000;
or
makerFee = data / 256**24;
```

✓ FIXED MEDIUM
UNRELIABLE METHOD OF MEASURING TIME USING A LARGE NUMBER OF BLOCKS

In order to withdraw from the contract, a user calls lockme() and then wait a period of locked time before he can call withdraw to get the fund from the contract.

```
uint256 public lockPeriod = 100000;

function withdraw (address token, uint256 amount) public {
    require(getBlock() > userLock[msg.sender] && userLock[msg.sender] != 0);
    require(balances[token][msg.sender] >= amount);
    balances[token][msg.sender] = balances[token][msg.sender].sub(amount);
    if(token == 0) {
        msg.sender.transfer(amount);
    } else {
        require(Token(token).transfer(msg.sender, amount));
    }
    Withdraw(token, msg.sender, amount, balances[token][msg.sender]);
}

function lockMe () public {
    userLock[msg.sender] = getBlock().add(lockPeriod);
    Lock (msg.sender, userLock[msg.sender]);
}

// ------------------------------------------ helper functions
function getBlock () public view returns (uint256) {
    return block.number;
}
```

The waiting time period is calculated based on the duration of 100,000 blocks. However, the time between blocks could be changed due to the difficulty bomb. There are EIPs such as Metropolis Difficulty Bomb Delay and there's also a potential for a change in block time when we have proof of stake. These changes will likely affect the expected time between blocks in the future and cause unexpected behaviors of Joyso (for example: a user might have to wait for a much longer or shorter time to withdraw fund from the contract).

We would not recommend using block.number for counting time for medium and long-term periods. Joyso should use timestamps instead of block numbers to check for the lock period. Please note that while block timestamps can be manipulated to a limited degree by the miners, it should not be an issue in Joyso use case.

✓ FIXED LOW
TYPO IN EVENT NAME TRADESCUESS

From the source code provided to us for audit, there is a typo in event name TradeScuess. It should be TradeSuccess instead.

```
    function processTakerOrder (uint256 gasFee, uint256 data, uint256 tokenExecute, uint256
etherExecute, uint256 isBuy, uint256 tokenId, bytes32 orderHash)
        internal
    {
        uint256 etherFee = calculateEtherFee(gasFee, data, etherExecute, orderHash, true);
        uint256 joyFee = calculateJoyFee(gasFee, data, etherExecute, orderHash, true);
        updateUserBalance(data, isBuy, etherExecute, tokenExecute, etherFee, joyFee,
tokenId);
        orderFills[orderHash] = orderFills[orderHash].add(tokenExecute);
        TradeScuess(userId2Address[decodeOrderUserId(data)], etherExecute, tokenExecute,
isBuy, etherFee, joyFee);
    }
```

RECOMMENDATIONS / SUGGESTIONS

- ✓ FIXED
  We recommend do not use "var" keyword to ensure strict typing within every scope. For examples:

  joyso-contracts/contracts/Joyso.sol
  Line 161 in 17b41a0

  ```
  161     var (paymentMethod, tokenId, userId) = decodeWithdrawData(inputs[2]);
  ```

  joyso-contracts/contracts/Joyso.sol
  Line 226 in 17b41a0

  ```
  226     var (tokenId, isBuy) = decodeOrderTokenIdAndIsBuy(inputs[3]);
  ```

  joyso-contracts/contracts/Joyso.sol
  Line 268 in 17b41a0

  ```
  268     var (nonce, paymentMethod, userId) = decodeCancelData(inputs[1]);
  ```

Should be changed to

```
uint256 tokenId, isBuy;
(tokenId, isBuy) = decodeOrderTokenIdAndIsBuy(inputs[3]);
```

- ✓ FIXED
Some internal functions (within "internal/private function" code section) such as *calculateJoyFee, calculateEtherFee, calculateEtherGet, calculateTokenGet* are public methods. While exposing these functions might not introduce a security risk, please double-checking the visibility of functions and state variables to make sure visibility like external, internal, private and public is used and defined properly.

  Call graphs of Joyso contract can be found in Appendix I

- As recent release of solidity has been introduced 'emit' keyword to use with events to differentiate them from functions. Please consider using "emit" prefix to invoke events to avoid confusions and typo/mistake.

- While the code below in depositToken function has no security issue, please consider moving the transferFrom action towards the end of the function to follow secure coding best practices.

```
    function depositToken (address token, uint256 amount) public {
        require(address2Id[token] != 0);
        addUser(msg.sender);
        require(Token(token).transferFrom(msg.sender, this, amount));
        balances[token][msg.sender] = balances[token][msg.sender].add(amount);
        Deposit(token, msg.sender, amount, balances[token][msg.sender]);
    }
```

- ✓ FIXED There is a potential abuse of registerToken function by admin by modifying the token address of an already registered token to another token. For example, an admin might change the

---

address of a low-value token to high-value token before calling the withdrawByAdmin. We recommend to not allow changing the address of the already registered tokens.

```
function registerToken (address tokenAddress, uint256 index) public onlyAdmin {
        require (index > 1);
        require (address2Id[tokenAddress] == 0);
        address2Id[tokenAddress] = index;
        tokenId2Address[index] = tokenAddress;
    }
```

- In some functions, parameters are accessed directly using array/byte offset with different meanings. This makes the code hard to read and high level of error-prone. While we know that there is a hard limit on the number of local variables, we recommend using variable renaming, packing data into structs and splitting apart functions whenever possible.

```
    function withdrawByAdmin (uint256[] inputs) onlyAdmin public {
        /**
            inputs[0] (uint256) amount;
            inputs[1] (uint256) gasFee;
            inputs[2] (uint256) dataV
            inputs[3] (bytes32) r
            inputs[4] (bytes32) s
            ---------------------------------
            dataV[0 .. 7] (uint256) nonce --> doesnt used when withdraw
            dataV[23..23] (uint256) paymentMethod --> 0: ether, 1: JOY, 2: token
            dataV[24..24] (uint256) v --> should be uint8 when used
            dataV[52..55] (uint256) tokenId
            dataV[56..63] (uint256) userId
```

- Solidity contracts can have a special form of comments that form the basis of the Ethereum Natural Specification Format. Please consider to change the comments inside Joyso smart contract following https://github.com/ethereum/wiki/wiki/Ethereum-Natural-Specification-Format.

CONCLUSION

Joyso smart contracts have been audited by Verichains Lab using various public and in-house analysis tools and intensively manual code review. The assessment identified some security issues in Joyso smart contracts code and issues have been fixed. Overall, the code reviewed is of good quality, written with the awareness of smart contract development best practices.

LIMITATIONS

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.

APPENDIX I

Figure 1 Call graph of Joyso.sol