WaykiChain Security Audit Report

Contents

1. Executive Summary

On Oct.12, 2020, the SlowMist security team received the WaykiChain team's security audit application for WaykiChain, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "black, grey box lead, white box assists" to conduct a complete security test on the project in the way closest to the real attack.

SlowMist blockchain system test method:

Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code module through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect wether there are vulnerabilities in programs suck as nodes, SDK, etc.

SlowMist blockchain risk level:

Critical vulnerabilities	Critical vulnerabilities will have a significant impact on the security of the blockchain, and it is strongly recommended to fix the critical vulnerabilities.
High-risk vulnerabilities	High-risk vulnerabilities will affect the normal operation of blockchain. It is strongly recommended to fix high-risk vulnerabilities.
Medium-risk vulnerablities	Medium vulnerability will affect the operation of blockchain. It is recommended to fix medium-risk vulnerabilities.

Low-risk vulnerabilities	Low-risk vulnerabilities may affect the operation of blockchain in certain scenarios. It is suggested that the project party should evaluate and consider whether these vulnerabilities need to be fixed.
Weaknesses	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.
Enhancement Suggestions	There are better practices for coding or architecture.

2. Project Background (Context)

2.1 Project Introduction

Project website: https://www.waykichain.com/

Project source code: https://github.com/WaykiChain/WaykiChain

Coin symbol: WICC

Initial Audit version: v3.2.1.1

Re Audit version:

https://github.com/WaykiChain/WaykiChain/commit/573a858a46a999e29651496f39e6a2cdf2db
a086

2.2 Scope of Audit

The main types of security audit include:

No.	Audit Category	Audit Result

1	Code Compliance Audit	PASS
2	Random Number Generation Algorithm Audit	PASS
3	Keystore Audit	PASS
4	Signature Verification Audit	PASS
5	Transaction Verification Audit	PASS
6	Encryption Strength Audit	PASS
7	Length Extension Attack Audit	PASS
8	Transaction Malleability Attack Audit	PASS
9	Transaction Replay Attack Audit	PASS
10	Block Verification Audit	PASS
11	Top-up Program Audit	PASS
12	RPC Permission Audit	PASS

(other unknown security vulnerabilities are not included in the scope of responsibility of this audit)

3. Code Overview

3.1 Infrastructure

WaykiChain developed based on the open source **Bitcoin v0.9.0**, using DPoS consensus algorithm, using both UXTO and Account/Balance model.

3.2 Code Compliance Audit

Fork open source blockchain source code will cause problems such as replay attacks and node peer pool pollution. We conduct relevant security compliance assessments for this.

- src/config/chainparams.h

```
typedef enum {
MAIN_NET = 0,
TEST_NET = 1,
REGTEST_NET = 2,
NULL_NETWORK_TYPE = 3
} NET_TYPE;
```

WaykiChain MainNet P2P protocol message structure’ s magic value is `0`, which is the different with Bitcoin, it will not cause node peer pool pollution.

3.3 Random Number Generation Algorithm Audit

When we run `getnewaddr` command to generating a new account or address, it will call the following function:

- src/rpc/rpcwallet.cpp

```
void CKey::MakeNewKey(bool fCompressedIn) {
RandAddSeedPerfmon();
do {
RAND_bytes(vch, sizeof(vch));
} while (!Check(vch));
fValid = true;
assert(fCompressedIn == true);
fCompressed = fCompressedIn;
}
```

`RAND_bytes` is security, the bytes will be pseudo-random (but still cryptographically strong).

RAND_bytes returns 1 for success, and 0 otherwise. If you changed the RAND_METHOD and it is not supported, then the function will return -1. In case of error, you can call ERR_get_error.

Reference: 《An Analysis of OpenSSL’ s Random Number Generator》

https://eprint.iacr.org/2016/367.pdf

3.4 Keystore Audit

The wallet password strength has not been verified. Weak passwords such as `123456` can be used in the test, which can be easily crack.

3.5 Signature Verification Audit

Signature algorithm: secp256k1

Hash algorithm: SHA256

- WaykiChain/src/entities/key.cpp

```
bool CPubKey::Verify(const uint256 &hash, const vector<uint8_t> &vchSig) const {
if (!IsValid())
return false;

secp256k1_pubkey pubkey;
secp256k1_ecdsa_signature sig;
if (!secp256k1_ec_pubkey_parse(secp256k1_context_verify, &pubkey, vch, size())) {
return false;
}
if (!ecdsa_signature_parse_der_lax(secp256k1_context_verify, &sig, vchSig.data(), vchSig.size())) {
return false;
}
/* libsecp256k1's ECDSA verification requires lower-S signatures, which have
* not historically been enforced in Bitcoin, so normalize them first. */
secp256k1_ecdsa_signature_normalize(secp256k1_context_verify, &sig, &sig);
return secp256k1_ecdsa_verify(secp256k1_context_verify, &sig, hash.begin(), &pubkey);
}
```

No error calls were found.

3.6 Transaction Verification Audit

The transaction struct is as follows:

- WaykiChain/src/tx/tx.h

```
class CBaseTx {
public:
static const int32_t CURRENT_VERSION = INIT_TX_VERSION;

int32_t nVersion;
TxType nTxType;
mutable CUserID txUid;
int32_t valid_height;
TokenSymbol fee_symbol; // fee symbol, default is WICC, some tx (MAJOR_VER_R1) not serialize this field
uint64_t llFees;
UnsignedCharArray signature;

/*************** Memory Cache Only ******************/
CTxCord txCord;
uint64_t fuel = 0;
int32_t nFuelRate = 0;
mutable TxID sigHash;
map< CKeyID, std::shared_ptr<CAccount> > account_map;
std::shared_ptr<CAccount> sp_tx_account = nullptr;

ReceiptList receipts; //!< not persisted within Tx Cache
```

//...code snip...

```
bool CheckBaseTx(CTxExecuteContext &context);
virtual bool CheckTx(CTxExecuteContext &context) = 0;
virtual bool ExecuteTx(CTxExecuteContext &context) = 0;
bool ExecuteFullTx(CTxExecuteContext &context);

inline bool CheckAndExecuteTx(CTxExecuteContext& context) {
return CheckBaseTx(context) && CheckTx(context) && ExecuteFullTx(context);
}

bool IsValidHeight(int32_t nCurHeight, int32_t nTxCacheHeight) const;

bool IsBlockRewardTx() { return nTxType == BLOCK_REWARD_TX || nTxType == UCOIN_BLOCK_REWARD_TX; }
bool IsPriceMedianTx() { return nTxType == PRICE_MEDIAN_TX; }
bool IsPriceFeedTx() { return nTxType == PRICE_FEED_TX; }
bool IsCoinMintTx() { return nTxType == UCOIN_MINT_TX; }
bool IsRelayForbidden() { return kForbidRelayTxSet.count(nTxType) > 0; }
```

All transaction fields have been correctly verified.

3.7 Length Extension Attack Audit

In cryptography and computer security, a length extension attack is a type of attack where an attacker can use Hash(message1) and the length of message1 to calculate Hash(message1 ‖ message2) for an attacker-controlled message2, without needing to know the content of message1. Algorithms like MD5, SHA-1, and SHA-2 that are based on the Merkle–Damgard construction are susceptible to this kind of attack. The SHA-3 algorithm is not susceptible.

No error calls were found.

3.8 Transaction Malleability Attack Audit

Bip-66/Bip-62 had solve transaction malleability attack in Bitcoin, but WaykiChain forked an older version of Bitcoin, so we check it again.

We found that waykichain's utxo model does not include Bitcoin script input, and there is no transaction malleability problem.

Vulnerability reference: https://en.bitcoinwiki.org/wiki/Transaction_Malleability

3.9 Transaction Replay Attack Audit

- WaykiChain/src/tx/txmempool.cpp

```
// already confirmed in block
```

```
if (cw->txCache.HasTx(txid)) {
LogPrint(BCLog::INFO, "txid %s confirmed in block\n", txid.GetHex());
return state.Invalid(false, REJECT_INVALID, "tx-duplicate-confirmed");
}
```

`CheckTxInMemPool` function check whether transaction is duplicated include in block.

3.10 Block Verification Audit

The block header structure is as follows:

- WaykiChain/src/persistence/block.h

```
class CBlockHeader {
public:
// header
static const int32_t CURRENT_VERSION = INIT_BLOCK_VERSION;

protected:
int32_t nVersion;
uint256 prevBlockHash;
uint256 merkleRootHash;
uint32_t nTime;
uint32_t nNonce;
uint32_t height;
uint64_t nFuelFee;
uint32_t nFuelRate;
vector<unsigned char> vSignature;
```

Compute block hash:

- WaykiChain/src/persistence/block.cpp

```
uint256 CBlockHeader::ComputeSignatureHash() const {
CHashWriter ss(SER_GETHASH, CLIENT_VERSION);
ss << nVersion << prevBlockHash << merkleRootHash << nTime << nNonce << height << nFuelFee << nFuelRate;
return ss.GetHash();
}
```

All block header fields are included when compute signature hash.

Check block fields when processing block:

- WaykiChain/src/main.h

```
/** Functions for validating blocks and updating the block tree */

/** Undo the effects of this block (with given index) on the UTXO set represented by coins.
* In case pfClean is provided, operation will try to be tolerant about errors, and *pfClean
* will be true if no problems were found. Otherwise, the return value will be false in case
* of problems. Note that in any case, coins may be modified. */
bool DisconnectBlock(CBlock &block, CCacheWrapper &cw, CBlockIndex *pIndex, CValidationState &state,
bool *pfClean = nullptr);
// Apply the effects of this block (with given index) on the UTXO set represented by coins
bool ConnectBlock (CBlock &block, CCacheWrapper &cw, CBlockIndex *pIndex, CValidationState &state, bool
fJustCheck = false);

// Add this block to the block index, and if necessary, switch the active block chain to this
bool AddToBlockIndex(CBlock &block, CValidationState &state, const CDiskBlockPos &pos);

// Context-independent validity checks
bool CheckBlock(const CBlock &block, CValidationState &state, CCacheWrapper &cw,
bool fCheckTx = true, bool fCheckMerkleRoot = true);

bool ProcessForkedChain(const CBlock &block, CValidationState &state);

// Store block on disk
// if dbp is provided, the file is known to already reside on disk
bool AcceptBlock(CBlock &block, CValidationState &state, CDiskBlockPos *dbp = nullptr, bool mining =
false);

//disconnect block for test
bool DisconnectTip(CValidationState &state);

/** Mark a block as invalid. */
bool InvalidateBlock(CValidationState &state, CBlockIndex *pIndex);
```

```
/** Import blocks from an external file */
bool LoadExternalBlockFile(FILE *fileIn, CDiskBlockPos *dbp = nullptr);
/** Initialize a new block tree database + block data on disk */
bool InitBlockIndex();
/** Load the block tree and coins database from disk */
bool LoadBlockIndex();
/** Unload database information */
void UnloadBlockIndex();
/** Push getblocks request */
void PushGetBlocks(CNode *pNode, CBlockIndex *pindexBegin, uint256 hashEnd);
/** Push getblocks request with different filtering strategies */
void PushGetBlocksOnCondition(CNode *pNode, CBlockIndex *pindexBegin, uint256 hashEnd);
/** Process an incoming block */
bool ProcessBlock(CValidationState &state, CNode *pFrom, CBlock *pBlock, CDiskBlockPos *dbp = nullptr);
/** Print the loaded block tree */
void PrintBlockTree();

void UpdateTime(CBlockHeader &block, const CBlockIndex *pIndexPrev);

/** Find the best known block, and make it the tip of the block chain */
bool ActivateBestChain(CValidationState &state, CBlockIndex* pNewIndex = nullptr);

/** Remove invalidity status from a block and its descendants. */
bool ReconsiderBlock(CValidationState &state, CBlockIndex *pIndex, bool children);
```

All fields in the block header have been correctly verified.

3.11 Consensus and failsafes Audit

WaykiChain uses DPoS+pBFT consensus algorithms to balance overall performance and consensus efficiency. A total of 11 supernodes are set up by voting, and a new block is generated every 3 seconds.

Too few block producer nodes, with centralization risk.

3.12 False top-up Attack Audit

There are many different types of transactions. In addition to checking the amount when depositing, the exchange also need to check the transaction type.

- WaykiChain/src/config/txbase.h

```
enum TxType: uint8_t {
NULL_TX = 0, //!< NULL_TX

/** R1 Tx types */
BLOCK_REWARD_TX = 1, //!< Miner Block Reward Tx or Genesis Mint Reward (backward compatbility)
ACCOUNT_REGISTER_TX = 2, //!< Account Registration Tx
BCOIN_TRANSFER_TX = 3, //!< BaseCoin Transfer Tx
LCONTRACT_INVOKE_TX = 4, //!< LuaVM Contract Invocation Tx
LCONTRACT_DEPLOY_TX = 5, //!< LuaVM Contract Deployment Tx
DELEGATE_VOTE_TX = 6, //!< Vote Delegate Tx

/** R2 newly added Tx types below */
UCOIN_STAKE_TX = 8, //!< Stake Fund Coin Tx in order to become a price feeder

ASSET_ISSUE_TX = 9, //!< a user issues onchain asset
UIA_UPDATE_TX = 10, //!< a user update onchain asset

UCOIN_TRANSFER_TX = 11, //!< Universal Coin Transfer Tx
UCOIN_MINT_TX = 12, //!< Universal Coin Mint Tx
UCOIN_BLOCK_REWARD_TX = 13, //!< Universal Coin Miner Block Reward Tx
UCONTRACT_DEPLOY_TX = 14, //!< universal VM contract deployment, @@Deprecated
UCONTRACT_INVOKE_TX = 15, //!< universal VM contract invocation, @@Deprecated
PRICE_FEED_TX = 16, //!< Price Feed Tx: WICC/USD | WGRT/USD | WUSD/USD
PRICE_MEDIAN_TX = 17, //!< Price Median Value on each block Tx
UTXO_TRANSFER_TX = 18, //!< UTXO & HTLC Coin
UTXO_PASSWORD_PROOF_TX = 19, //!< UTXO password proof

CDP_STAKE_TX = 21, //!< CDP Staking/Restaking Tx
CDP_REDEEM_TX = 22, //!< CDP Redemption Tx (partial or full)
CDP_LIQUIDATE_TX = 23, //!< CDP Liquidation Tx (partial or full)
CDP_FORCE_SETTLE_INTEREST_TX= 24, //!< CDP Settle Interst Tx

ACCOUNT_PERMS_CLEAR_TX = 50, //!< Self removal of one's perms

PROPOSAL_REQUEST_TX = 70,
PROPOSAL_APPROVAL_TX = 71,

/** deprecated below for backward compatibility **/
DEX_LIMIT_BUY_ORDER_TX = 84, //!< dex buy limit price order Tx
DEX_LIMIT_SELL_ORDER_TX = 85, //!< dex sell limit price order Tx
DEX_MARKET_BUY_ORDER_TX = 86, //!< dex buy market price order Tx
DEX_MARKET_SELL_ORDER_TX = 87, //!< dex sell market price order Tx

/** active order tx types **/
DEX_CANCEL_ORDER_TX = 88, //!< dex cancel order Tx
DEX_TRADE_SETTLE_TX = 89, //!< dex settle Tx
DEX_ORDER_TX = 90, //!< dex common order tx, support BUY|SELL LIMIR|MARKET order
DEX_OPERATOR_ORDER_TX = 91, //!< dex operator order tx, need dex operator signing
DEX_OPERATOR_REGISTER_TX = 92, //!< dex operator register tx
DEX_OPERATOR_UPDATE_TX = 93, //!< dex operator update tx

/** unified tx for all future on-chain interactions **/
UNIVERSAL_TX = 100, //!< universal system or user contract deployment/invocation

/////////////////////<< NO MORE TX TYPES AFTER THIS LINE >>/////////////////////////////
};
```

3.13 RPC Permission Audit

RPC has wallet function, the node keep RPC port open in WAN by default, it’s not secure,

- .WaykiChain/WaykiChain.conf

```
#nettype=main|test|regtest
nettype=main
rpcserver=1
rpcallowip=0.0.0.0/0
```

```
rpcport=6968
rpcuser=wiccuser
rpcpassword=wicc1000
rpcthreads=8
logprinttoconsole=0
logprinttofile=1
logprintfileline=1
logtimestamps=1
listen=1
disablesafemode=1
genblock=0
genblocklimit=1000000
```

Vulnerability reference: https://mp.weixin.qq.com/s/Kk2lsoQ1679Gda56Ec-zJg

4. Audit Result

4.1 Low-risk vulnerabilities

- RPC port open in WAN by default configuration, hacker can steal wallet by RPC.
- `RAND_bytes` call failed will make the private key predictable. [fixed in re-audit version]

4.2 Enhancement Suggestions

- Add more block producer nodes, reduce centralization risk.
- Check return value when calling `RAND_bytes`, and abort the program when an error occurs.

4.3 Exchange Security Summary

- Recharge into the account when the transaction is confirmed to be irreversible.
- There are multiple kinds of token, and the token name and type need to be checked.
- Keep RPC port closed, or do not open in WAN.

4.4 Conclusion

Audit result: PASS

Audit No. : BCA002010230001

Audit date: October 23, 2020

Audit team: SlowMist security team

Summary conclusion: After correction, all problems found have been fixed and the above risks have been eliminated by WaykiChain. Comprehensive assessed, WaykiChain has no risks above already.

5. Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility base on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance this report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.