BEOSIN
Blockchain Security

Public Blockchain Security Audit Report

Audit Number：202109221821

Public Blockchain Name： QITCOIN

Source Code Link： https://github.com/qitchain/qitcoin

Initial commit： fd519d9d598da3e385a2ca14b47256d228d24900

Final commit: 5db1aa996dc019c378d504c341f31da51cb9d415

Start Date： 2021.09.06

Completion Date： 2021.09.22

Overall Result：Pass (Distinction)

Audit Team：Beosin Technology Co. Ltd.

Audit Categories and Results:

No.	**Categories**	**Subitems**	**Results**
1	Language Coding Security Audit	Language Feature Security	Pass
		Arithmetic Operation	Pass
2	RPC Security Audit	RPC Sensitive Interface Permissions	Pass
		Traditional Web Security	Pass
		RPC Interface Security	Pass
3	Wallet Module & Account Security Audit	Private Key / Mnemonic Word Storage Security	Pass
		Private Key / Mnemonic Word Usage Security	Pass
		Wallet Module Code Security	Pass
4	Transaction Model Security Audit	Double-spending Attack	Pass
		Transaction Replay	Pass
		Transaction Processing Logic	Pass
5	Consensus Security	Design Of Consensus Mechanism	Pass
		Implementation Of Consensus Verification	Pass
		Incentive Mechanism Audit	Pass
6	Third-party Library Vulnerability Detection	Dependent library Security Audit	Pass

Disclaimer: This report is made in response to the project code. No description, expression or wording in this report shall be construed as an endorsement, affirmation or confirmation of the project. This audit is only applied to the type of auditing specified in this report and the scope of given in the results table. Other unknown security vulnerabilities are beyond auditing responsibility. Beosin Technology only issues this report based on the attacks or vulnerabilities that already existed or occurred before the issuance of this report. For the emergence of new attacks or vulnerabilities that exist or occur in the future, Beosin Technology lacks the capability to judge its possible impact on the security status of block chain, thus taking no responsibility for them. The security audit analysis and other contents of this report are based solely on the documents and materials that the block chain provider has provided to Beosin Technology before the issuance of this report, and the block chain provider warrants that there are no missing, tampered, deleted; if the documents and materials provided by the block chain provider are missing, tampered, deleted, concealed or reflected in a situation that is inconsistent with the actual situation, or if the documents and materials provided are changed after the issuance of this report, Beosin Technology assumes no responsibility for the resulting loss or adverse effects. The audit report issued by Beosin Technology is based on the documents and materials provided by the block chain provider, and relies on the technology currently possessed by Beosin. Due to the technical limitations of any organization, this report conducted by Beosin still has the possibility that the entire risk cannot be completely detected. Beosin disclaims any liability for the resulting losses.

The final interpretation of this statement belongs to Beosin.

Basic information of the chain:

Name	QITCOIN
Symbol	QTC
Decimals	8
Block Size	2MB
Block Generation Time	3 min
Total Supply	105 Million
Initial Block Reward	75 QTC per block
Block Reward Halve Period	The first halving time is 420,000 block height, and will be halved every 700,000 blocks later, and the maximum number of halving is 64 times
Consensus Algorithm	Conditioned-Proof of Capacity, CPoC
Token distribution	20% (21000000) is allocated to the preset address, and the remaining 80% to the mining rewards and POS income. According to the project owner, the QITCOIN main chain will use the pre-allocated tokens respectively for the Foundation (5%) and Search Lab (15%). The on-chain data shows that the Foundation address (3BvLuBGM4JoAhBEbiuRK8hjDk2VCFQEGgu) has about 5.25 million tokens and has not been released; the Search Lab address (3LPUjNd6j48NiWL7J8c2bZZ8rv45u2f2TY) initially has 15.75 million, and 3.15 million have been released linearly as of September 15, 2021.

Table 1 Basic information of the QITCOIN

Detailed explanations of the Audit Results:

1. Language coding security audit

Through static scanning of the public chain source code and manual analysis of dangerous functions, no major security vulnerabilities were found.

(1) Memory leak

Description: As shown in the figure below, walletInstance is not explicitly released in some branches, and system resources will not be recycled and reused, thereby reducing the future availability of resources. However, the program will exit shortly after entering this branch and has less impact.

```
std::shared_ptr<CWallet> CWallet::CreateWalletFromFile(interfaces::Chain& chain, const WalletLocation& location, uint64_t wallet_creation_flags)
{
    const std::string walletFile = WalletDataFilePath(location.GetPath()).string();

    // needed to restore wallet transaction meta data after -zapwallettxes
    std::vector<CWalletTx> vWtx;

    if (gArgs.GetBoolArg("-zapwallettxes", false)) {
        chain.initMessage(_("Zapping all transactions from wallet...").translated);

        std::unique_ptr<CWallet> tempWallet = MakeUnique<CWallet>(&chain, location, WalletDatabase::Create(location.GetPath()));
        DBErrors nZapWalletRet = tempWallet->ZapWalletTx(vWtx);
        if (nZapWalletRet != DBErrors::LOAD_OK) {
            chain.initError(strprintf(_("Error loading %s: Wallet corrupted").translated, walletFile));
            return nullptr;
        }
    }

    chain.initMessage(_("Loading wallet...").translated);

    int64_t nStart = GetTimeMillis();
    bool fFirstRun = true;
    // TODO: Can't use std::make_shared because we need a custom deleter but
    // should be possible to use std::allocate_shared.
    std::shared_ptr<CWallet> walletInstance(new CWallet(&chain, location, WalletDatabase::Create(location.GetPath())), ReleaseWallet);
    DBErrors nLoadWalletRet = walletInstance->LoadWallet(fFirstRun);
    if (nLoadWalletRet != DBErrors::LOAD_OK)
    {
        if (nLoadWalletRet == DBErrors::CORRUPT) {
            chain.initError(strprintf(_("Error loading %s: Wallet corrupted").translated, walletFile));
            return nullptr;
        }
```

Figure 1 Pointers that are not explicitly released

Suggestion: Use smart pointers such as std::shared_ptr<CWallet> to manage the instance (walletInstance).

(2) Null pointer reference

Description: As shown in the figure below, frand may be null during execution, and according to the code logic, it is normal for frand to be empty, but then fclose on the null pointer will cause an abnormal crash. But because the code is in the test code of secp256k1, the impact is very small.

```
    }
    } else {
        FILE *frand = fopen("/dev/urandom", "r");
        if ((frand == NULL) || fread(&seed16, 1, sizeof(seed16), frand) != sizeof(seed16)) {
            uint64_t t = time(NULL) * (uint64_t)1337;
            fprintf(stderr, "WARNING: could not read 16 bytes from /dev/urandom; falling back to insecure PRNG\n");
            seed16[0] ^= t;
            seed16[1] ^= t >> 8;
            seed16[2] ^= t >> 16;
            seed16[3] ^= t >> 24;
            seed16[4] ^= t >> 32;
            seed16[5] ^= t >> 40;
            seed16[6] ^= t >> 48;
            seed16[7] ^= t >> 56;
        }
        if (frand) {
            fclose(frand);
        }
```

Figure 2 Null pointer reference

Suggestion: Before fclose, judge whether frand is NULL.

(3) Use variables that are not explicitly initialized

Description: As shown in the figure below, during the execution of the constructor of the Stats class, the uninitialized start_ is set to last_op_finish_. This will cause the value of last_op_finish_ to be unexpected, which has certain safety risks. However, this problem also exists in the latest BTC source code. And the impact and scope are not large.

```
class Stats {
 private:
  double start_;
  double finish_;
  double seconds_;
  int done_;
  int next_report_;
  int64_t bytes_;
  double last_op_finish_;
  Histogram hist_;
  std::string message_;

 public:
  Stats() { Start(); }

  void Start() {
    next_report_ = 100;
    last_op_finish_ = start_;
    hist_.Clear();
    done_ = 0;
    bytes_ = 0;
    seconds_ = 0;
    start_ = g_env->NowMicros();
    finish_ = start_;
    message_.clear();
  }
```

Figure 3 Variables that are not explicitly initialized

Suggestion: When defining the class, set the initial value of start_ to 0.0.

2. RPC Security Audit

(1) rpc request permission control

Like Bitcoin, if users need to open http RPC interface, manually opening it with –server parameter and setting the rpcuser & rpcpassword are required. Each time when calling RPC interface, the authentication is required. If failed in check, this calling will be failed.

(2) rpc interface security

- Node crash caused by the assert keyword

As shown in the figure below, when the getplottermininginfo and getactivebindplotteraddress interfaces are called to obtain data in the cli, the following errors will appear:

```
[redacted] qitcoin-cli getplottermininginfo "18229143786996629828"
error: Could not connect to the server 127.0.0.1:13343

Make sure the qitcoind server is running and that you are connecting to the correct RPC port.
```

Figure 4 The result of calling the getplottermininginfo interface

The corresponding node will also crash accordingly:

```
qitcoind: coins.cpp:493: const Coin& CCoinsViewCache::GetLastBindPlotterCoin(const uint64_t&, COutPoint*) const: Assertion `!coin.IsSpent()' failed.
Aborted (core dumped)
```

Figure 5 Node information

The relevant codes involved are as follows:

```
const Coin& CCoinsViewCache::GetLastBindPlotterCoin(const uint64_t &plotterId, COutPoint *outpoint) const {
    CBindPlotterInfo lastBindInfo = GetLastBindPlotterInfo(plotterId);
    if (outpoint) *outpoint = lastBindInfo.outpoint;

    const Coin& coin = AccessCoin(lastBindInfo.outpoint);
    assert(!coin.IsSpent());
    assert(coin.IsBindPlotter());
    assert(BindPlotterPayload::As(coin.payload)->GetId() == plotterId);
    return coin;
}
```

Figure 6 The relevant codes

In addition, the addsignprivkey interface does not check the validity of the added private key. If the private key passed in is an invalid private key, it will also cause the node to crash.

```
[redacted] qitcoin-cli addsignprivkey "Invalid key"
error: Could not connect to the server 127.0.0.1:13343

Make sure the qitcoind server is running and that you are connecting to the correct RPC port.
```

Figure 7 The result of calling the addsignprivkey interface

```
qitcoind: key.cpp:185: CPubKey CKey::GetPubKey() const: Assertion `fValid' failed.
Aborted (core dumped)
```

Figure 8 Node information

The relevant codes involved are as follows:

```
CPubKey CKey::GetPubKey() const {
    assert(fValid);
    secp256k1_pubkey pubkey;
    size_t clen = CPubKey::PUBLIC_KEY_SIZE;
    CPubKey result;
    int ret = secp256k1_ec_pubkey_create(secp256k1_context_sign, &pubkey, begin());
    assert(ret);
    secp256k1_ec_pubkey_serialize(secp256k1_context_sign, (unsigned char*)result.begin(), &clen, &pubkey, fCompressed ?
    assert(result.size() == clen);
    assert(result.IsValid());
    return result;
```

Figure 9 The relevant codes

Suggestion: Check the use of assert in the project to avoid the termination of the main program caused by incorrect parameters.

Fixed Result: Fixed.

- Wrong help message

As shown in the figure below, the help message of the decodebindplotterdata interface prompts that two parameters need to be passed in: address and hexdata, but in fact only one parameter hexdata needs to be passed in.

```
[redacted]; qitcoin-cli help decodebindplotterdata
decodebindplotterdata "address" "hexdata"

Decode bind plotter hex data.

Arguments:
1. hexdata              (string, required) The bind hex data

Result:
[
  {
    "plotterId":"plotterId",              (string) The binded plotter ID.
    "lastActiveHeight":lastActiveHeight,  (numeric) The bind last active height for tx package.
    "pubkey":"publickeyhex",              (string) The public key.
    "signature":"signaturehex"            (string) The signature.
  }
]

Examples:
```

Figure 10 Help message of the decodebindplotterdata interface

Suggestion: Modify the help message.

Fixed Result: Fixed.

3. Wallet Module & Account Security Audit

The same as Bitcoin, QITCOIN use UTXO model, and support encrypt/decrypt interface to encrypt/decrypt local wallet. Manual lock & unlock wallet are supported. The local wallet files are stored encrypted.

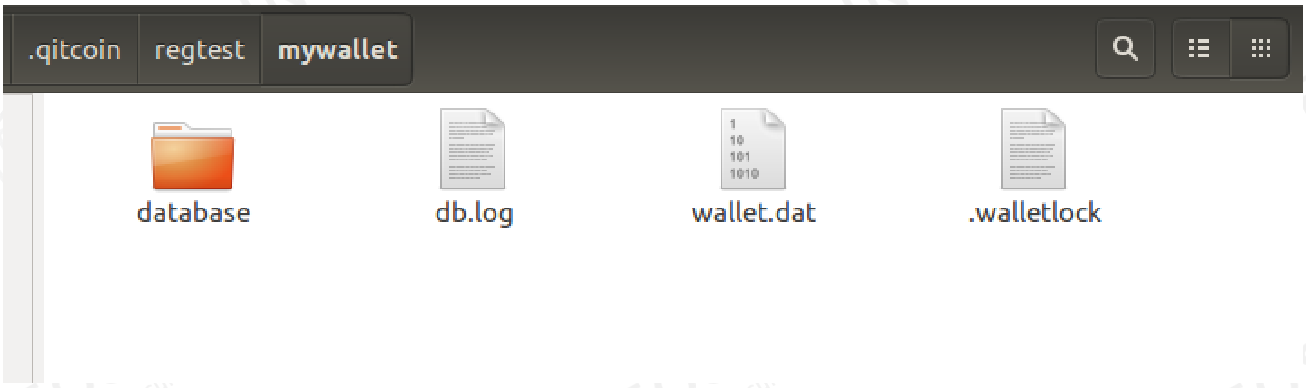

Figure 11 local wallet file

In addition, according to QTICOIN's consensus mechanism, miners need to meet a certain pointReceived before they can get a high percentage of miner rewards; at the same time, the top 10 on the staking-list can get mining dividends based on the amount of staking. QITCOIN provides related interfaces for this function for users to operate QTC assets to obtain higher returns.

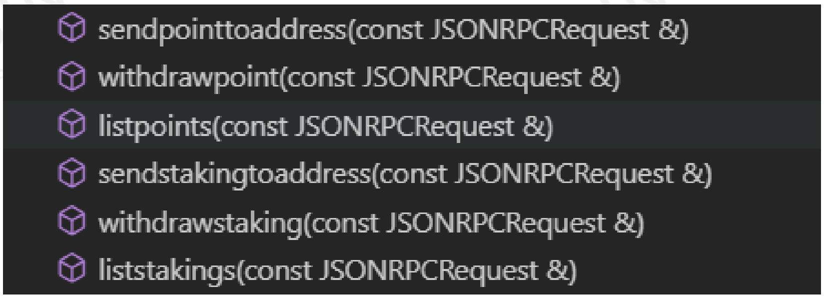

Figure 12 QITCOIN's unique interface

After testing, the new interface function of this part is in line with the design, and there are no security issues.

4. Transaction model security

4.1 Transaction replay audit

According to the current UTXO model of QITCOIN and its only one main chain, it is not possible to perform replay attacks where transactions are executed on different chains. However, if there are subsequent fork chains, please pay attention to the version control of the chain to avoid replay attacks.

4.2 Double-spending attack audit

For ordinary transactions on the QITCOIN chain, a pre-check will be carried out to update the coin status and check the transaction double-spending.

```
bool Consensus::CheckTxInputs(const CTransaction& tx, CValidationState& state, const CCoinsViewCache& inputs, const CCoinsViewCache& prevInputs,
    int nSpendHeight, CAmount& txfee, const Consensus::Params& params)
{
    // are the actual inputs available?
    if (!inputs.HaveInputs(tx)) {···

    CAmount nValueIn = 0;
    for (unsigned int i = 0; i < tx.vin.size(); ++i) {
        const COutPoint &prevout = tx.vin[i].prevout;
        const Coin& coin = inputs.AccessCoin(prevout);
        assert(!coin.IsSpent());

        // If prev is coinbase, check that it's matured
        if (coin.IsCoinBase() && nSpendHeight - coin.nHeight < COINBASE_MATURITY) {
            return state.Invalid(ValidationInvalidReason::TX_PREMATURE_SPEND, false, REJECT_INVALID, "bad-txns-premature-spend-of-coinbase",
                strprintf("tried to spend coinbase at depth %d", nSpendHeight - coin.nHeight));
        }

        // Check for negative or overflow input values
        nValueIn += coin.out.nValue;
        if (!MoneyRange(coin.out.nValue) || !MoneyRange(nValueIn)) {
            return state.Invalid(ValidationInvalidReason::CONSENSUS, false, REJECT_INVALID, "bad-txns-inputvalues-outofrange");
        }

        // Check special coin spend
        if (coin.IsBindPlotter() && nSpendHeight < GetUnbindPlotterLimitHeight(CBindPlotterInfo(prevout, coin), prevInputs, params)) {
            return state.Invalid(ValidationInvalidReason::TX_INVALID_BIND, false, REJECT_INVALID, "bad-txns-unbindplotter-limit");
        }
        if (coin.IsPoint() && coin.nHeight + PointPayload::As(coin.payload)->GetLockBlocks() > (uint32_t) nSpendHeight) {
            return state.Invalid(ValidationInvalidReason::CONSENSUS, false, REJECT_INVALID, "bad-txns-point-locked");
        }
        if (coin.IsStaking() && coin.nHeight + StakingPayload::As(coin.payload)->GetLockBlocks() > (uint32_t) nSpendHeight) {
            return state.Invalid(ValidationInvalidReason::CONSENSUS, false, REJECT_INVALID, "bad-txns-staking-locked");
        }
    }
```

Figure 13 Transaction check function

In addition to ordinary transactions, the team conducted a double-spending attack test for special transactions in QITCOIN: sendpointtoaddress, sendstakingtoaddress, withdrawpoint, withdrawstaking, etc., and found that none of the attacks were successful.

5. Consensus Security

The consensus algorithm used by QITCOIN is CPoC (Conditioned-Proof of Capacity), that is, conditional capacity proof. Participants get different profit coefficients according to different conditions. This consensus is an upgraded and improved algorithm of PoC2 (Proof of Capacity) consensus. On the basis of PoC consensus, a POS consensus mechanism that requires miners to use QTC as the stake is added.

In the current version of the consensus mechanism, the block reward is divided into two parts: miner income and POS income. If the point amount that miners received meets the mining required balance, they will receive 80% of the block rewards, and the remaining 20% will be obtained by the top 10 of the staking list in proportion to the amount; if the point amount do not meet the mining required balance, they will only receive 5% of the block rewards, and the remaining 95% will be obtained by the top 10 of the staking list.

```
std::vector<CTxOut> GetBlockReward(const CBlockIndex* pindexPrev, const CAmount& nFees, const CAccountID& generatorID, const uint64_t& nPlotterId, const CCoinsViewCache& view,
{
    const int nHeight = pindexPrev ? (pindexPrev->nHeight + 1) : 0;
    const CAmount nSubsidy = GetBlockSubsidy(nHeight, consensusParams) + nFees;

    std::vector<CTxOut> vTxOut;

    if (nSubsidy == 0)
        return vTxOut;

    if (nHeight == 1)···
    else
    {
        if (nHeight < consensusParams.nBindPlotterCheckHeight)···
        else
        {
            // reward to miner
            const CAmount balancePointReceived = view.GetAccountPointReceivedBalance(generatorID);
            const CAmount miningRequireBalance = poc::GetMiningRequireBalance(generatorID, nPlotterId, nHeight, view, nullptr, consensusParams);
            if (balancePointReceived >= miningRequireBalance) {
                vTxOut.push_back(CTxOut((nSubsidy * consensusParams.nPledgeFullRewardRatio) / 1000, CScript()));
            } else {
                vTxOut.push_back(CTxOut((nSubsidy * consensusParams.nPledgeLowRewardRatio) / 1000, CScript()));
            }

            // staking to top 10
            const CAccountBalanceList vSortedTopAccount = view.GetTopStakingAccounts(10);
            if (!vSortedTopAccount.empty()) {
                CAmount totalBalance = 0;
                for (auto &acc : vSortedTopAccount) {
                    totalBalance += acc.second;
                }

                if (totalBalance > 0) {
                    const CAmount nStakingAmount = nSubsidy - vTxOut[0].nValue;
                    for (auto &acc : vSortedTopAccount) {
                        CAmount nUserStakingAmount = ((1000 * acc.second / totalBalance) * nStakingAmount) / 1000;
                        if (nUserStakingAmount == 0) {
                            break;
                        }
                        vTxOut.push_back(CTxOut(nUserStakingAmount, GetScriptForAccountID(acc.first)));
                    }
                }
            }
        }
```

Figure 14 Block reward calculation function

Currently, there are two types for miners to increase the point amount: lock-up for 540 days and 360 days. The number of effective records for point amount with a lock-up period of 540 days is the number of user's QTC locked; while the number of effective records for a 360-day point amount is half the number of QTC locked. In addition, the chain itself requires that the amount of lock-up no less than 10 QTC each time and that the user has sufficient QTC balance.

```
CAmount GetPointAmount(CAmount amount, int lockBlocks)
{
    if (lockBlocks == 360 * 480) {
        return amount / 2;
    }
    if (lockBlocks == 540 * 480) {
        return amount;
    }

    return 0;
}
```

Figure 15 Effective point amount calculation function

For miners, the mining required balance required is the product of the user’s bound hard drive capacity and the mining pledge ratio, where the mining pledge ratio is initially 5 QTC/TiB and will be halved in tandem with the halving of the block reward.

```
CAmount GetMiningPledgeRatio(int nMiningHeight, const Consensus::Params& params)
{
    AssertLockHeld(cs_main);
    CAmount nSubsidy = GetBlockSubsidy(nMiningHeight, params);
    if (nSubsidy == 0) {
        return 0;
    }
    int half = (75 * COIN) / nSubsidy;
    return params.nPledgeRatio / half;
}

CAmount GetCapacityRequireBalance(int64_t nCapacityTB, int nMiningHeight, const Consensus::Params& params)
{
    CAmount ratio = GetMiningPledgeRatio(nMiningHeight, params);
    return ((ratio * nCapacityTB + COIN/2) / COIN) * COIN;
}
```

Figure 16 Minimum point amount calculation function

For the miner part of the reward, the miner will first get a one-time 20%, and the remaining 80% will be released linearly at 5% per 5,400 blocks within 180 days; for the top ten users of staking, the rewards and dividends are sent at one time.

```
    // re-calc reward to miner. 20% to miner, 80% release in 86400 blocks (180 days, 5% release every 5400 blocks, 16 rounds)
    vTxOut[0].nValue = (vTxOut[0].nValue * 20) / 100;
    for (const CBlockIndex* pindex = pindexPrev;
            pindex != nullptr && pindex->nHeight > 1 && pindex->nHeight >= pindexPrev->nHeight - 86400;
            pindex = pindex->GetAncestor(pindex->nHeight - 5400)) {
        vTxOut.push_back(CTxOut(pindex->minerRewardTxOut.nValue / 4, pindex->minerRewardTxOut.scriptPubKey)); // 5%
    }
}
```

Figure 17 Miner reward release mechanism

Addresses participating in the staking ranking can receive staking from themselves or other addresses, and their lock-up period and valid staking quantity are recorded in the same way as miners' stake (540 days of lock-up is recorded by QTC quantity; 360 days of lock-up is recorded by half of QTC quantity), and the corresponding table is maintained in the database. In each block reward distribution, the node will pull the staking data from the database, sort and extract the top 10 to distribute the reward in proportion to the number of staking.

```
// staking to top 10
const CAccountBalanceList vSortedTopAccount = view.GetTopStakingAccounts(10);
if (!vSortedTopAccount.empty()) {
    CAmount totalBalance = 0;
    for (auto &acc : vSortedTopAccount) {
        totalBalance += acc.second;
    }

    if (totalBalance > 0) {
        const CAmount nStakingAmount = nSubsidy - vTxOut[0].nValue;
        for (auto &acc : vSortedTopAccount) {
            CAmount nUserStakingAmount = ((1000 * acc.second / totalBalance) * nStakingAmount) / 1000;
            if (nUserStakingAmount == 0) {
                break;
            }
            vTxOut.push_back(CTxOut(nUserStakingAmount, GetScriptForAccountID(acc.first)));
        }
    }
}
```

Figure 18 Reward distribution of staking top 10

The following are the stages of change in the mining consensus:

(1) Genesis stage

Block range: 1

Block reward: No reward, allocate initial tokens to the preset address. Distribute 10500000 QTC to 3LX1uGfaDm6LGj6gy7aFJc7azpyzKhUaRs and 3JSgHDJjzDSHr1o5Lx2b1Fe6AwfFn8LNSX addresses.

(2) Condition-free mining stage

Block range: 2-3359(About 1 week)

Block reward: 75QTC/Block (Miners get 100% block rewards without any staking)

(3) Conditional mining stage (CPoC)

Block range: 3360+

Block reward: The initial reward is 75 QTC per block. The block height for the first halving time is 420,000, after that every 700,000 blocks are halved, and there is no reward after 64 halving times.

6. Dependent library security audit

Some dependent versions are updated rapidly and the latest version information may be delayed. Please refer to the actual situation.

(1)berkeley-db

Current version: 18.1.32

Latest version: 18.1.40

Historical vulnerabilities: None

Security recommendations: None

(2)Boost

Current version: 1.70.0

Latest version: 1.77

Historical vulnerabilities: None

Security recommendations: None

(3) expat

Current version: 2.2.7

Latest version: 2.4.1

Historical vulnerabilities: CVE-2019-15903, CVE-2021-40439

Security recommendations: None. Historical vulnerabilities do not affect the security of the chain for the time being.

(4)fontconfig

Current version: 2.12.1

Latest version: 2.13.94

Historical vulnerabilities: None

Security recommendations: None

(5)freetype

Current version: 2.7.1

Latest version: 2.11.0

Historical vulnerabilities: None

Security recommendations: None

(6)libevent

Current version: 2.1.8-stable

Latest version: 2.1.12-stable

Historical vulnerabilities: Integer overflow

Security recommendations: None. Historical vulnerabilities will not affect the security of the chain for the time being.

(7)MiniUPnPc

Current version: 2.0.20180203

Latest version: 2.2.3

Historical vulnerabilities: None

Security recommendations: None

(8)openssl

Current version: 1.0.1k

Latest version: Openssl 3.0

Historical vulnerabilities: CVE-2016-0705, CVE-2017-3732, CVE-2018-0737, CVE-2019-10211, CVE-2020-36165, CVE-2021-3712

Security recommendations: None. Historical vulnerabilities will not affect the security of the chain for the time being.

(9)protobuf

Current version: 2.6.1

Latest version: 3.18.1

Historical vulnerabilities: CVE-2021-30179, CVE-2021-3121

Security recommendations: None. Historical vulnerabilities will not affect the security of the chain for the time being.

(10)qrencode

Current version: 3.4.4

Latest version: 4.1.1

Historical vulnerabilities: None

Security recommendations: None

(11)QT

Current version: 5.9.7

Latest version: 6.2

Historical vulnerabilities: CVE-2018-19873, CVE-2019-12828, CVE-2020-12267, CVE-2021-3401 etc.

Security recommendations: None. Historical vulnerabilities will not affect the security of the chain for the time being.

(12)ZeroMQ

Current version: 4.3.1

Latest version: 4.3.2

Historical vulnerabilities: CVE-2019-6250, CVE-2020-15166, CVE-2021-20234 etc.

Security recommendations: None. Historical vulnerabilities will not affect the security of the chain for the time being.

(13)zlib

Current version: 1.2.11

Latest version: 1.2.11

Historical vulnerabilities: None

Security recommendations: None

7. Audit Conclusion

Beosin Technology has used a simulated attack to conduct multi-dimensional and comprehensive security audit on aspects of the module security and business logic security of the public blockchain QITCOIN. **The public blockchain QITCOIN passed all audit items. The overall result is Pass.**

Official Website

https://lianantech.com

E-mail

market@lianantech.com

Twitter

https://twitter.com/Beosin_com