知道创宇

密级 ★ | 项目组公开

Blockchain Security Audit Report

STATUS

SAFE

★ ★ ★ ★ ★

主测人：何平生 王梓龙

Release notes

Revised one	Revise the	The revision	The version	review
He Pingsheng wang Zilong	Written document	2020/10/20	V1.0	Chenyu

Document information

The document name	Document	The document number	Level of
DeFiChain safety audit report	V1.0	【DeFiChain-GLSJ-20201020】	Open project Team

The statement

Knownsec shall only issue this report for the facts that have occurred or existed before the issuance of this report, and shall assume corresponding responsibilities accordingly.Knownsec is not in a position to judge the security status of its intelligent contract and is not responsible for the occurrence or existence of such facts after issuance.The security audit analysis and other contents in this report are based only on the documents and materials provided by the information provider to Knownsec as of the issuance of this report.Creation hypothesis: there is no absence, modification, deletion or concealment of the provided information.In the event that the information provided is missing, altered, deleted, concealed or reflected does not conform to the actual situation, Knownsec shall not be liable for any loss or adverse effect caused thereby.

Directory

1. Review

The effective test report from October 14, 2020 to the end of October 20, 2020, during this period for DeFiChain male chain security and normative audit and report as a statistical basis.

Knownsec blockchain system security test method:

White box testing

Through the source code of the program, SDK carries out security audit and dynamic debugging vulnerability mining for p2p and RPC nodes

Grey box testing

Source code is securely scanned and audited by tools to find vulnerability points that may lead to exceptions

Black box testing

Simulate the attacker to conduct security test attack on the node to see whether the node responds normally

Through this code audit, it is known that Knownsec engineers found zero types of common security risks in object codes, including zero serious risk code files or functions, zero high-risk code files or functions, zero medium-risk code files or functions, and zero low-risk code files or functions.

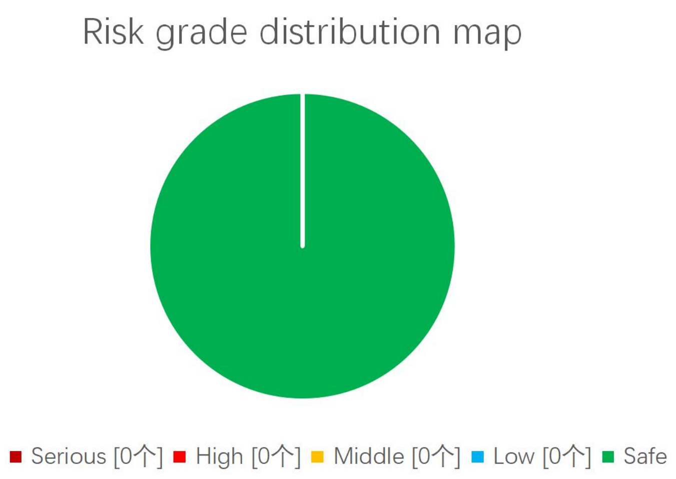
Since this test is conducted in a non-production environment, all codes are updated, the test process is communicated with the relevant interface personnel, and relevant test operations are carried out under the control of operational risks, so as to avoid production and operation risks and code security risks in the test process.

2. Vulnerability analysis

2.1. Vulnerability grade distribution

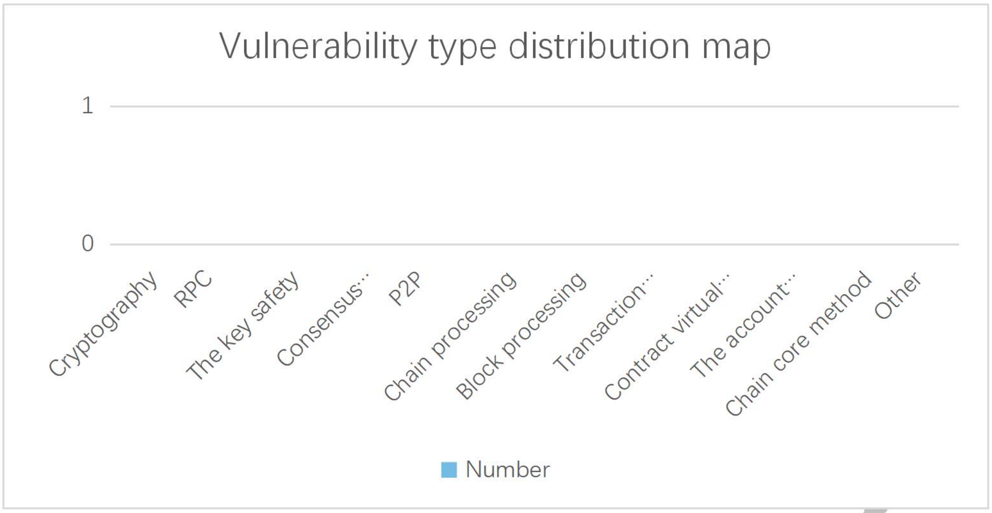
This vulnerability risk is calculated by level:

Statistical table of vulnerability risk levels			
Serious	High	Middle	Low risk
0	0	0	0



2.2. Distribution of vulnerability types

The statistics of this vulnerability by type:


Vulnerability type distribution map
1
0
Cryptography
RPC
The key safety
Consensus…
P2P
Chain processing
Block processing
Transaction…
Contract virtual…
The account…
Chain core method
Other
Number

3. Project analysis

3.1. Project introduction

As described in the original Satoshi white paper, Bitcoin was designed as a form of digital cash, a store and exchange of value. The development of Ethereum and smart contracts allowed new capabilities to be built on top of blockchain, and the concept of a global operating system for everything created a system that required a complex code base to handle smart contracts, slow throughput, and the difficulties surrounding system governance. The cryptocurrency industry is based on a simple premise: people should have complete control over their finances. While this may seem like a simple and obvious statement, the current system is far from providing truly user-controlled financial services. DeFi Blockchain was born out of this.

DeFi Blockchain focus on block chain function, and use it to disperse financing, at the same time provides unparalleled high transaction throughput, to reduce the risk of error, and specially designed for the block chain function of financial services and the development of intelligent dedicated to realize satoshi purpose: to create a reliable alternative forms of financial services based on the currency.

DeFi Blockchain regards decentralized financing as a specific and critical part of the Blockchain community. DeFi is a dedicated Blockchain optimized for DeFi applications. DeFi Blockchain is intentionally non-Turing complete and does not support any functions except those required for decentralized financing, resulting in higher throughput and better functions provided by Blockchain, especially finance-related Dapps. The advantage of a non-Turing complete command set is that coding errors (such as A DAO attack or a locked fund with parity) that plague an Ethereum intelligent contract are much less likely.

Website address: https://defichain.com

Main coding language: C++

3.2. The audit scope

Consensus Mechanism:

DeFiChain\ain\src\consensus\consensus.h

DeFiChain\ain\src\consensus\merkle.cpp

DeFiChain\ain\src\consensus\merkle.h

DeFiChain\ain\src\consensus\params.h

DeFiChain\ain\src\consensus\tx_check.cpp

DeFiChain\ain\src\consensus\tx_check.h

DeFiChain\ain\src\consensus\tx_verify.cpp

DeFiChain\ain\src\consensus\tx_verify.h

DeFiChain\ain\src\consensus\validation.h

DeFiChain\ain\src\pos.cpp

DeFiChain\ain\src\pos_kernel.cpp

Encryption to decrypt

DeFiChain\ain\src\crypto\sha512.cpp

DeFiChain\ain\src\crypto\sha256.cpp

DeFiChain\ain\src\wallet\crypter.cpp

Node operation

DeFiChain\ain\src\interfaces\node.cpp

DeFiChain\ain\src\interfaces\node.h

DeFiChain\ain\src\rpc\net.cpp

The wallet operation

DeFiChain\ain\src\interfaces\wallet.cpp

DeFiChain\ain\src\interfaces\wallet.h

DeFiChain\ain\src\wallet\walletutil.h

DeFiChain\ain\src\wallet\walletutil.cpp

DeFiChain\ain\src\wallet\wallettool.cpp

DeFiChain\ain\src\wallet\wallet.cpp

DeFiChain\ain\src\wallet\rpcwallet.cpp

DeFiChain\ain\src\wallet\rpcwallet.h

DeFiChain\ain\src\wallet\load.cpp

DeFiChain\ain\src\wallet\init.cpp

The DB operation

DeFiChain\ain\src\leveldb\*

DeFiChain\ain\src\wallet\walletdb.cpp

DeFiChain\ain\src\wallet\db.cpp

Token operation

DeFiChain\ain\src\masternodes\tokens.h

DeFiChain\ain\src\masternodes\tokens.cpp

Transaction

DeFiChain\ain\src\node\transaction.h

DeFiChain\ain\src\node\transaction.cpp

DeFiChain\ain\src\policy\fees.cpp

DeFiChain\ain\src\policy\feerate.cpp

DeFiChain\ain\src\primitives\transaction.cpp

DeFiChain\ain\src\primitives\transaction.h

RPC calls

DeFiChain\ain\src\rpc\protocol.h

DeFiChain\ain\src\rpc\server.h

DeFiChain\ain\src\rpc\server.cpp

DeFiChain\ain\src\rpc\request.h

DeFiChain\ain\src\rpc\request.cpp

DeFiChain\ain\src\rpc\register.h

DeFiChain\ain\src\rpc\rawtransaction_util.cpp

DeFiChain\ain\src\rpc\rawtransaction.cpp

DeFiChain\ain\src\rpc\mining.cpp

.

The main entry and scope of security testing are the modules listed above and extend to the context of the scope based on actual testing.

The main types of security audit include:

Code security

RPC security

P2P security

Consensus safety

Account system security

Contract virtual machine security

Business logic design security

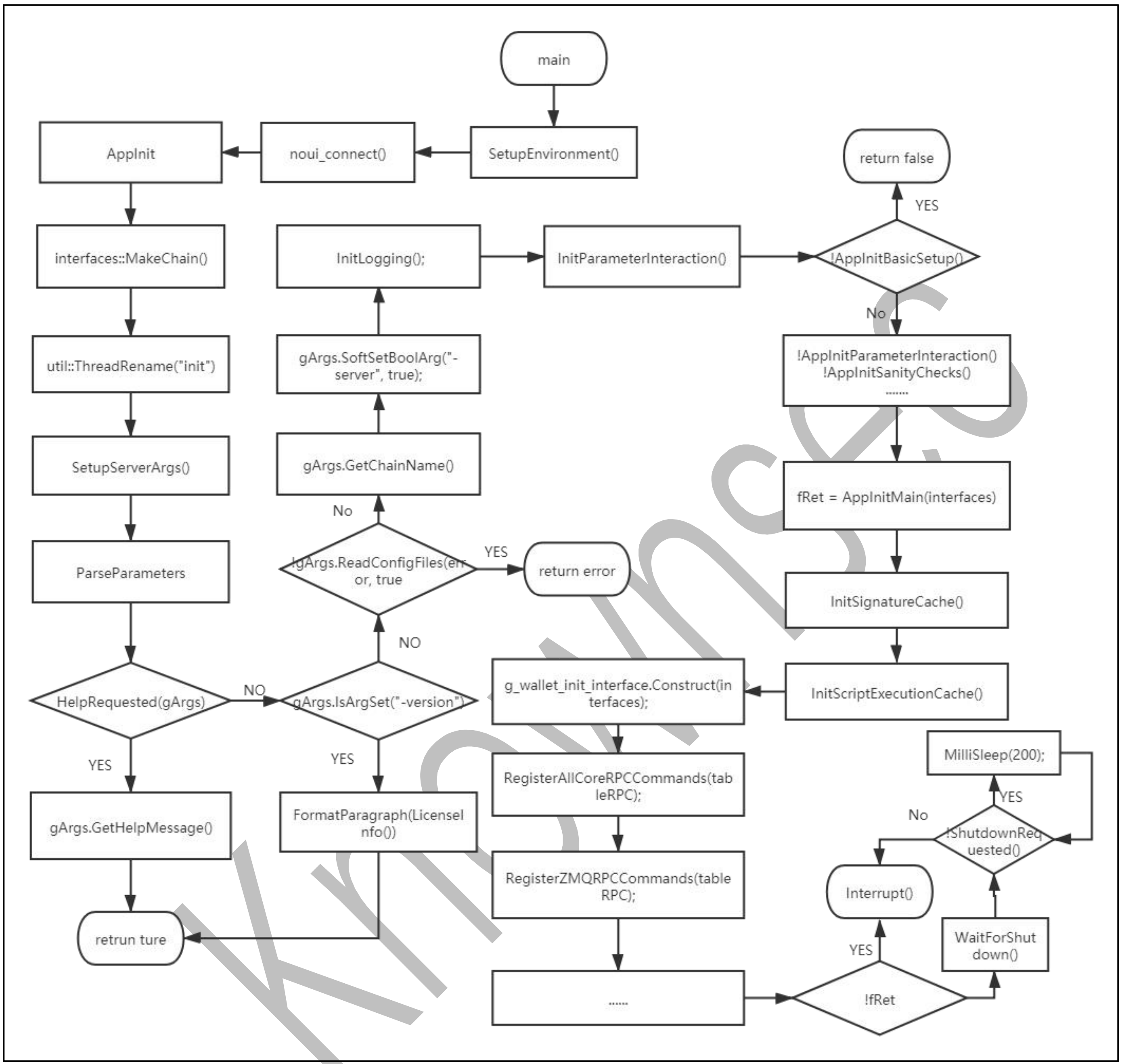
3.3. The directory structure

Core code directory structure:

```
├ ─ bench
| └ ─ data
├ ─ compat
├ ─ config
├ ─ consensus
├ ─ crypto
| └ ─ ctaes
├ ─ index
├ ─ interfaces
├ ─ leveldb
| ├ ─ the db
| ├ ─ doc
|   |   └ — bench
|   ├ — helpers
|   |   └ — memenv
|   ├ — the include
|   |   └ — leveldb
|   ├ — issues
|   ├ — the port
|   |   └ — win
|   ├ — the table
|   └ — util
├ — masternodes
├ — node
├ — the policy
├ — the primitives
├ — RPC
├ — script
├ — secp256k1
|   ├ — build - aux
|   |   └ — the m4
|   ├ — contrib
|   ├ — the include
|   ├ — obj
|   ├ — sage
|   └ — the SRC
|   ├ — the asm
|   ├ — Java
|   |   └ — org
|   |   └ — bitcoins
|   └ — modules
|   ├ — ecdh
|   └ — recovery
├ — SPV
```

```
|   ├ — bcash
|   ├ — bitcoins
|   └ — support
├ — support
|   └ — allocators
├ — test
|   ├ — data
|   ├ — fuzz
|   └ — gen
├ — univalue
|   ├ — build - aux
|   |   └ — the m4
|   ├ — gen
|   ├ — the include
|   ├ — lib
|   ├ — PC
|   └ — test
├ — util
├ — wallet
|   └ — test
└ — ZMQ
```

3.4. Start the process


main
SetupEnvironment()
noui_connect()
AppInit
interfaces::MakeChain()
util::ThreadRename("init")
SetupServerArgs()
ParseParameters
HelpRequested(gArgs)
YES
gArgs.GetHelpMessage()
retrun ture
NO
gArgs.IsArgSet("-version")
YES
FormatParagraph(LicenseInfo())
NO
!gArgs.ReadConfigFiles(error, true
YES
return error
No
gArgs.GetChainName()
gArgs.SoftSetBoolArg("-server", true);
InitLogging();
InitParameterInteraction()
!AppInitBasicSetup()
YES
return false
No
!AppInitParameterInteraction()
!AppInitSanityChecks()
....
fRet = AppInitMain(interfaces)
InitSignatureCache()
InitScriptExecutionCache()
g_wallet_init_interface.Construct(interfaces);
RegisterAllCoreRPCCommands(tableRPC);
RegisterZMQRPCCommands(tableRPC);
......
!fRet
YES
Interrupt()
WaitForShutdown()
!ShutdownRequested()
No
YES
MilliSleep(200);

3.5. A list of P2P related methods

*static int _BRTxPeerListHasPeer(const BRTxPeerList *list, UInt256 txHash, const BRPeer *peer)*

*static size_t _BRTxPeerListCount(const BRTxPeerList *list, UInt256 txHash)*

*static size_t _BRTxPeerListAddPeer(BRTxPeerList **list, UInt256 txHash, const BRPeer *peer)*

```
static int _BRTxPeerListRemovePeer(BRTxPeerList *list, UInt256 txHash, const BRPeer *peer)
static void _BRPeerManagerPeerMisbehavin(BRPeerManager *manager, BRPeer *peer)
static void _BRPeerManagerSyncStopped(BRPeerManager *manager)
static int _BRPeerManagerAddTxToPublishList(BRPeerManager *manager, BRTransaction *tx, void *info,void (*callback)(void *, int))
static size_t _BRPeerManagerBlockLocators(BRPeerManager *manager, UInt256 locators[], size_t locatorsCount)
static void _BRPeerManagerLoadBloomFilter(BRPeerManager *manager, BRPeer *peer)
.
```

3.6. List of RPC core interfaces

```
static UniValue getdifficulty(const JSONRPCRequest& request)
static UniValue getblockcount(const JSONRPCRequest& request)
static UniValue getbestblockhash(const JSONRPCRequest& request)
static UniValue waitforblockheight(const JSONRPCRequest& request)
static UniValue getblockheader(const JSONRPCRequest& request)
static CBlock GetBlockChecked(const CBlockIndex* pblockindex)
static UniValue getblock(const JSONRPCRequest& request)
static UniValue gettxoutsetinfo(const JSONRPCRequest& request)
UniValue gettxout(const JSONRPCRequest& request)
UniValue getblockchaininfo(const JSONRPCRequest& request)
static UniValue getchaintips(const JSONRPCRequest& request)
UniValue MempoolInfoToJSON(const CTxMemPool& pool)
static UniValue getmempoolinfo(const JSONRPCRequest& request)
static UniValue getchaintxstats(const JSONRPCRequest& request)
static UniValue getblockstats(const JSONRPCRequest& request)
UniValue scantxoutset(const JSONRPCRequest& request)
static UniValue getblockfilter(const JSONRPCRequest& request)
static UniValue getnetworkhashps(const JSONRPCRequest& request)
static UniValue generatetoaddress(const JSONRPCRequest& request)
static UniValue getmintinginfo(const JSONRPCRequest& request)
static UniValue prioritisetransaction(const JSONRPCRequest& request)
static UniValue getblocktemplate(const JSONRPCRequest& request)
static UniValue submitheader(const JSONRPCRequest& request)
static UniValue estimatesmartfee(const JSONRPCRequest& request)
static UniValue estimaterawfee(const JSONRPCRequest& request)
static UniValue validateaddress(const JSONRPCRequest& request)
static UniValue createmultisig(const JSONRPCRequest& request)
UniValue getdescriptorinfo(const JSONRPCRequest& request)
UniValue deriveaddresses(const JSONRPCRequest& request)
static UniValue verifymessage(const JSONRPCRequest& request)
static UniValue signmessagewithprivkey(const JSONRPCRequest& request)
static UniValue getmemoryinfo(const JSONRPCRequest& request)
UniValue logging(const JSONRPCRequest& request)
static UniValue getnetworkinfo(const JSONRPCRequest& request)
static UniValue setban(const JSONRPCRequest& request)
static UniValue listbanned(const JSONRPCRequest& request)
static UniValue clearbanned(const JSONRPCRequest& request)
static UniValue getnodeaddresses(const JSONRPCRequest& request)
static UniValue createrawtransaction(const JSONRPCRequest& request)
static UniValue decoderawtransaction(const JSONRPCRequest& request)
static UniValue decodescript(const JSONRPCRequest& request)
static UniValue combinerawtransaction(const JSONRPCRequest& request)
static UniValue signrawtransactionwithkey(const JSONRPCRequest& request)
static UniValue sendrawtransaction(const JSONRPCRequest& request)
static UniValue testmempoolaccept(const JSONRPCRequest& request)
UniValue decodepsbt(const JSONRPCRequest& request)
UniValue combinepsbt(const JSONRPCRequest& request)
.
```

3.7. Third-party libraries

The Berkeley DB 4.3.0

The Boost 1.70.0

Clang 3.3 +

The Expat 2.2.7

Fontconfig 2.12.1

FreeType 2.7.1

GCC 4.8 +

Libevent 2.1.8 - stable

MiniUPnPc 2.0.20180203

Protobuf 2.6.1

Qrencode 3.4.4

XCB

Xkbcommon

ZeroMQ 4.3.1

Zlib 1.2.11

4. The audit results

4.1. Cryptography

4.1.1. Random number Range and Probability Distribution [Pass]

Audit instructions: Check whether the key random number generation algorithm is reasonable, and test whether the real probability of key random number generation meets the requirements (normal uniform distribution).

Risk and harm: sensitive information leakage, arbitrary issuance of transactions, affecting the safety of assets, etc.

Risk document:

Audit process: It was detected that the official RANDOM library involving BTC CORE in the common chain code initialized random seeds, which were in line with the safe coding operation.

```
Static uint64_t GetRdRand() Noexcept //knownsec//Reference BTC CORE official code design
{
    // RdRand may very rarely fail. Invoke it up to 10 times in a loop to reduce this risk.
#ifdef __i386__
        uint8_t ok;
        uint32_t r1, r2;
        for (int i = 0;     i < 10;     ++i) {
                __asm__ volatile (".byte 0x0f, 0xc7, 0xf0;     setc %1" : "=a"(r1), "=q"(ok) :: "cc");     //
rdrand %eax
                if (ok) break;
        }
        for (int i = 0;     i < 10;     ++i) {
```

```
        __asm__ volatile (".byte 0x0f, 0xc7, 0xf0;    setc %1" : "=a"(r2), "=q"(ok) :: "cc");    //
rdrand %eax
        if (ok) break;
    }
    return (((uint64_t)r2) << 32) | r1;
#elif DeFined(__x86_64__) || DeFined(__amd64__)
    uint8_t ok;
    uint64_t r1;
    for (int i = 0;    i < 10;    ++i) {
        __asm__ volatile (".byte 0x48, 0x0f, 0xc7, 0xf0;    setc %1" : "=a"(r1), "=q"(ok) :: "cc");
// rdrand %rax
        if (ok) break;
    }
    return r1;
#else
#error "RdRand is only supported on x86 and x86_64"
#endif
}
```

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.1.2. Cryptography Algorithm Implementation/Use [Pass]

Audit note: Check whether the use or implementation of cryptography such as signature, hash and verification is reasonable.

Risk and harm: Forgery of blocks/trades or, in serious cases, chain bifurcation.

Risk document:

Audit process: After detection, the library Crypto/Sha256 and Crypto/Sha512 were used when the core data encryption operation was involved in the public chain code, which conforms to the safe coding operation.

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.2. RPC

4.2.1. Sensitive interface permission [Pass]

Audit notes: Check the access rights of RPC sensitive interfaces, whether they are exposed, and test whether RPC interfaces have access to sensitive operations or data.

Risk and harm: sensitive information leakage, arbitrary issuance of transactions, affecting the safety of assets, etc.

Risk document:

Audit process: Through detection, the core data operation involved in the public chain code has relevant permission verification, in line with the security coding operation.

```
Static Bool RPCAuthorized(const STD :: String & strAuth, STD :: String &
strAuthUsernameOut){//knownsec//strictly checks the IDENTITY of RPC caller
      if (strRPCUserColonPass.empty()) // Belt-and-suspenders measure if InitRPCAuthentication
was not called
           return false;
      if (strAuth.substr(0, 6) ! = "Basic ")
```

```
        return false;
    std::string strUserPass64 = strAuth.substr(6);
    boost::trim(strUserPass64);
    std::string strUserPass = DecodeBase64(strUserPass64);

    if (strUserPass.find(':') ! = std::string::npos)
        strAuthUsernameOut = strUserPass.substr(0, strUserPass.find(':'));

    //Check if authorized under single-user field
    if (TimingResistantEqual(strUserPass, strRPCUserColonPass)) {
        return true;
    }
    return multiUserAuthorized(strUserPass);
}
```

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.2.2. Business logic [Pass]

Audit instructions: check the access rights of RPC interface, export wallet, backup wallet and other functions to see if there are oversteps, logic errors, etc.

Risk and harm: sensitive information leakage, arbitrary issuance of transactions, affecting the safety of assets, etc.

Risk document:

Audit process: Through testing, the core operation involved in the public chain code is correct in the use of authentication and related processing logic, in line with the safe coding operation.

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.2.3. Traditional Web Security [Pass]

Audit notes: Check/test traditional Web security items such as XXE injection, CSRF, SSRF, path traversal, and plaintext transmission of sensitive information.

Risks and hazards: remote command execution, RPC interface exposure, CSRF and SSRF attacks, arbitrary file reading, sensitive information leakage, etc.

Risk document:

Audit process: After detection, there is no common web attack risk such as XXE injection in the public chain code.

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.3. The key safety

4.3.1. Private key/Mnemonic Generation Algorithm [Pass]

Audit note: Check if the logic of the private key generation algorithm is reasonable and test if the complexity bottleneck is as expected.

Risk and harm: sensitive information leakage, arbitrary issuance of transactions, affecting the safety of assets, etc.

Risk document:

Audit process: After detection, the private key/mnemonic generation algorithm in the public chain code referred to the official source code of Bitcoin, and no related risks were found.

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.3.2. Private key/Mnemonic plaintext storage [Pass]

Audit notes: Check how the file is stored, whether it is encrypted or can be decrypted

Risk and harm: cooperating with other vulnerabilities can lead to the disclosure of private keys, affecting the security of assets, or even lead to chain bifurcation and other problems.

Risk document:

Audit process: after detection, the private key/mnemonic in the public chain code encrypted storage.

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.3.3. Private key/Mnemonic use trace [Pass]

Audit note: Check whether the private key/mnemonic in the code logic is cleaned up in time after the use of traces, and check whether sensitive data can be found from logs, memory dumps after the use of traces can leak information.

Risk and harm: cooperating with other vulnerabilities can lead to the disclosure of private keys, affecting the security of assets, or even lead to chain bifurcation and other problems.

Risk document:

Audit process: After detection, the use of private key/mnemonic in the public chain code conforms to the safe operation specification, and no related risks are found.

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.3.4. Private key/Mnemonic residue [Pass]

Audit note: Check that the private key/mnemonic deletion operation is complete.

Risk and harm: cooperating with other vulnerabilities can lead to the disclosure of private keys, affecting the security of assets, or even lead to chain bifurcation and other problems.

Risk document:

Audit process: After detection, the use of private key/mnemonic in the public chain code conforms to the safe operation specification, and no related risks are found.

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.3.5. Abuse of private keys/Mnemonics [Pass]

Audit note: Check if private keys/mnemonics are used in non-essential locations.

Risk hazard: Increases the risk of private key leakage.

Risk document:

Audit process: After detection, the use of private key/mnemonic in the public chain code conforms to the safe operation specification, and no related risks are found.

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.4. Consensus mechanism

4.4.1. Consensus Verification Implementation [Pass]

Audit note: Check whether consensus such as POW/POC can be used to construct legal blocks at less than expected cost.

Risk and harm: forgery of calculation force, centralization of calculation force, and serious cases can lead to 51% attack, chain bifurcation and other problems.

Risk document:

Audit process: No significant security risks were detected.

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.4.2. Consensus Mechanism Design [Pass]

Audit note: For the new type of consensus mechanism, check whether there are security risks in the design according to its documents, such as missing transaction/non-verification of transaction/falsification of calculation power proof/centralization /BP crime, etc.

Risk and harm: analyze according to specific situation.

Risk document:

Audit process: After audit, no obvious security risk was found.

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.5. P2P

4.5.1. Differentiated response to Sensitive information [Pass]

Audit notes: Examine the response data logic for the various responses and test whether there are different responses for sensitive information in the various responses.

Risk and harm: sensitive information leakage.

Risk document:

Audit process: After detection, the public chain code involved in P2P connection response function only to meet the standard MSG protocol response, in line with the security coding specifications.

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.5.2. Fuzz malformation data Test [Pass]

Audit n notes: Check the response data logic of the various responses and test whether the nodes in the various responses are responding normally.

Hazard risk: Malformed data attacks can cause nodes to deny service or divulge chain-related information.

Risk document:

Audit process: After detection, the public chain code involved in P2P connection response function only to meet the standard MSG protocol response, in line with the security coding specifications.

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.5.3. Node discovery Algorithm [Pass]

Audit note: Check nodes to find out whether the algorithm is balanced and unpredictable, such as distance algorithm imbalance.

Risk and harm: analyze according to specific situation.

Risk document:

Audit process: no obvious security problems were detected.

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.5.4. Node Communication Protocol [Pass]

Audit description: check whether there is design/implementation layer problem in the communication protocol between nodes, carry out fuzzy test on nodes, and test the response of nodes to malformed packets.

Risk and harm: analyze according to specific situation.

Risk document:

Audit process: After detection, the public chain code involved in the core P2P network communication layer referred to the official source code of Bitcoin, no obvious security problems were found.

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.5.5. Routing table pollution [Pass]

Audit note: Check whether the routing table can be inserted or overwritten at will.

Hazard: reduces the difficulty of eclipse attacks, etc.

Risk document:

Audit process: After detection, the public chain code involved in the core P2P network communication layer referred to the official source code of Bitcoin, no obvious security problems were found.

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.5.6. Flow limit [Pass]

Audit note: Check the restrictions on connection number and packet size in various P2P protocols in the code logic, try to establish a large number of connections with the target node, and construct packets sending various sizes.

Risks and hazards: DoS nodes occupy local storage resources, or even cause the node program to crash.

Risk document:

Audit process: After detection, the public chain code involved in the core P2P network communication layer referred to the official source code of Bitcoin, no obvious security problems were found.

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.5.7. Node punishment mechanism [Pass]

Audit note: Check whether the penalty mechanism for nodes is reasonable.

Hazard risk: Isolate the normal node.

Risk document:

Audit process: After detection, the public chain code involved in the core P2P network communication layer referred to the official source code of Bitcoin, no obvious security problems were found.

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.5.8. Eclipse attack [Pass]

Audit notes: Assess the costs and hazards of a solar eclipse attack and provide quantitative analysis if necessary.

Risk and harm: isolate normal nodes, double asset, or even lead to chain rollback, chain bifurcation and other problems.

Risk document:

Audit process: After detection, the public chain code involved in the core P2P network communication layer referred to the official source code of Bitcoin, no obvious security problems were found.

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.6. Chain processing

4.6.1. Longest chain Selection and Switching Algorithm [Pass]

Audit description: Check whether the longest chain selection and switching algorithm and implementation are reasonable.

Risk hazards: chain rollback, chain bifurcation.

Risk document:

Audit process: The chain switching and selection logic in the common chain code conforms to the security logic after detection, and no obvious security problems are found.

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.6.2. Chain synchronization logic [Pass]

Audit notes: Check the scope of the synchronous block, the validation logic, and the reasonability of other asynchronous operations during the period.

Risk and harm: chain branching.

Risk document:

Audit process: After detection, the chain synchronization related codes in the common chain code conform to the safe coding operation, and no obvious security problems were found.

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.7. Block processing

4.7.1. Block hash collision [Pass]

Audit note: Check the block hash collision construction mode, and the processing of the collision is reasonable.

Hazard: DoS node occupying local storage resource.

Risk document:

Audit process: It was detected that sha256 or SHA512 were used in all the core data encryption operations in the public chain code.

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.7.2. Block processing resource restrictions [Pass]

Audit notes: Check for reasonable resource restrictions on orphan block pools, validation calculations, hard disk addressing, etc.

Risk and harm: use local storage, CPU resources DoS node, or even cause the node program crash.

Risk document:

Audit process: no obvious security problems were found in the block processing logic in the common chain code after detection.

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.7.3. Block signature [Pass]

Audit note: Check the block signature items and verify that the logic is sufficient.

Risk and harm: forgery of blocks can generate unexpected returns, or even lead to chain bifurcation, data tampering and other problems.

Risk document:

Audit process: no obvious security problems were found in the block processing logic in the common chain code after detection.

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.7.4. Merkle Tree Root Construction Method [Pass]

Audit note: Check whether Merkle Tree root is built in a reasonable way.

Risk and harm: forgery block, resulting in chain bifurcation and other problems.

Risk document:

Audit process: No obvious security problems were found in the related logic of block Merck tree construction in the common chain code after testing.

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.8. Transaction processing

4.8.1. Transaction signature [Pass]

Audit note: Check whether the signature items and validation logic of each type of transaction are sufficient.

Risk and harm: replay of in-chain or cross-chain transactions, counterfeit legal transactions, affecting the safety of assets.

Risk document:

Audit process: no obvious security problems were found in the transaction signature logic in the common chain code after detection.

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.8.2. Transaction processing logic [Pass]

Audit instructions: check the detection and treatment of double flower trading, double trading, illegal trading, etc.

Risk and harm: double blossom of assets in the same chain, trade replay.

Risk document:

Audit process: after detection, no obvious security problems were found in the transaction processing logic in the common chain code.

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.8.3. Transaction hash collision [Pass]

Audit notes: Check how the transaction hash collision is constructed and how the collision is handled.

Risk and harm: lost trades, frozen assets.

Risk document:

Audit process: After detection, the transaction in the public chain code requires the user's private key signature, and no obvious security problems were found.

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.8.4. Transaction processing resource constraints [Pass]

Audit notes: Check for reasonable resource limits for trading pools, validation calculations, hard disk addressing, etc.

Risk and harm: use local storage, CPU resources DoS node, or even cause the node program crash.

Risk document:

Audit process: after detection, no obvious security problems were found in the transaction processing logic in the common chain code.

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.8.5. Transaction fee setting [Pass]

Audit note: Check whether the charges for each atomic operation of transaction processing/contract execution are proportional to resource consumption.

Hazard risk: DoS at lower cost throughout the network.

Risk document:

Audit process: After testing, the transaction fee design in the common chain code is reasonable, and no obvious security problems are found.

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.9. Contract virtual machine

4.9.1. Contract VIRTUAL machine Sandbox Escape [Pass]

Audit instructions: check the sandbox logic of the virtual machine, judge whether the key object interception is thorough, test whether the common character escape and other attacks are effective, can access objects outside the sandbox environment.

Risk and hazard: Bypassing sandbox restrictions, in severe cases, can affect the execution logic of other contracts, or cause DoS and remote commands to execute.

Risk document:

Audit process: After testing, no relevant risks were found.

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.9.2. Contract Execution Logic [Pass]

Audit notes: Check the contract execution logic for any unexpected control flow and for any reasonable limitations in each link.

Risk and harm: unexpected behavior occurs in contract execution, or in serious cases, data tampering or node program crash.

Risk document:

Audit process: After testing, no relevant risks were found.

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.9.3. Contract deployment security [Pass]

Audit instructions: Check whether design defects and security risks exist in the deployment process of intelligent contract, and whether reasonable limitations have been made in each link.

Risk and harm: unexpected behavior in contract deployment, or in severe cases, program crash.

Risk document:

Audit process: After testing, no relevant risks were found.

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.9.4. Contract interface debugging [Pass]

Audit instructions: Debug the related functions of the contract interface for any defects or safety problems, and check whether reasonable restrictions have been made in each link.

Risk and harm: sensitive information leakage, program crash.

Risk document:

Audit process: After testing, no relevant risks were found.

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.10. Liquidity safety

4.10.1. Token creation logic [Pass]

Audit note: Check if there is a problem with the token creation logic.

Hazard: The node creating the Token operation cannot be in the Initial Block Download state, and strict checking of parameter types is required to avoid causing unexpected results.

Risk document:

Audit process: after detection, the common chain code token creation logic did not find obvious security problems.

```
UniValue createtoken(const JSONRPCRequest& request) {
        CWallet* const pwallet = GetWallet(request);
                        .
        If (pwallet->chain(). IsInitialBlockDownload ()) {//knownsec//node cannot be in the
InitialBlockDownload state
            throw JSONRPCError(RPC_CLIENT_IN_INITIAL_DOWNLOAD, "Cannot create
token while still in Initial Block Download");
        }
        pwallet->BlockUntilSyncedToCurrentChain();
      //knownsec//Check for RPC parameter types
        RPCTypeCheck(request.params, {UniValue::VOBJ, UniValue::VARR}, true);
        if (request.params[0].isNull()) {
            throw JSONRPCError(RPC_INVALID_PARAMETER,
                               "Invalid parameters, arguments 1 must be non-null and
expected as object at least with "
                               "{\"symbol\",\"collateralAddress\"}");
        }

        const UniValue metaObj = request.params[0].get_obj();
        UniValue txInputs = request.params[1];
        if (txInputs.isNull())
        {
            txInputs.setArray();
        }

        std::string collateralAddress = metaObj["collateralAddress"].getValStr();
        CTxDestination collateralDest = DecodeDestination(collateralAddress);
        If (alldest. Which () == 0) {//knwonsec// parameter check
            throw JSONRPCError(RPC_INVALID_PARAMETER, "collateralAddress (" +
collateralAddress + ") does not refer to any valid address");
        }

        int targetHeight;
        {
            LOCK(cs_main);
            targetHeight = ::ChainActive().Height() + 1;
        }

        CToken token;
        token.symbol = trim_ws(metaObj["symbol"].getValStr()).substr(0, CToken::MAX_TOKEN_SYMBOL_LENGTH);
        token.name = trim_ws(metaObj["name"].getValStr()).substr(0, CToken::MAX_TOKEN_NAME_LENGTH);
        token.flags = metaObj["isDAT"].getBool() ? token.flags | (uint8_t)CToken::TokenFlags::isDAT : token.flags;    // setting isDAT
//    token.decimal = metaObj["name"].get_int();    // fixed for now, check range later
//    token.limit = metaObj["limit"].get_int();    // fixed for now, check range later
//        token.flags = metaObj["mintable"].get_bool() ? token.flags | CToken::TokenFlags::Mintable : token.flags;    // fixed for now, check later
//        token.flags = metaObj["tradeable"].get_bool() ? token.flags | CToken::TokenFlags::Tradeable : token.flags;    // fixed for now, check later

        CDataStream metadata(DfTxMarker, SER_NETWORK, PROTOCOL_VERSION);
        metadata << static_cast<unsigned char>(CustomTxType::CreateToken)
                 << token;

        CScript scriptMeta;
        scriptMeta << OP_RETURN << ToByteVector(metadata);

        const auto txVersion = GetTransactionVersion(targetHeight);
        CMutableTransaction rawTx(txVersion);

        if(metaObj["isDAT"].getBool())
        {
            for(std::set<CScript>::iterator it = Params().GetConsensus().foundationMembers.begin(); it != Params().GetConsensus().foundationMembers.end() && rawTx.vin.size() == 0;    it++)
```

```
            {
                if(IsMine(*pwallet, *it) == ISMINE_SPENDABLE)
                {
                    CTxDestination destination;
                    if (! ExtractDestination(*it, destination)) {
                        throw JSONRPCError(RPC_INVALID_ADDRESS_OR_KEY, "Invalid
destination");
                    }
                    try {
                        rawTx.vin = GetAuthInputs(pwallet, destination, txInputs.get_array());
//knownsec//Permission check
                    }
                    catch (const UniValue& objError) {}
                }
            }
            if(rawTx.vin.size() == 0)
                throw JSONRPCError(RPC_INVALID_REQUEST, "Incorrect Authorization");
        }
        else
            rawTx.vin = GetInputs(txInputs.get_array());

        rawTx.vout.push_back(CTxOut(GetTokenCreationFee(targetHeight), scriptMeta));
        rawTx.vout.push_back(CTxOut(GetTokenCollateralAmount(),
GetScriptForDestination(collateralDest)));

        rawTx = fund(rawTx, request, pwallet);

        // check execution
        {
            LOCK(cs_main);
            CCustomCSView mnview_dummy(*pcustomcsview);     // don't write into actual DB
            const auto res = ApplyCreateTokenTx(mnview_dummy, g_chainstate->CoinsTip(),
CTransaction(rawTx), targetHeight,
                                                    ToByteVector(CDataStream{SER_NETWORK,
PROTOCOL_VERSION, token}), Params().GetConsensus());
            if (! res.ok) {
                throw JSONRPCError(RPC_INVALID_REQUEST, "Execution test failed:\n" +
res.msg);
            }
      }
      return signsend(rawTx, request, pwallet)->GetHash().GetHex();
}
```

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.10.2.Token destruction logic [Pass]

Audit note: Check if there is a problem with the token destruction logic.

Risk and hazard: The nodes of the destroyed Token operation cannot be in the Initial Block Download state. At the same time, it is necessary to determine whether the Token to be destroyed exists or not, and it is also necessary to judge whether the Token is a stable coin and strictly verify the passed parameters, so as to avoid unexpected results.

Risk document:

Audit process: after detection, the token destruction logic in the common chain code has not found obvious security problems.

```
UniValue destroytoken(const JSONRPCRequest& request) {
     CWallet* const pwallet = GetWallet(request);
     .
```

```
        If (pwallet->chain(). IsInitialBlockDownload ()) {//knwonsec// node cannot be in the InitialBlockDownload state
            throw JSONRPCError(RPC_CLIENT_IN_INITIAL_DOWNLOAD,
                               "Cannot resign Masternode while still in Initial Block Download");
        }
        pwallet->BlockUntilSyncedToCurrentChain();
        //knownsec//RPC parameter type checking
        RPCTypeCheck(request.params, {UniValue::VARR, UniValue::VSTR}, true);

        std::string const tokenStr = trim_ws(request.params[1].getValStr());
        CTxDestination ownerDest;
        uint256 creationTx{};
        {
            LOCK(cs_main);
            DCT_ID id;
            auto token = pcustomcsview->GetTokenGuessId(tokenStr, id); / / knownsec / / query token
            if (! token) {
                throw JSONRPCError(RPC_INVALID_PARAMETER, strprintf("Token %s does not exist!" , tokenStr));
            }
            if (id < CTokensView::DCT_ID_START) {
                throw JSONRPCError(RPC_INVALID_PARAMETER, strprintf("Token %s is a 'stable coin'", tokenStr));
            }
            LOCK(pwallet->cs_wallet);
            auto tokenImpl = static_cast<CTokenImplementation const& >(*token);
            auto wtx = pwallet->GetWalletTx(tokenImpl.creationTx);
            if (! wtx || ! ExtractDestination(wtx->tx->vout[1].scriptPubKey, ownerDest)) {
                throw JSONRPCError(RPC_INVALID_PARAMETER,
                                   strprintf("Can't extract destination for token's %s
collateral", tokenStr));
                }
                creationTx = tokenImpl.creationTx;
            }

            CMutableTransaction rawTx;
           //knownsec//Permission check
            rawTx.vin = GetAuthInputs(pwallet, ownerDest, request.params[0].get_array());

            CDataStream metadata(DfTxMarker, SER_NETWORK, PROTOCOL_VERSION);
            metadata << static_cast<unsigned char>(CustomTxType::DestroyToken)
                     << creationTx;

            CScript scriptMeta;
            scriptMeta << OP_RETURN << ToByteVector(metadata);

            rawTx.vout.push_back(CTxOut(0, scriptMeta));

            rawTx = fund(rawTx, request, pwallet);

            // check execution
            {
                LOCK(cs_main);
                CCustomCSView mnview_dummy(*pcustomcsview);     // don't write into actual
DB
                const                    auto                    res                    =
ApplyDestroyTokenTx(mnview_dummy,                          ::ChainstateActive().CoinsTip(),
CTransaction(rawTx), ::ChainActive().Tip()->height + 1,

ToByteVector(CDataStream{SER_NETWORK, PROTOCOL_VERSION, creationTx}));
                if (! Res. OK) {// Knownsec // judgment of processing result
                    throw JSONRPCError(RPC_INVALID_REQUEST, "Execution test failed:\n"
+ res.msg);
```

```
            }
        }
        return    signsend(rawTx,    request,    pwallet)->GetHash().GetHex();//knownsec//
signature
    }
```

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.10.3. Token update logic [Pass]

Audit note: Check if there is a problem with the Token update logic.

Risk and hazard: The node of the update Token operation cannot be in the Initial Block Download state. At the same time, it is necessary to judge whether the Token to be updated exists, judge whether the Token is a stable coin, and strictly verify the passed parameters, so as to avoid unexpected results.

Risk document:

Audit process: after detection, the token update operation logic in the common chain code has not found obvious security problems.

```
UniValue updatetoken(const JSONRPCRequest& request) {
    CWallet* const pwallet = GetWallet(request);
    .
     If (pwallet->chain(). IsInitialBlockDownload ()) {//knownsec// node cannot be in the
InitialBlockDownload state
        throw JSONRPCError(RPC_CLIENT_IN_INITIAL_DOWNLOAD,
                           "Cannot resign Masternode while still in Initial Block
Download");
```

```
    }
    pwallet->BlockUntilSyncedToCurrentChain();
   //knownsec//RPC parameter type checking
    RPCTypeCheck(request.params, {UniValue::VARR, UniValue::VOBJ}, true);
    UniValue metaObj = request.params[1].get_obj();
    std::string const tokenStr = trim_ws(metaObj["token"].getValStr());
    CTxDestination ownerDest;
    uint256 creationTx{};
    {
        LOCK(cs_main);
        DCT_ID id;
        auto token = pcustomcsview->GetTokenGuessId(tokenStr, id);
//knownsec//query token
        if (! (Token) {//knownsec// Judge whether token exists
            throw JSONRPCError(RPC_INVALID_PARAMETER, strprintf("Token %s does
not exist!" , tokenStr));
        }
        if (id < CTokensView::DCT_ID_START) {
            throw JSONRPCError(RPC_INVALID_PARAMETER, strprintf("Token %s is a
'stable coin'", tokenStr));
        }
        LOCK(pwallet->cs_wallet);
        auto tokenImpl = static_cast<CTokenImplementation const& >(*token);
        creationTx = tokenImpl.creationTx;
    }
    CMutableTransaction rawTx;
    for(std::set<CScript>::iterator it =
Params().GetConsensus().foundationMembers.begin();     it ! =
Params().GetConsensus().foundationMembers.end() && rawTx.vin.size() == 0;     it++)
    {
        if(IsMine(*pwallet, *it) == ISMINE_SPENDABLE)
        {
            CTxDestination destination;
            if (! ExtractDestination(*it, destination)) {
                throw JSONRPCError(RPC_INVALID_ADDRESS_OR_KEY, "Invalid
destination");
            }
            try {
                rawTx.vin = GetAuthInputs(pwallet, destination,
request.params[0].get_array()); //knownsec//gets the input information after the permission
check
            }
            catch (const UniValue& objError) {}
        }
    }
    if(rawTx.vin.size() == 0)
        throw JSONRPCError(RPC_INVALID_REQUEST, "Incorrect Authorization");

    CDataStream metadata(DfTxMarker, SER_NETWORK, PROTOCOL_VERSION);
    metadata << static_cast<unsigned char>(CustomTxType::UpdateToken)
                 << creationTx << metaObj["isDAT"].getBool();

    CScript scriptMeta;
    scriptMeta << OP_RETURN << ToByteVector(metadata);
    rawTx.vout.push_back(CTxOut(0, scriptMeta));
    rawTx = fund(rawTx, request, pwallet);
    // check execution
    {
        LOCK(cs_main);
        CCustomCSView mnview_dummy(*pcustomcsview);     // don't write into actual
DB
        const auto res =
ApplyUpdateTokenTx(mnview_dummy, ::ChainstateActive().CoinsTip(),
CTransaction(rawTx), ::ChainActive().Tip()->height + 1,
        ToByteVector(CDataStream{SER_NETWORK, PROTOCOL_VERSION, creationTx,
metaObj["isDAT"].getBool()}));
        if (! Res. OK) {// Knownsec // result judgment
            throw JSONRPCError(RPC_INVALID_REQUEST, "Execution test failed:\n" +
res.msg);
        }
    }
    return signsend(rawTx, request, pwallet)->GetHash().GetHex();
}
```

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.10.4. Token issuance logic [Pass]

Audit note: Check whether there is a problem with token additional logic.

Risk and hazard: The node of the Token operation for additional issuance cannot be in the Initial Block Download state. At the same time, it is necessary to judge whether the Token for additional issuance exists, judge whether the Token is a stable coin, and strictly verify the passed parameters, so as to avoid unexpected results.

Risk document:

Audit process: after detection, the logic of token additional operation in the common chain code has not found obvious security problems.

```
UniValue minttokens(const JSONRPCRequest& request) {
    CWallet* const pwallet = GetWallet(request);
    If (pwallet->chain(). IsInitialBlockDownload ()) {//knownsec//node cannot be in the
InitialBlockDownload state
        throw JSONRPCError(RPC_CLIENT_IN_INITIAL_DOWNLOAD,
          "Cannot resign Masternode while still in Initial Block Download");
    }
    pwallet->BlockUntilSyncedToCurrentChain();
    const CBalances minted = DecodeAmounts(pwallet->chain(), request.params[1], "");
    CMutableTransaction rawTx;
    // auth
    {
        for (auto const & kv : minted.balances) {
            if (kv.first < CTokensView::DCT_ID_START) {
                throw JSONRPCError(RPC_INVALID_PARAMETER, strprintf("Token %s is a
'stable coin', can't mint stable coin", kv.first.ToString()));
            }
            CTxDestination ownerDest;
            {
                LOCK(cs_main);
                auto token = pcustomcsview->GetToken(kv.first); //knownsec//access
token
                if (! token) {
                    throw JSONRPCError(RPC_INVALID_PARAMETER,
strprintf("Token %s does not exist!" , kv.first.ToString()));
                }
                auto tokenImpl = static_cast<CTokenImplementation const& >(*token);
                if (tokenImpl.destructionTx ! = uint256{}) {
                    throw JSONRPCError(RPC_INVALID_PARAMETER,
                                           strprintf("Token %s already destroyed at
height %i by tx %s", tokenImpl.symbol,
                                                     tokenImpl.destructionHeight,
tokenImpl.destructionTx.GetHex()));
                }
```

```
                    LOCK(pwallet->cs_wallet);
                    auto wtx = pwallet->GetWalletTx(tokenImpl.creationTx); //knownsec//
Access to wallet trading information
                    if (! wtx || ! ExtractDestination(wtx->tx->vout[1].scriptPubKey,
ownerDest)) {
                        throw JSONRPCError(RPC_INVALID_PARAMETER,
                                           strprintf("Can't extract destination for
token's %s collateral", tokenImpl.symbol));
                    }
                }
                rawTx.vin = GetAuthInputs(pwallet, ownerDest, request.params[0].get_array());
//knownsec//Obtain the input content after authorized inspection
            }
        }
        CDataStream metadata(DfTxMarker, SER_NETWORK, PROTOCOL_VERSION);
        metadata << static_cast<unsigned char>(CustomTxType::MintToken)
                 << minted;     /// @note here, that whole CBalances serialized! , not a
'minted.balances'!

        CScript scriptMeta;
        scriptMeta << OP_RETURN << ToByteVector(metadata);

        rawTx.vout.push_back(CTxOut(0, scriptMeta));

        // fund
        rawTx = fund(rawTx, request, pwallet);

        // check execution
        {
            LOCK(cs_main);
            CCustomCSView mnview_dummy(*pcustomcsview);     // don't write into actual
DB
```

```
            const auto res = ApplyMintTokenTx(mnview_dummy, g_chainstate->CoinsTip(),
CTransaction(rawTx),

ToByteVector(CDataStream{SER_NETWORK, PROTOCOL_VERSION, minted }));
            if (! Res.ok) {// Knownsec // check results
                throw JSONRPCError(RPC_INVALID_REQUEST, "Execution test failed:\n" +
res.msg);
            }
      }
      return signsend(rawTx, request, pwallet)->GetHash().GetHex(); //knownsec//ignature
}
```

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.10.5.Coin pair creation logic[Pass]

Audit note: Check if there are any problems with the transaction's creation logic.

Risk and harm: The node creating the transaction pair operation cannot be in the Initial Block Download state, and the Token transaction pair of the perforator needs to be traded and the incoming parameters and address, as well as the IDENTITY of THE RPC caller needs to be strictly checked to avoid unauthorized RPC calls and unexpected results.

Risk document:

Audit process: after detection, no obvious security problems were found in the logic of creating transactions in the common chain code.

```
UniValue createpoolpair(const JSONRPCRequest& request) {
    CWallet* const pwallet = GetWallet(request);
  .
     If (pwallet->chain(). IsInitialBlockDownload ()) {//knownsec//node cannot be in the
InitialBlockDownload state
         throw JSONRPCError(RPC_CLIENT_IN_INITIAL_DOWNLOAD, "Cannot create
transactions while still in Initial Block Download");
     }
     RPCTypeCheck(request.params, {UniValue::VOBJ, UniValue::VARR}, true);
//knownsec//Input type check
     std::string tokenA, tokenB, pairSymbol;
     CAmount commission;
     bool status = true;    // default Active
     UniValue paginationObj = request.params[0].get_obj();
     if (! PaginationObj ["tokenA"]. IsNull ()) {//knownsec//checks tokenA for NULL
         tokenA = paginationObj["tokenA"].get_str();
     }
     if (! PaginationObj ["tokenB"]. IsNull ()) {//knownsec//checks tokenB for NULL
         tokenB = paginationObj["tokenB"].get_str();
     }
     if (! paginationObj["comission"].isNull()) {
         commission = paginationObj["comission"].get_int64();
     }
     if (! paginationObj["status"].isNull()) {
         status = paginationObj["status"].get_bool();
     }
     if (! PaginationObj ["pairSymbol"]. IsNull ()) {//knownsec//Check if the trade pair exists
         pairSymbol = paginationObj["pairSymbol"].get_str();
     }
     DCT_ID idtokenA, idtokenB;
     {
         LOCK(cs_main);
         auto token = pcustomcsview->GetTokenGuessId(tokenA, idtokenA);
```

```
            if (token) {
            }
            else
                throw JSONRPCError(RPC_INVALID_ADDRESS_OR_KEY, "TokenA was not
found");
            auto token2 = pcustomcsview->GetTokenGuessId(tokenB, idtokenB);
            if (token2) {
            }
            else
                throw JSONRPCError(RPC_INVALID_ADDRESS_OR_KEY, "TokenB was not
found");
        }
        CPoolPairMessage poolPairMsg;
        poolPairMsg.idTokenA = idtokenA;
        poolPairMsg.idTokenB = idtokenB;
        poolPairMsg.commission = commission;
        poolPairMsg.status = status;
        poolPairMsg.pairSymbol = pairSymbol;
        CDataStream metadata(DfTxMarker, SER_NETWORK, PROTOCOL_VERSION);
        metadata << static_cast<unsigned char>(CustomTxType::CreatePoolPair)
                  << poolPairMsg;

        CScript scriptMeta;
        scriptMeta << OP_RETURN << ToByteVector(metadata);

        CMutableTransaction rawTx;
        for(std::set<CScript>::iterator it =
Params().GetConsensus().foundationMembers.begin();    it ! =
Params().GetConsensus().foundationMembers.end() && rawTx.vin.size() == 0;    it++)
        {
            if(IsMine(*pwallet, *it) == ISMINE_SPENDABLE)
            {
                CTxDestination destination;
                if (! ExtractDestination(*it, destination)) {
                     throw JSONRPCError(RPC_INVALID_ADDRESS_OR_KEY, "Invalid
destination");
                }
                try {
                    rawTx.vin = GetAuthInputs(pwallet, destination,
request.params[0].get_array());
                }
                catch (const UniValue& objError) {}
            }
      }
      if(rawTx.vin.size() == 0)
            throw JSONRPCError(RPC_INVALID_REQUEST, "Incorrect Authorization");
      rawTx = fund(rawTx, request, pwallet);
      // check execution
      {
            LOCK(cs_main);
            CCustomCSView mnview_dummy(*pcustomcsview);    // don't write into actual
DB
            const auto res = ApplyCreatePoolPairTx(mnview_dummy,
g_chainstate->CoinsTip(), CTransaction(rawTx), ::ChainActive().Tip()->height + 1,
                                                 ToByteVector(CDataStream{SER_NETWORK,
PROTOCOL_VERSION, poolPairMsg}));
            if (! Res.ok) {//knownsec//check results
                throw JSONRPCError(RPC_INVALID_REQUEST, "Execution test failed:\n" +
res.msg);
            }
      }
      return signsend(rawTx, request, pwallet)->GetHash().GetHex(); //knownsec//signature
}
```

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.10.6.Currency pair transaction logic [Pass]

Audit note: Check the liquidity pool token exchange logic for any problems.

Risk and harm: When conducting token pair transactions, authorization transaction shall be carried out first, and the existence of token and transaction pair shall be strictly verified. At the same time, safe numerical operation shall be carried out on operation objects when increasing or decreasing tokens, for example;Do overflow check and so on.

Risk document:

Audit process: no obvious security problems were found in the logic of increasing liquidity in the common chain code after detection.

```
Res ApplyPoolSwapTx(CCustomCSView &mnview, const CCoinsViewCache &coins, const
CTransaction &tx, uint32_t height, const std::vector<unsigned char> &metadata)
{
    CPoolSwapMessage poolSwapMsg;
    std::string pairSymbol;
    CDataStream ss(metadata, SER_NETWORK, PROTOCOL_VERSION);
    ss >> poolSwapMsg;
    if (! ss.empty()) {
        return Res::Err("PoolSwap: deserialization failed: excess %d bytes",   ss.size());
    }
    const std::string base{"PoolSwap creation: " + poolSwapMsg.ToString()};

    // check auth
    if (! HasAuth(TX, COINS, Poolswapmsg. from) {//knownsec//Permissions check
        return Res::Err("%s: %s", base, "tx must have at least one input from account owner");
    }
    auto tokenFrom = mnview.GetToken(poolSwapMsg.idTokenFrom);
    if (! TokenFrom) {//knownsec//token existence judgment
        throw Res::Err("%s: token %s does not exist!" , base,
poolSwapMsg.idTokenFrom.ToString());
    }
    auto tokenTo = mnview.GetToken(poolSwapMsg.idTokenTo);
    if (! TokenTo) {//knownsec//token existence judgment
        throw Res::Err("%s: token %s does not exist!" , base,
poolSwapMsg.idTokenTo.ToString());
    }
    auto poolPair = mnview.GetPoolPair(poolSwapMsg.idTokenFrom,
poolSwapMsg.idTokenTo);
    if (! PoolPair) {//knownsec//Trading pairs exist
        throw Res::Err("%s: didn't find the poolpair!" , base);
    }
    CPoolPair pp = poolPair->second;
    const auto res = pp.Swap({poolSwapMsg.idTokenFrom, poolSwapMsg.amountFrom},
poolSwapMsg.maxPrice, [&] (const CTokenAmount &tokenAmount) {
        auto resPP = mnview.SetPoolPair(poolPair->first, pp);//knownsec//Swap deal with it
        if (! resPP.ok) {
            return Res::Err("%s: %s", base, resPP.msg);
        }
        auto sub = mnview.SubBalance(poolSwapMsg.from, {poolSwapMsg.idTokenFrom,
poolSwapMsg.amountFrom});
        if (! sub.ok) {
            return Res::Err("%s: %s", base, sub.msg);
        }
        auto add = mnview.AddBalance(poolSwapMsg.to, tokenAmount);
        if (! add.ok) {
                return Res::Err("%s: %s", base, add.msg);
            }
            return Res::Ok();
        });
        if (! Res.ok) {//knownsec//check results
            return Res::Err("%s: %s", base, res.msg);
        }
        return Res::Ok();
}
```

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.10.7.Increase liquidity logic [Pass]

Audit note: Check to see if there is a problem with the logic of increasing liquidity.

Risk and harm: The node that increases the liquidity operation cannot be in the Initial Block Download state, and the incoming parameters, address, RPC caller identity and so on need to be strictly verified when increasing the liquidity, so as to avoid unexpected results and RPC unauthorized malicious calls.

Risk document:

Audit process: no obvious security problems were found in the logic of increasing liquidity in the common chain code after detection.

```
UniValue addpoolliquidity(const JSONRPCRequest& request) {
        CWallet* const pwallet = GetWallet(request);

                        .
        If (pwallet->chain(). IsInitialBlockDownload ()) {//knownsec//node cannot be at
InitialBlockDownload
            throw JSONRPCError(RPC_CLIENT_IN_INITIAL_DOWNLOAD, "Cannot create
transactions while still in Initial Block Download");
        }
       //knownsec//RPC input type check
        RPCTypeCheck(request.params, { UniValue::VOBJ, UniValue::VARR }, true);
        if (request.params[0].isNull()) {
            throw JSONRPCError(RPC_INVALID_PARAMETER, "Invalid parameters, at least
argument 1 must be non-null");
        }

        UniValue metaObj = request.params[0].get_obj();

        std::string shareAddress = metaObj["shareAddress"].getValStr();
        CTxDestination collateralDest = DecodeDestination(shareAddress);
        If (alldest. Which () == 0) {//knownsec//parameter check
            throw JSONRPCError(RPC_INVALID_PARAMETER, "shareAddress (" +
shareAddress + ") does not refer to any valid address");
        }

        CPoolPair poolPair;
        // fill here
        CMutableTransaction rawTx;
        CDataStream metadata(DfTxMarker, SER_NETWORK, PROTOCOL_VERSION);
        metadata << static_cast<unsigned char>(CustomTxType::AddPoolLiquidity)
                 << poolPair;
        CScript scriptMeta;
        scriptMeta << OP_RETURN << ToByteVector(metadata);
        // // TODO block
        for(std::set<CScript>::iterator it =
Params().GetConsensus().foundationMembers.begin();   it ! =
Params().GetConsensus().foundationMembers.end() && rawTx.vin.size() == 0;   it++)
        {
            if(IsMine(*pwallet, *it) == ISMINE_SPENDABLE)
            {
                CTxDestination destination;
                if (! ExtractDestination(*it, destination)) {
                    throw JSONRPCError(RPC_INVALID_ADDRESS_OR_KEY, "Invalid
destination");
                }
                try {
                    rawTx.vin = GetAuthInputs(pwallet, destination,
request.params[0].get_array()); //knownsec//Gets authenticated input
                }
                catch (const UniValue& objError) {}
            }
        }
        if(rawTx.vin.size() == 0)
            throw JSONRPCError(RPC_INVALID_REQUEST, "Incorrect Authorization");

        // // TODO end block
        rawTx.vout.push_back(CTxOut(0, scriptMeta));
        // fund
        rawTx = fund(rawTx, request, pwallet);
        // check execution
        {
            LOCK(cs_main);
            CCustomCSView mnview_dummy(*pcustomcsview);   // don't write into actual
DB
```

```
                // const auto res = ApplyAddPoolLiquidityTx(mnview_dummy,
CTransaction(rawTx), ToByteVector(CDataStream{SER_NETWORK, PROTOCOL_VERSION,
poolPair}));
                // if (! Res.ok) {// Knownsec // result check
                //        throw JSONRPCError(RPC_INVALID_REQUEST, "Execution test failed:\n" +
res.msg);
                / /}
           }
           return signsend(rawTx, request, pwallet)->GetHash().GetHex(); //knownsec//signature
     }
```

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.10.8.Remove liquidity logic [Pass]

Audit note: Check to see if there is a problem with the removal liquidity logic.

Risk and harm: The node that removed the liquidity operation cannot be in the Initial Block Download state, and the incoming parameters and the existence of RPC caller permission should be strictly checked when reducing the liquidity, so as to avoid unexpected results and RPC unauthorized malicious calls.

Risk document:

Audit process: after detection, no obvious security problems were found when the logic of liquidity operation was removed from the common chain code.

```
    UniValue removepoolliquidity(const JSONRPCRequest& request) {
    CWallet* const pwallet = GetWallet(request);
    .
        If (pwallet->chain(). IsInitialBlockDownload ()) {//knownsec//node cannot be at
InitialBlockDownload
                throw JSONRPCError(RPC_CLIENT_IN_INITIAL_DOWNLOAD, "Cannot create
transactions while still in Initial Block Download");
            }
            //knownsec//RPC input type check
            RPCTypeCheck(request.params, { UniValue::VOBJ, UniValue::VARR }, true);

            // decode
            CRemoveLiquidityMessage msg{};
            msg.from = DecodeRecipients(pwallet->chain(), request.params[0].get_obj());

            // encode
            CDataStream markedMetadata(DfTxMarker, SER_NETWORK, PROTOCOL_VERSION);
            markedMetadata << static_cast<unsigned char>(CustomTxType::AddPoolLiquidity)
                           << msg;
            CScript scriptMeta;
            scriptMeta << OP_RETURN << ToByteVector(markedMetadata);

            CMutableTransaction rawTx;
            rawTx.vout.push_back(CTxOut(0, scriptMeta));

            CTxDestination ownerDest;
            if (! request.params[2].get_array().empty()) {
                rawTx.vin = GetAuthInputs(pwallet, ownerDest, request.params[2].get_array());
//knownsec//Get access to permissions checked entry
            } else {
                for (const auto& kv : msg.from) {
                    if (! ExtractDestination(kv.first, ownerDest)) {
```

```
                    throw JSONRPCError(RPC_INVALID_ADDRESS_OR_KEY, "Invalid owner
destination");
                }
                std::vector<CTxIn> rawIn = GetAuthInputs(pwallet, ownerDest,
UniValue(UniValue::VARR));
                rawTx.vin.insert(rawTx.vin.end(), rawIn.begin(), rawIn.end());
            }
        }
        // fund
        rawTx = fund(rawTx, request, pwallet);
        // check execution
        {
            LOCK(cs_main);
            CCustomCSView mnview_dummy(*pcustomcsview);    // don't write into actual
DB
            const auto res = ApplyRemovePoolLiquidityTx(mnview_dummy,
g_chainstate->CoinsTip(), CTransaction(rawTx), ::ChainActive().Height() + 1,
ToByteVector(CDataStream{SER_NETWORK, PROTOCOL_VERSION, msg}));
            if (! Res.ok) {//knownsec//result check
                throw JSONRPCError(RPC_INVALID_REQUEST, "Execution test failed:\n" +
res.msg);
            }
        }
        return signsend(rawTx, request, pwallet)->GetHash().GetHex(); //knownsec//signature
    }
```

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.11. The account system

4.11.1. Account system authentication [Pass]

Audit note: Check if there is a problem with the authentication logic.

Risk and harm: forge identity to bypass authority check, or seriously threaten the security of assets.

Risk document:

Audit process: No obvious security problem was found in the authorization operation logic of account wallet system in the public chain code after detection.

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.11.2. Account CRUD logic [Pass]

Audit notes: Check if the CRUD logic of the account system has illegal boundary conditions, and when the operation takes effect, test whether there are vulnerabilities and the impact on nodes and data.

Risks and hazards: Improper setting of CURD logical permissions will lead to malicious operations, such as security risks such as asset loss caused by malicious deletion of accounts.

Risk document:

Audit process: No obvious security problem was found in the authorization operation logic of account wallet system in the public chain code after detection.

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.12. Other

4.12.1.Database security [Pass]

Audit note: Check all database statements for injection points, test for injection points and the impact they can have.Test the database operation method for logical errors.

Risk and harm: tampering with the database or, in severe cases, remote command execution.

Risk document:

Audit process: After detection, the logic related to database operation in the public chain code referred to the official standard of Bitcoin and no obvious security problems were found.

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.12.2.Message processing security [Pass]

Audit note: Check the deserialization of messages and other operations.

Risk and harm: remote command execution, occupying local storage resources, resulting in node program crash, etc.

Risk document:

Audit process: After detection, the logic related to message processing operation in the public chain code referred to the Bitcoin standard and no obvious security problems were found.

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.12.3.Thread safety [Pass]

Audit description: analyze multi-threaded Shared resources and check the locking status of Shared resources.

Risk and harm: data competition, atomic destruction of things, node process crash, etc.

Risk document:

Audit results: No concurrent security-related risks were found after audit.

Safety advice: None.

4.12.4.Security of environment variables [Pass]

Audit notes: Check dependencies on environment variables and test the impact of controlled environment variables on node programs.

Risk and harm: affect the safety of funds, etc., according to the specific situation analysis.

Risk document:

Audit process: no obvious security problems were found in the logic related to environment variables in the common chain code.

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.12.5. Centralization degree Detection [Pass]

Audit note: Identify if there is over-centralization in system design.

Risks and hazards: Centralized design tends to centralize risks. Once a single point of failure occurs, serious problems such as data tampering and chain bifurcation may occur.

Risk document:

Audit process: no related security problems were found in the centralized logic in the common chain code after detection.

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.12.6. File permission security [Pass]

Audit note: Check that the relevant log files and keystore file permissions are set correctly.

Risk and harm: File permission is one of the security Settings at the bottom level of the system, which ensures that the file can be used by users to carry out corresponding operations. Improper file permission setting may lead to risks such as file tampering and information leakage.

Risk document:

Audit process: after detection, there is no security problem in the file permission setting in the public chain code.

Audit results: No relevant risks were found after audit.

Safety advice: None.

4.12.7.Concurrent security risks [Pass]

Audit note: Check whether there is a concurrent security risk for operations such as arrays in common chain code.

Risks and hazards: when multiple threads share data, they may produce inconsistent results, which may cause data loss, value overwrite, and other problems.

Risk document:

Audit process: no obvious security problems were detected.

Audit results: No relevant risks were found after audit.

Safety advice: None.

5. Appendix B: Vulnerability risk rating criteria

Blockchain system vulnerability risk rating criteria	
Vulnerability rating	**Vulnerability rating description**
Serious vulnerabilities	Forgery of computing power, centralization of computing power, serious can lead to 51% attack; Vulnerabilities can lead to the disclosure of private keys, affecting the security of assets; It can lead to bifurcation of the common chain by some means. The use of local storage resources DoS node, in serious cases, can lead to node program crash, forge blocks, generate unexpected income, in serious cases, can lead to chain bifurcation, data tampering and other problems; Duplicate trading in the same chain or across the chain, forge legal transactions, affecting the security of assets, double flow of assets in the same chain, trade replay, intelligent bypass the sandbox restrictions, in serious cases can affect the execution logic of other contracts, or cause DoS and remote command execution; Unexpected behavior occurs in the execution of the contract. In severe cases, it may cause data tampering or node program crash, forge identity to bypass authority check, or threaten asset security in severe cases. The current stage is a large-scale outbreak should cause enough attention to security vulnerabilities.
High risk vulnerabilities	Can seriously affect the operation of the business through certain means, may bring huge losses to the customer, such as the key business RPC login

	interface can be hit library, business system can be easily collected wool, etc. The vulnerability that requires user interaction to obtain user identity information. Any file can be read, modified, overwritten, deleted and downloaded. Circumventing restrictions to modify the user's personal data, personal information, or force the user to perform certain operations; Serious information leakage, backup of sensitive information files or source code leakage that can harm the target system (such as storage key leakage, database connection password leakage, SVN/Git account leakage, VPN account leakage, etc.). Vulnerabilities that can directly cause losses of token contracts or users' funds, such as: numerical overflow loopholes that can cause the value of token to return to zero, false charging loopholes that can cause losses of tokens in exchanges, or ETH or re-entry loopholes in contract accounts; Vulnerabilities that can cause the loss of escrow rights of token contracts, such as access control defects of key functions, access control bypass of key functions caused by call injection, etc. Vulnerabilities that cause token contracts to not work properly, such as the denial of service vulnerability caused by sending the ETH to a malicious address, or the denial of service vulnerability caused by running out of gas.
In the dangerous holes	Which can affect the security of the target system but cannot be directly verified to be exploitable; Those that can obtain the authorization of the target network device but determine that they cannot be further utilized; Non-important information leakage, CORS vulnerability of non-core critical business operations, etc.

	High-risk vulnerabilities that require a specific address to trigger, such as numerical overflow vulnerabilities that can only be triggered by the owner of a token contract; Access control defects of non-critical functions, logical design defects that cannot cause direct capital loss, etc.
Low latent loophole	For example, the management port of the target system is open to the public network but cannot be directly used, the Banner information of the target system can be identified, and other penetration testers identify vulnerabilities that are difficult to use but may have potential security threats. Vulnerabilities that are difficult to be triggered, vulnerabilities that have limited harm after being triggered, such as numerical overflow vulnerabilities that require a large number of ETH or tokens to be triggered, vulnerabilities that the attacker cannot directly profit after triggering numerical overflow, and transaction sequence dependence risks triggered by specifying high gas, etc.

6. Appendix C: Blockchain system risk rating criteria

Public chain system Risk Rating standard (Union chain, private chain risk level downgraded one level)	
Systemic risk rating	**Vulnerability rating description**
Severe risk system	A system with 1 or more serious vulnerabilities or 2 or more high-risk vulnerabilities
High risk system	Systems with 1 or more high-risk vulnerabilities, or 3 or more medium-risk vulnerabilities
Medium risk system	A system with 1 or more medium-risk vulnerabilities or 5 or more low-risk vulnerabilities

Low risk system	Systems with 3 to 5 low-risk vulnerabilities
Security system	3 systems with low risk vulnerabilities or no vulnerabilities

7. Appendix D: Introduction to vulnerability testing tools

7.1. Gosec

By scanning the Go Gosec AST check whether source code security problems, you can scan for hard-coded proof, not check the audit error, using did not escape the data in HTML templates, use a predictable path to create temporary file, extract the zip archive file security problem such as traversal, rules can be configured to run only a subset, exclude certain path to the file, and generate different format of the report.

7.2. Snyk

A developer-first solution that automatically finds and fixes known vulnerabilities in dependencies.

7.3. Flawfinder

Flawfinder is an open source tool for Static scan analysis of C/C++. It conducts static search according to the internal dictionary database and matches simple defects and vulnerabilities. Instead of compiling C/C++ code, it can be scanned and analyzed directly using Flawfinder. Simple and fast, the biggest advantage is free, no need to compile.

7.4. SonarQube

SonarQube provides an overview of the overall health of the source code. More importantly, it highlights problems found in the new code. Using the quality gate setup in the project, you will be able to easily fix bugs and mechanically improve the code.

7.5. Yasca

Yasca is a source code analysis tool that was developed in 2007 and is designed to be flexible and easy to extend.

7.6. Safesql

Golang static analysis tool, can detect SQL injection and other database security issues.

7.7. CFSSL

CFSSL is CloudFlare's PKI/TLS Swiss Army Knife. It is both a command line tool and an HTTP API server for signing, validating, and binding TLS certificates.

7.8. Sublime Text-IDE

Sublime Text supports the syntax highlighting of multiple programming languages, has excellent code completion, and has the capability of snippets to save commonly used code snippets and call them whenever needed. VIM mode is supported and you can use most of the commands in VIM mode. To support macros, you simply record an operation or write your own command, and then play back the operation or command you just recorded.

7.9. Know Knownsec penetration tester kit

The tool kit, developed, collected and used by The penetration testing engineer of KnowKnownsec, contains batch automated testing tools, independently developed tools, scripts or utilization tools, etc. for testers.

KNOWNSEC

Beijing KnownSec Information Technology Co., Ltd.

Advisory telephone	+86(10)400 060 9587
E-mail	sec@knownsec.com
Website	www.knownsec.com
Address	wangjing soho T2-B2509,Chaoyang District, Beijing