KNOWNSEC

Degree of secrecy ★ | Project team open

Public Blockchain Audit Report

Security status

Safe

★ ★ ★ ★ ★



Principal tester: Knownsec blockchain security team

Version Summary

Content	Date	Version
Editing Document	2021/03/19	V1.0

Report Information

Title	Version	Document Number	Type
BOHR Public Blockchain Audit Report	V1.0	ea389fced63a46bd8d82e83565dd6115	Open to project team

Copyright Notice

Knownsec only issues this report for facts that have occurred or existed before the issuance of this report, and assumes corresponding responsibilities for this. Knownsec is unable to determine the security status of its Public Blockchain and is not responsible for the facts that will occur or exist in the future. The security audit analysis and other content made in this report are only based on the documents and information provided to us by the information provider as of the time this report is issued. Knownsec's assumption: There is no missing, tampered, deleted or concealed information. If the information provided is missing, tampered with, deleted, concealed or reflected in the actual situation, Knownsec shall not be liable for any losses and adverse effects caused thereby.

Table of Contents

1. Introduction ..... - 7 -

2. Code vulnerability analysis ..... - 8 -

2.1. Vulnerability Level Distribution ..... - 8 -

2.2. Vulnerability type distribution ..... - 8 -

3. Project Analysis ..... - 9 -

3.1. Project Introduction ..... - 9 -

3.2. Audit scope ..... - 10 -

3.3. Directory structure ..... - 10 -

3.4. Startup process ..... - 24 -

3.5. P2P related methods ..... - 25 -

3.6. API core interface list ..... - 25 -

3.7. Third-party library reference list ..... - 27 -

4. Audit Results ..... - 29 -

4.1. Cryptography ..... - 29 -

4.1.1. Random number range and probability distribution【PASS】 ..... - 29 -

4.1.2. Implementation/use of cryptographic algorithms【PASS】 ..... - 30 -

4.2. RPC ..... - 31 -

4.2.1. Sensitive interface permissions【PASS】 ..... - 31 -

4.2.2. Business logic【PASS】 ..... - 32 -

4.2.3. Traditional Web Security【PASS】 ..... - 35 -

knownsec

4.3. Key security ..... - 36 -
4.3.1. Private key/mnemonic word generation algorithm【PASS】 ..... - 36 -
4.3.2. Private key/mnemonic phrase storage in plain text【PASS】 ..... - 36 -
4.3.3. Use traces of private keys/mnemonics【PASS】 ..... - 37 -
4.3.4. Residual private key/mnemonic phrase【PASS】 ..... - 38 -
4.3.5. Misuse of private key/mnemonic phrase【PASS】 ..... - 38 -
4.4. Consensus mechanism ..... - 39 -
4.4.1. Implementation of consensus verification【PASS】 ..... - 39 -
4.4.2. Consensus mechanism design【PASS】 ..... - 45 -
4.4.3. Voting verification logic【PASS】 ..... - 51 -
4.4.4. Proposal verification logic【PASS】 ..... - 52 -
4.5. P2P ..... - 55 -
4.5.1. Differential response to sensitive information【PASS】 ..... - 55 -
4.5.2. Fuzz malformed data test【PASS】 ..... - 57 -
4.5.3. Node Discovery Algorithm【PASS】 ..... - 59 -
4.5.4. Node communication protocol【PASS】 ..... - 60 -
4.5.5. Routing table pollution【PASS】 ..... - 60 -
4.5.6. Traffic Limit【PASS】 ..... - 61 -
4.5.7. Node penalty mechanism【PASS】 ..... - 62 -
4.6. Chain processing ..... - 62 -
4.6.1. Longest chain selection and switching algorithm【PASS】 ..... - 62 -
4.6.2. Chain synchronization logic【PASS】 ..... - 63 -

Knownsec

4.7. Block processing ..... - 63 -
4.7.1. Block hash collision【PASS】 ..... - 63 -
4.7.2. Block processing resource limit【PASS】 ..... - 64 -
4.7.3. Block signature processing【PASS】 ..... - 65 -
4.7.4. Block verification processing【PASS】 ..... - 70 -
4.7.5. Merkle tree root construction method【PASS】 ..... - 76 -
4.8. Transaction processing ..... - 77 -
4.8.1. Transaction signature【PASS】 ..... - 77 -
4.8.2. Transaction processing logic【PASS】 ..... - 79 -
4.8.3. Transaction processing resource limit【PASS】 ..... - 83 -
4.8.4. Transaction fee setting【PASS】 ..... - 84 -
4.9. Contract virtual machine ..... - 85 -
4.9.1. Contract virtual machine sandbox escape【PASS】 ..... - 85 -
4.9.2. Contract execution logic【PASS】 ..... - 86 -
4.10. Account system ..... - 91 -
4.10.1. Account system authentication【PASS】 ..... - 91 -
4.10.2. Account password policy【Low】 ..... - 91 -
4.10.3. Account CRUD logic【PASS】 ..... - 93 -
4.11. *other* ..... - 94 -
4.11.1. Database Security【PASS】 ..... - 94 -
4.11.2. Message Processing Security【PASS】 ..... - 94 -
4.11.3. Thread safety【PASS】 ..... - 95 -

Knownsec

4.11.4. Security of environment variables【PASS】 ..... - 95 -

4.11.5. Centralization degree detection【PASS】 ..... - 96 -

4.11.6. File permission security【PASS】 ..... - 96 -

4.11.7. Concurrency security risks【PASS】 ..... - 97 -

4.11.8. Third-party dependent libraries【Low】 ..... - 98 -

Knownsec

1. Introduction

The effective test time of this report is from March 8, 2021 to March 19, 2021. During this period, the security and standardization of the BOHR public chain will be audited and used as the statistical basis for the report.。

Knownsec blockchain system security test method：

White-box testing

Through the source code of the program, the SDK conducts security audits, and conducts dynamic debugging and vulnerability mining on node p2p and RPC.

Gray-box testing

Security scanning and auditing of source code through tools, looking for vulnerabilities that may cause anomalies

Black-box testing

Simulate an attacker to conduct a security test attack on the node to check whether the node responds normally.

Through this code audit, we know that Chuangyu engineers found that the target code has 0 common security risks and 0 security problems in total, including 0 serious-risk code files or functions, 0 high-risk code files or functions, and 0 medium-risk codes files or functions, and 2 low-risk code files or functions.

Since this testing process is carried out in a non-production environment, all codes are up-to-date backups, the testing process is communicated with the relevant interface people, and relevant testing operations are carried out under the controllable operational risk to avoid production during the testing process Operational risk, code security risk.

Report information of this audit:

Report Number：ea389fced63a46bd8d82e83565dd6115

Report query address link:

https://attest.im/attestation/searchResult?qurey=ea389fced63a46bd8d82e83565dd6115

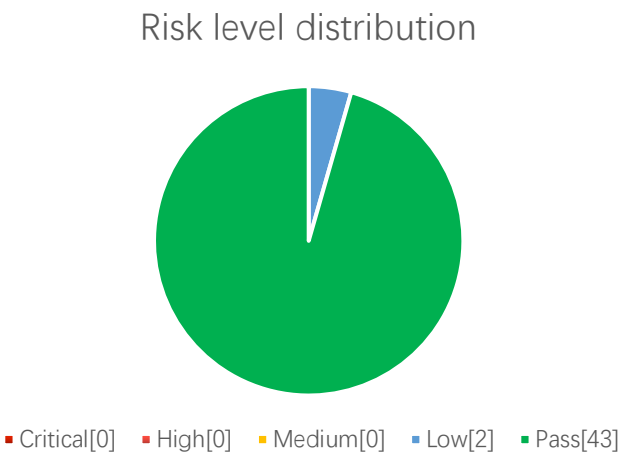
Knownsec

2. Code vulnerability analysis

2.1. Vulnerability Level Distribution

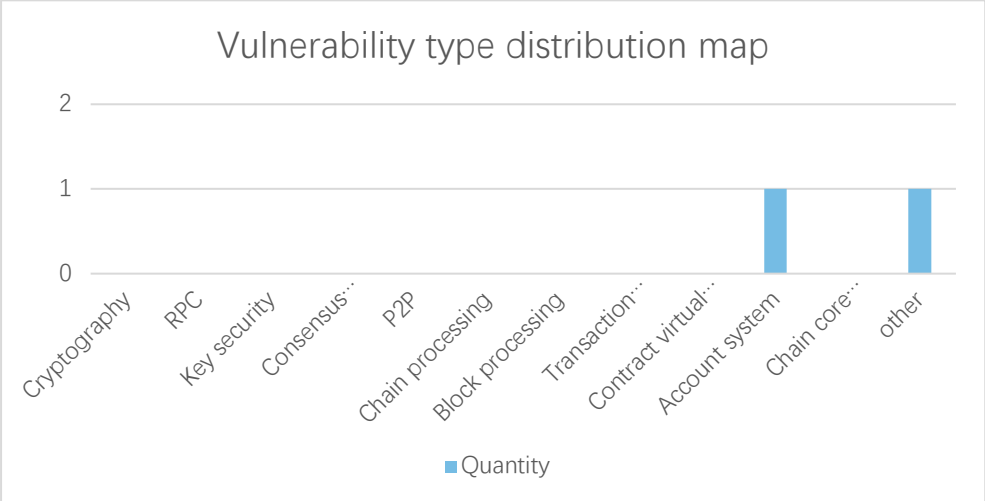
Vulnerability risk statistics by level：

Vulnerability risk level statistics table			
Serious	**High**	**Medium**	**Low**
0	**0**	**0**	**2**



2.2. Vulnerability type distribution

This vulnerability is counted by type：



3. Project Analysis

3.1. Project Introduction

Bohr is a global blockchain open source community project, a customizable blockchain infrastructure composed of microkernels and functional modules. Bohr has developed a set of industrialized blockchain technology standards, following the principles of hot plugging, modularization, and parallel expansion, providing alternative smart contracts, multi-chain parallelism, consensus mechanisms, P2P networks, storage, encryption, multi-level accounts and other functions Module.

Based on the simple basic main chain of Bohr, a wealth of modular components will be provided. Just like the assembly of CPU, hard disk, memory, and motherboard into a computer, users can freely choose consensus, storage, contract, arbitration, account system, anonymity strategy, and authority. Wait for the modules and assemble them into a sub-chain that suits your needs. The sub-chain is not limited to the public chain, but can also be a private chain or a consortium chain.

After market research and analysis, we found that there are some problems in the development of blockchain. The scarcity of blockchain technical talents and high R&D costs cannot be alleviated in a short time; more and more application scenarios require blocks. The support of chain technology; the performance of existing blockchains is limited, and communication between different chains is not possible; organizations will tend to use alliance chains and private chains, and the trust between the two is incomplete. Bohr can provide a reliable solution for this.

knownsec

Official website address: https://www.bohrweb.org/

Main coding language: JAVA

3.2. Audit scope

The main types and scope of security audits include:

Encoding security

RPC security

P2P security

Consensus security

Account system security

Contract virtual machine security

Business logic design security

3.3. Directory structure

Core code directory structure：

```
├──bohr-core
|    |    .swagger-codegen-ignore
|    |    pom.xml
|    ├──config
|    |         bohr.properties
|    |         ipfilter.json
|    |         log4j2.xml
|    ├──misc
|    |    |    formatter_eclipse.xml
|    |    |    formatter_intellij.xml
```

Knownsec

```
|   |   |   license_header.txt
|   |   └──systemd
|   |           bohr.service.app
|   └──src
|       └──main
|           ├──java
|           |   └──org
|           |       └──bohr
|           |           |   JvmOptions.java
|           |           |   Kernel.java
|           |           |   Launcher.java
|           |           |   Main.java
|           |           |   Network.java
|           |           |
|           |           ├──api
|           |           |   |   ApiHandler.java
|           |           |   |   ApiHandlerImpl.java
|           |           |   |   ApiVersion.java
|           |           |   |   BohrApiService.java
|           |           |   |
|           |           |   ├──http
|           |           |   |       HttpChannelInitializer.java
|           |           |   |       HttpConstants.java
|           |           |   |       HttpHandler.java
|           |           |   |
|           |           |   ├──util
|           |           |   |       TransactionBuilder.java
|           |           |   |
|           |           |   └──v2
|           |           |       |   BohrApiImpl.java
|           |           |       |   TypeFactory.java
|           |           |       |
|           |           |       ├──client
```

knownsec

```
|   |   |   |   |   BohrApi.java
|   |   |   |   |
|   |   |   |   └──server
|   |   |   |       BohrApi.java
|   |   ├──cli
|   |   |   BohrCli.java
|   |   |   BohrOption.java
|   |   |   DatabaseScanner.java
|   |   ├──config
|   |   |   |   AbstractConfig.java
|   |   |   |   ChainSpec.java
|   |   |   |   Config.java
|   |   |   |   Constants.java
|   |   |   |   DevnetConfig.java
|   |   |   |   MainnetConfig.java
|   |   |   |   TestnetConfig.java
|   |   |   |
|   |   |   └──exception
|   |   |       ConfigException.java
|   |   ├──consensus
|   |   |   |   BohrBft.java
|   |   |   |   BohrSync.java
|   |   |   |   Proof.java
|   |   |   |   Proposal.java
|   |   |   |   Vote.java
|   |   |   |   VoteSet.java
|   |   |   |   VoteType.java
|   |   |   |
|   |   |   └──exception
|   |   |       BohrBftException.java
|   |   ├──core
|   |   |   |   ActivatedForks.java
|   |   |   |   Amount.java
```

knownsec

```
|   |   |   |   BftManager.java
|   |   |   |   Block.java
|   |   |   |   Blockchain.java
|   |   |   |   BlockchainImpl.java
|   |   |   |   BlockchainListener.java
|   |   |   |   BlockHeader.java
|   |   |   |   BlockHeaderData.java
|   |   |   |   BlockPart.java
|   |   |   |   Fork.java
|   |   |   |   ForkSignalSet.java
|   |   |   |   Genesis.java
|   |   |   |   PendingManager.java
|   |   |   |   SyncManager.java
|   |   |   |   Transaction.java
|   |   |   |   TransactionExecutor.java
|   |   |   |   TransactionResult.java
|   |   |   |   TransactionType.java
|   |   |   |   Unit.java
|   |   |   |   Wallet.java
|   |   |   |
|   |   |   |   ├──event
|   |   |   |   |       BlockchainDatabaseUpgradingEvent.java
|   |   |   |   |       WalletLoadingEvent.java
|   |   |   |   ├──exception
|   |   |   |   |       BlockchainException.java
|   |   |   |   |       WalletLockedException.java
|   |   |   |   └──state
|   |   |   |           Account.java
|   |   |   |           AccountState.java
|   |   |   |           AccountStateImpl.java
|   |   |   |           Delegate.java
|   |   |   |           DelegateState.java
|   |   |   |           DelegateStateImpl.java
```

Knownsec

```
|   |   ├──crypto
|   |   |   |   Aes.java
|   |   |   |   CryptoException.java
|   |   |   |   Hash.java
|   |   |   |   Hex.java
|   |   |   |   Key.java
|   |   |   |   Native.java
|   |   |   |
|   |   |   ├──bip32
|   |   |   |   |   CoinType.java
|   |   |   |   |   HdKeyGenerator.java
|   |   |   |   |   HdKeyPair.java
|   |   |   |   |   Scheme.java
|   |   |   |   |
|   |   |   |   ├──key
|   |   |   |   |       HdKey.java
|   |   |   |   |       HdPrivateKey.java
|   |   |   |   |       HdPublicKey.java
|   |   |   |   |       KeyVersion.java
|   |   |   |   |
|   |   |   |   └──util
|   |   |   |           BitUtil.java
|   |   |   |           BytesUtil.java
|   |   |   |           HashUtil.java
|   |   |   |           Hmac.java
|   |   |   |           Secp256k1.java
|   |   |   ├──bip39
|   |   |   |       Dictionary.java
|   |   |   |       Language.java
|   |   |   |       MnemonicGenerator.java
|   |   |   ├──bip44
|   |   |   |       Bip44.java
|   |   |   |
```

Knownsec

```
|   |   |   └──cache
|   |   |       PublicKeyCache.java
|   |   ├──db
|   |   |   |   Database.java
|   |   |   |   DatabaseFactory.java
|   |   |   |   DatabaseName.java
|   |   |   |   LeveldbDatabase.java
|   |   |   |
|   |   |   └──exception
|   |   |       DatabaseException.java
|   |   ├──event
|   |   |       KernelBootingEvent.java
|   |   |       PubSub.java
|   |   |       PubSubEvent.java
|   |   |       PubSubFactory.java
|   |   |       PubSubSubscriber.java
|   |   ├──exception
|   |   |       LauncherException.java
|   |   ├──gui
|   |   |   |   Action.java
|   |   |   |   AddressBookEntry.java
|   |   |   |   BohrGui.java
|   |   |   |   ComboBoxItem.java
|   |   |   |   HorizontalSeparator.java
|   |   |   |   MainFrame.java
|   |   |   |   MenuBar.java
|   |   |   |   PlaceHolder.java
|   |   |   |   SplashScreen.java
|   |   |   |   StatusBar.java
|   |   |   |   StatusBarNew.java
|   |   |   |   SwingUtil.java
|   |   |   |   TextContextMenuItem.java
|   |   |   |   TextPrompt.java
```

Knownsec

```
|   |   |   |   TransactionSender.java
|   |   |   |   VerticalSeparator.java
|   |   |   |   WelcomeFrame.java
|   |   |   |
|   |   |   |---dialog
|   |   |   |       AddressBookDialog.java
|   |   |   |       AddressBookUpdateDialog.java
|   |   |   |       ChangePasswordDialog.java
|   |   |   |       ConsoleDialog.java
|   |   |   |       DelegateDialog.java
|   |   |   |       ExportPrivateKeyDialog.java
|   |   |   |       InitializeHdWalletDialog.java
|   |   |   |       InputDialog.java
|   |   |   |       SelectDialog.java
|   |   |   |       TransactionDialog.java
|   |   |   |       TransactionResultDialog.java
|   |   |   |---event
|   |   |   |       MainFrameStartedEvent.java
|   |   |   |       WalletSelectionDialogShownEvent.java
|   |   |   |---laf
|   |   |   |       AutoTextField.java
|   |   |   |       BlueProgressBarUI.java
|   |   |   |       DefaultButtonUI.java
|   |   |   |       DefaultComboBoxUI.java
|   |   |   |       DefaultLookAndFeel.java
|   |   |   |       DefaultOptionPaneUI.java
|   |   |   |       DefaultPasswordFieldUI.java
|   |   |   |       DefaultRadioButtonUI.java
|   |   |   |       DefaultTextAreaUI.java
|   |   |   |       DefaultTextFieldUI.java
|   |   |   |       EmptyBlueArcButtonUI.java
|   |   |   |       EmptyBlueRectButtonUI.java
|   |   |   |       HalfButtonUI.java
```

knownsec

```
|       |       |   |       HalfComBoBoxUI.java
|       |       |   |       HalfTextFieldUI.java
|       |       |   |       MyDefaultMetalTheme.java
|       |       |   |       RoundRectButtonUI.java
|       |       |   |       RoundRectPanel.java
|       |       |   |       TableHeadRender.java
|       |       |   |       TableRender.java
|       |       |   |       WhiteSimpleComBoBoxUI.java
|       |       |   ├──layout
|       |       |   |       AbsLayoutManager.java
|       |       |   |       TableLayout.java
|       |       |   |
|       |       |   ├──model
|       |       |   |       WalletAccount.java
|       |       |   |       WalletDelegate.java
|       |       |   |       WalletModel.java
|       |       |   ├──panel
|       |       |   |       ContractPanel.java
|       |       |   |       DelegatesPanel.java
|       |       |   |       HomePanel.java
|       |       |   |       ReceivePanel.java
|       |       |   |       SendPanel.java
|       |       |   |       TransactionsPanel.java
|       |       |   |       TransactionsPanelFilter.java
|       |       |   ├──render
|       |       |   |       MainFrameUserComboBoxRender.java
|       |       |   |
|       |       |   └──uiUtils
|       |       |           ColorUtils.java
|       |       |           FontUtils.java
|       |       |           GraphicsUtils.java
|       |       |           LAFUtils.java
|       |       |           ModelUtils.java
```

knownsec

```
|       |       ├──log
|       |       |       LoggerConfigurator.java
|       |       |
|       |       ├──message
|       |       |       CliMessages.java
|       |       |       GuiMessages.java
|       |       |       LanguageChoiceConfig.java
|       |       |       MessageFormatter.java
|       |       |       ResourceBundles.java
|       |       ├──net
|       |       |   |   BohrChannelInitializer.java
|       |       |   |   BohrFrameHandler.java
|       |       |   |   BohrMessageHandler.java
|       |       |   |   BohrP2pHandler.java
|       |       |   |   Capability.java
|       |       |   |   CapabilityTreeSet.java
|       |       |   |   Channel.java
|       |       |   |   ChannelManager.java
|       |       |   |   ConnectionLimitHandler.java
|       |       |   |   Frame.java
|       |       |   |   NodeManager.java
|       |       |   |   Peer.java
|       |       |   |   PeerClient.java
|       |       |   |   PeerServer.java
|       |       |   ├──filter
|       |       |   |   |   BohrIpFilter.java
|       |       |   |   |   FilterRule.java
|       |       |   |   |   SingleIpFilterRule.java
|       |       |   |   |
|       |       |   |   └──exception
|       |       |   |           IpFilterJsonParseException.java
|       |       |   └──msg
|       |       |       |   Message.java
```

Knownsec

```
|   |   |   |   MessageCode.java
|   |   |   |   MessageException.java
|   |   |   |   MessageFactory.java
|   |   |   |   MessageQueue.java
|   |   |   |   ReasonCode.java
|   |   |   |---consensus
|   |   |   |       BlockHeaderMessage.java
|   |   |   |       BlockMessage.java
|   |   |   |       BlockPartsMessage.java
|   |   |   |       GetBlockHeaderMessage.java
|   |   |   |       GetBlockMessage.java
|   |   |   |       GetBlockPartsMessage.java
|   |   |   |       NewHeightMessage.java
|   |   |   |       NewViewMessage.java
|   |   |   |       ProposalMessage.java
|   |   |   |       VoteMessage.java
|   |   |   |---p2p
|   |   |   |   |   DisconnectMessage.java
|   |   |   |   |   GetNodesMessage.java
|   |   |   |   |   NodesMessage.java
|   |   |   |   |   PingMessage.java
|   |   |   |   |   PongMessage.java
|   |   |   |   |   TransactionMessage.java
|   |   |   |   |
|   |   |   |   |---handshake
|   |   |   |   |   |---v1
|   |   |   |   |   |       HandshakeMessage.java
|   |   |   |   |   |       HelloMessage.java
|   |   |   |   |   |       WorldMessage.java
|   |   |   |   |   |
|   |   |   |   |   |---v2
|   |   |   |   |           HandshakeMessage.java
|   |   |   |   |           HelloMessage.java
```

knownsec

```
|   |   |   |   InitMessage.java
|   |   |   |   WorldMessage.java
|   |   |   ├──tools
|   |   |   |       DatabaseIntegrityChecker.java
|   |   |   |
|   |   |   ├──util
|   |   |   |   |   ArrayUtil.java
|   |   |   |   |   Base58.java
|   |   |   |   |   BasicAuth.java
|   |   |   |   |   BigIntegerUtil.java
|   |   |   |   |   ByteArray.java
|   |   |   |   |   Bytes.java
|   |   |   |   |   CircularFixedSizeList.java
|   |   |   |   |   ClosableIterator.java
|   |   |   |   |   CommandParser.java
|   |   |   |   |   ConsoleUtil.java
|   |   |   |   |   FileUtil.java
|   |   |   |   |   IOUtil.java
|   |   |   |   |   MerkleTree.java
|   |   |   |   |   MerkleUtil.java
|   |   |   |   |   NullPrintStream.java
|   |   |   |   |   Sha256Sm3Hash.java
|   |   |   |   |   SimpleApiClient.java
|   |   |   |   |   SimpleDecoder.java
|   |   |   |   |   SimpleEncoder.java
|   |   |   |   |   StringUtil.java
|   |   |   |   |   SystemUtil.java
|   |   |   |   |   TimeUtil.java
|   |   |   |   |
|   |   |   |   └──exception
|   |   |   |           BytesException.java
|   |   |   |           SimpleCodecException.java
|   |   |   |           UnreachableException.java
```

knownsec

```
|           |           |
|           |           └──vm
|           |               └──client
|           |                       BohrBlock.java
|           |                       BohrBlockStore.java
|           |                       BohrInternalTransaction.java
|           |                       BohrPrecompiledContracts.java
|           |                       BohrRepository.java
|           |                       BohrSpec.java
|           |                       BohrTransaction.java
|           |                       Conversion.java
└──bohr-evm
    |   pom.xml
    |
    └──src
        └──main
            └──java
                └──org
                    └──ethereum
                        └──vm
                        |   DataWord.java
                        |   FeeSchedule.java
                        |   LogInfo.java
                        |   MessageCall.java
                        |   OpCode.java
                        |   VM.java
                        |
                        ├──chainspec
                        |       BasePrecompiledContracts.java
                        |       BaseSpec.java
                        |       ByzantiumPrecompiledContracts.java
                        |       ByzantiumSpec.java
                        |       ConstantinoplePrecompiledContracts.java
```

Knownsec

```
│       ConstantinopleSpec.java
│       PrecompiledContract.java
│       PrecompiledContractContext.java
│       PrecompiledContracts.java
│       Spec.java
│
├──client
│       Block.java
│       BlockStore.java
│       Repository.java
│       Transaction.java
│       TransactionExecutor.java
│       TransactionReceipt.java
│       Unit.java
│
├──crypto
│   │   ECKey.java
│   │
│   └──zksnark
│           BN128.java
│           BN128Fp.java
│           BN128Fp2.java
│           BN128G1.java
│           BN128G2.java
│           Field.java
│           Fp.java
│           Fp12.java
│           Fp2.java
│           Fp6.java
│           PairingCheck.java
│           Params.java
│
├──program
```

Knownsec

```
|   |   InternalTransaction.java
|   |   Memory.java
|   |   Program.java
|   |   ProgramPreprocess.java
|   |   ProgramResult.java
|   |   Stack.java
|   |
|   ├──exception
|   |       BadJumpDestinationException.java
|   |       BytecodeExecutionException.java
|   |       CallDepthOverflowException.java
|   |       CallTooDeepException.java
|   |       ExceptionFactory.java
|   |       IllegalOperationException.java
|   |       InsufficientBalanceException.java
|   |       OutOfGasException.java
|   |       PrecompiledFailureException.java
|   |       ReturnDataCopyIllegalBoundsException.java
|   |       StackOverflowException.java
|   |       StackUnderflowException.java
|   |       StaticCallModificationException.java
|   |
|   └──invoke
|           ProgramInvoke.java
|           ProgramInvokeFactory.java
|           ProgramInvokeFactoryImpl.java
|           ProgramInvokeImpl.java
|
└──util
        BigIntegerUtil.java
        ByteArrayUtil.java
        ByteArrayWrapper.java
        BytecodeCompiler.java
```

Knownsec

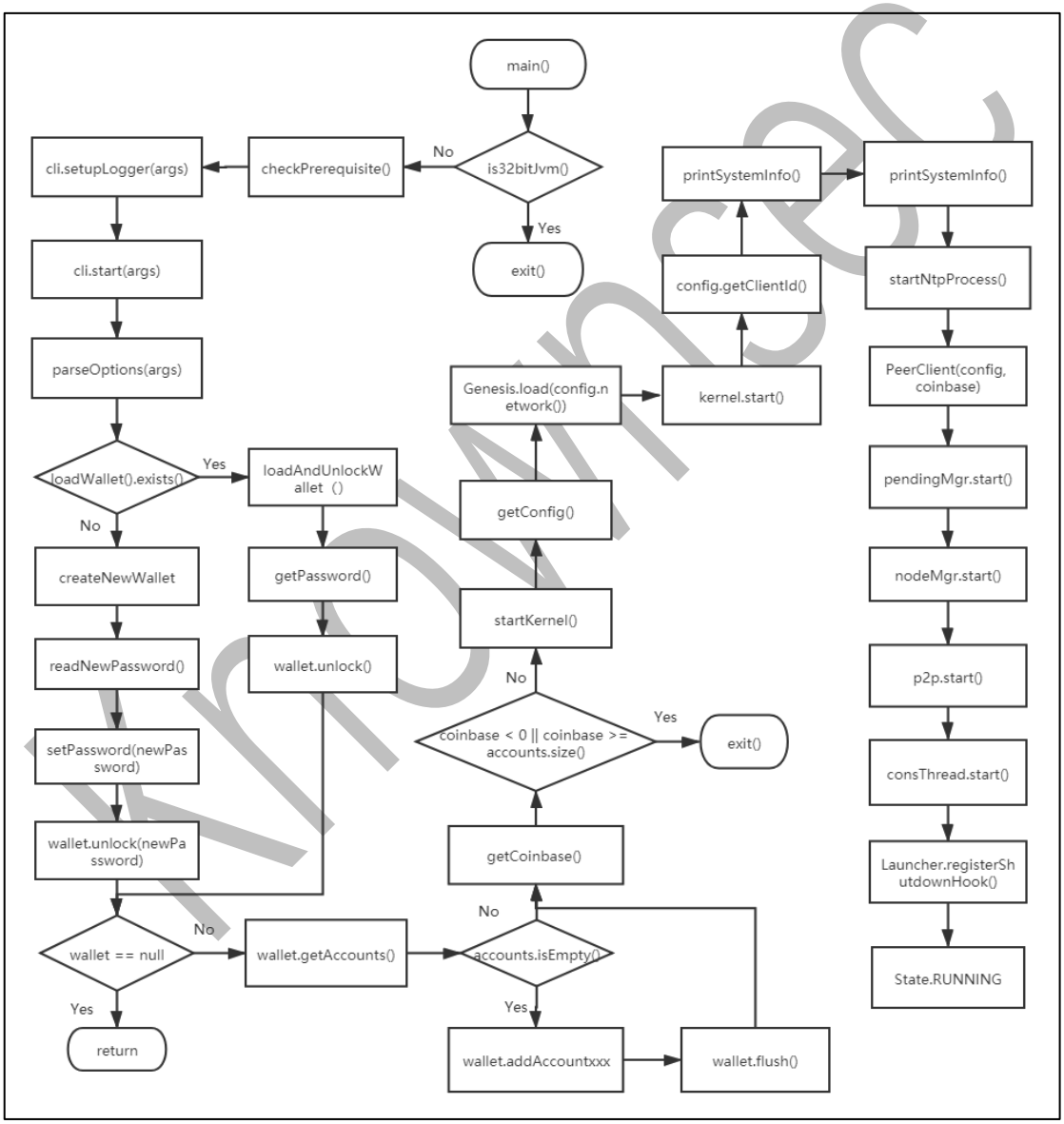
HashUtil.java

HexUtil.java

Pair.java

VMUtil.java

3.4. **Startup process**



3.5. **P2P related methods**

public Peer getPeer()

private static boolean validatePeer(Peer peer)

private static byte[] encodePeer(Peer peer)

private static Peer decodePeer(byte[] bytes)

public DisconnectMessage(ReasonCode reason)

public GetNodesMessage()

public NodesMessage(List<Node> nodes)

public PingMessage()

public PongMessage()

public String getIp()

public int getPort()

public Network getNetwork()

public Node getNode()

public String getClientId()

public String getPeerId()

public String[] getCapabilities()

public void setLatestBlockNumber(long number)

public long getLatency()

public void setLatency(long latency)

……

3.6. **API core interface list**

public Response addNode(String node)

public Response addToBlacklist(String ip)

public Response addToWhitelist(String ip)

public Response composeRawTransaction(String network, String type, String to, String value, String fee,String nonce, String timestamp, String data, String gas, String gasPrice)

public Response createAccount(String name, String privateKey)

public Response getAccount(String address)

public Response getAccountCode(String address)

Knownsec

public Response getAccountStorage(String address, String key)

public Response deleteAccount(String address)

public Response getAccountTransactions(String address, String from, String to)

public Response getAccountInternalTransactions(String address, String from, String to)

public Response getAccountPendingTransactions(String address, String from, String to)

public Response getAccountVotes(String address)

public Response getBlockByHash(String hashString)

public Response getBlockByNumber(String blockNum)

public Response getDelegate(String address)

public Response getDelegates()

public Response getLatestBlock()

public Response getLatestBlockNumber()

public Response getPeers()

public Response getPendingTransactions()

public Response getTransaction(String hash)

public Response getTransactionLimits(String type)

public Response getTransactionResult(String hash)

public Response getValidators()

public Response getVote(String delegate, String voter)

public Response getVotes(String delegate)

public Response getAccounts()

public Response broadcastRawTransaction(String raw)

public Response broadcastRawTransactionDeprecated(String raw)

public Response signMessage(String address, String message)

public Response signRawTransaction(String raw, String address)

public Response transfer(String from, String to, String value, String fee, String nonce, String data)

public Response verifyMessage(String address, String message, String signature)

private Response success(ApiHandlerResponse resp)

private Response badRequest(String message)

private NodeManager.Node validateAddNodeParameter(String node)

private Transaction getTransaction(TransactionType type, String from, String to, String value, String fee,String nonce, String data, String gas, String gasPrice)

private Response doTransactions(TransactionType type, String froms, String tos, String values, String

Knownsec

nonces)

private String parseIp(String ip, boolean required)

private byte[] parseAddress(String address, boolean required, String name)

private byte[] parseHash(String hash, boolean required)

private Integer parseInt(String num, boolean required, String name)

private byte[] parseHex(String data, boolean required, String name)

.....

3.7. Third-party library reference list

swagger-annotations 1.5.22

swagger-ui 3.22.2

javax.ws.rs-api 2.1.1

validation-api 2.0.1.Final

jaxb-api 2.3.0

Activation 1.1.1

jersey-common 2.29

slf4j-api 1.7.26

bcprov-jdk15on 1.62

leveldbjni-all 1.18.3

netty-all 4.1.37.Final

snappy-java 1.1.7.3

oshi-core 3.13.3

commons-lang3 3.9

commons-collections4 4.3

commons-validator 1.6

commons-cli 1.4

com.fasterxml.jackson.core 2.9.9

jackson-databind 2.9.9.2

net.i2p.crypto 0.3.0

com.github.zafarkhaja 0.9.0

org.glassfish.jersey.ext 2.29

org.glassfish.jersey.media 2.29

knownsec

com.madgag.spongycastle 1.51.0.0

net.revelc.code.formatter 2.6.0

……

Knownsec

4. Audit Results

4.1. Cryptography

4.1.1. Random number range and probability distribution【PASS】

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

Risk hazard: Leakage of sensitive information, arbitrary issuance of transactions, affecting asset security, etc.

Risk file:

Audit process: Check whether the generation and use of random numbers in the public chain meet the requirements.

```
private int getUniformDistPrimaryValidatorNumber(int size, long height, long view) {
        // use round-robin for view 0
        if (view == 0) {
            return (int) (height % (long) size);
        }

        // here we ensure there will never be consecutive block forgers after view
        // change
        int deterministicRand;
        final int prevDeterministicRand = getUniformDistPrimaryValidatorNumber(size, height,
view - 1);
        BigInteger subView = BigInteger.ZERO;
        do {
            BigInteger seed = BigIntegerUtil
                    .random(BigInteger.valueOf(height))
```

Knownsec

```
                    .xor(BigIntegerUtil.random(BigInteger.valueOf(view)))
                    .add(subView);//knownsec// Initialize the random number seed
            deterministicRand = BigIntegerUtil
                    .random(seed)
                    .mod(BigInteger.valueOf(size))
                    .intValue();
            subView = subView.add(BigInteger.ONE);
        } while (deterministicRand == prevDeterministicRand && size > 1);

        return deterministicRand;
}
```

Audit results: After auditing, applications involving random numbers in the public chain code have initialized random seeds, which are in line with secure coding operations.

Safety advice: nothing.

4.1.2. Implementation/use of cryptographic algorithms【PASS】

Audit description: Check whether the use or implementation of cryptography in signature, hash, and verification links is reasonable.

Risk hazard: Forged blocks/transactions, severe cases can lead to chain forks.

Risk file:

Audit process: Check whether the encryption algorithm used in the signature, hash, and verification links is safe.

Audit results: After testing, the core data encryption operations in the public chain code use sha256 and sha512 libraries, which are in line with secure coding operations.

knownsec

Safety advice: nothing.

4.2. RPC

4.2.1. Sensitive interface permissions【PASS】

Audit description: Check the access authority of the RPC sensitive interface, whether it is exposed, test whether the RPC interface can access sensitive operations or data.

Risk hazard: Leakage of sensitive information, arbitrary issuance of transactions, affecting asset security, etc.

Risk file:

Audit process: Check the access permissions of API sensitive interfaces.

```
#================
# API
#================
# Whether to enable API services. Make sure you change the password below.
api.enabled = false

# Binding IP address and port
api.listenIp = 127.0.0.1
api.listenPort = 5391

# Basic access authentication credential
api.username = BOHR_API_USERNAME
api.password = BOHR_API_PASSWORD

# Enable services below as public (authentication optional)
api.public = blockchain,account,delegate,tool
```

Knownsec

```
# Enable services below as private (authentication required)
api.private = node,wallet
```

Audit results: After testing, the public chain API interface is closed by default, and only local access is allowed by default in the open state, and identity authentication is required.

Safety advice: nothing.

4.2.2. Business logic【PASS】

Audit description: Check the access rights of the API interface, export wallets, backup wallets, and other functions for unauthorized use, logic errors, etc.

Risk hazard: leakage of sensitive information, arbitrary issuance of transactions, affecting asset security, etc.

Risk file:

Audit process: Check whether there are overreach and logic errors in the business logic design of the public chain contract operation.

```
protected void importPrivateKey(String key) {
        try {
            Wallet wallet = loadAndUnlockWallet();
            byte[] keyBytes = Hex.decode0x(key);
            Key account = new Key(keyBytes);

            boolean accountAdded = wallet.addAccount(account);
            if (!accountAdded) {
                logger.error(CliMessages.get("PrivateKeyAlreadyInWallet"));
                exit(SystemUtil.Code.ACCOUNT_ALREADY_EXISTS);
```

Knownsec

```
                return;
            }

            boolean walletFlushed = wallet.flush();
            if (!walletFlushed) {
                logger.error(CliMessages.get("WalletFileCannotBeUpdated"));
                exit(SystemUtil.Code.FAILED_TO_WRITE_WALLET_FILE);
                return;
            }

            logger.info(CliMessages.get("PrivateKeyImportedSuccessfully"));
            logger.info(CliMessages.get("Address", account.toAddressString()));
            logger.info(CliMessages.get("PublicKey", Hex.encode(account.getPublicKey())));
        } catch (InvalidKeySpecException exception) {
            logger.error(CliMessages.get("PrivateKeyCannotBeDecoded",
exception.getMessage()));
            exit(SystemUtil.Code.INVALID_PRIVATE_KEY);
        } catch (WalletLockedException exception) {
            logger.error(exception.getMessage());
            exit(SystemUtil.Code.WALLET_LOCKED);
        }
    }
    protected Wallet loadAndUnlockWallet() {

        Wallet wallet = loadWallet();
        if (getPassword() == null) {
            if (wallet.unlock("")) {
                setPassword("");
            } else {
                setPassword(readPassword());
            }
        }
```

Knownsec

```
        if (!wallet.unlock(getPassword())) {//knownsec// Password validation
            logger.error("Invalid password");
            exit(SystemUtil.Code.FAILED_TO_UNLOCK_WALLET);
        }
        return wallet;
    }
    protected void dumpPrivateKey(String address) {
        Wallet wallet = loadAndUnlockWallet();

        byte[] addressBytes = Hex.decode0x(address);
        Key account = wallet.getAccount(addressBytes);
        if (account == null) {
            logger.error(CliMessages.get("AddressNotInWallet"));
            exit(SystemUtil.Code.ACCOUNT_NOT_EXIST);
        } else {
            System.out.println(CliMessages.get("PrivateKeyIs",
Hex.encode(account.getPrivateKey())));
        }
    }
    protected Wallet loadAndUnlockWallet() {
        Wallet wallet = loadWallet();
        if (getPassword() == null) {
            if (wallet.unlock("")) {
                setPassword("");
            } else {
                setPassword(readPassword());
            }
        }

        if (!wallet.unlock(getPassword())) {//knownsec// Password authentication
            logger.error("Invalid password");
            exit(SystemUtil.Code.FAILED_TO_UNLOCK_WALLET);
        }
```

knownsec

```
        return wallet;
    }
```

Audit results: After testing, the core operation processing logic involved in the public chain code is correct and conforms to the safe coding operation.

Safety advice: nothing.

4.2.3. Traditional Web Security【PASS】

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

Risk hazard: remote command execution, exposure of RPC interfaces, caused by CSRF and SSRF attacks, arbitrary file reading, leakage of sensitive information, etc.

Risk file:

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

Audit results: After auditing, no relevant risks were found.

Safety advice: nothing.

Knownsec

4.3. Key security

4.3.1. Private key/mnemonic word generation algorithm【PASS】

Audit description: Check whether the logic of the private key generation algorithm is reasonable, and whether the test complexity bottleneck meets expectations.

Risk hazard: leakage of sensitive information, arbitrary issuance of transactions, affecting asset security, etc.

Risk file:

Audit process: After testing, the public chain uses the Ed25519/EdDSA signature algorithm to generate the private key, and no related security risks are found..

Audit results: After auditing, no relevant risks were found.

Safety advice: nothing.

4.3.2. Private key/mnemonic phrase storage in plain text【PASS】

Audit description: check the storage method, whether the file is encrypted or can be decrypted.

Risk hazard: Cooperating with other vulnerabilities can lead to the disclosure of private keys and affect asset security. In severe cases, it can lead to problems such as chain forks.

Risk file:

Knownsec

Audit process: Check whether the private key/mnemonic phrase is stored in plain text.

Audit results: After testing, the private key/mnemonic phrase in the public chain code is encrypted and stored.d.

Safety advice: nothing.

4.3.3. Use traces of private keys/mnemonics【PASS】

Audit description: Check whether the private key/mnemonic phrase in the code logic is used to clean up traces in time, and check whether sensitive data is used, and whether traces of information can be found from the log and memory dump.

Risk hazard: Cooperating with other vulnerabilities can lead to the disclosure of private keys and affect asset security. In severe cases, it can cause chain forks and other problems.

Risk file:

Audit process: Check whether the private key/mnemonic phrase in the code logic is used to clean up the traces in time.

Audit results: After testing, the use of private keys/mnemonic words in the public chain code complies with safe operation specifications, and no related risks have been found.

Safety advice: nothing.

knownsec

4.3.4. Residual private key/mnemonic phrase【PASS】

Audit description: Check whether the deletion of the private key/mnemonic phrase is complete.

Risk hazard: Cooperating with other vulnerabilities can lead to the disclosure of private keys and affect asset security. In severe cases, it can lead to problems such as chain forks.

Risk file:

Audit process: Check whether the deletion of the private key/mnemonic phrase is complete.

Audit results: After testing, the use of private keys/mnemonic words in the public chain code complies with safe operation specifications, and no related risks have been found.

Safety advice: nothing.

4.3.5. Misuse of private key/mnemonic phrase【PASS】

Audit description: Check whether the private key/mnemonic phrase is used in an unnecessary place.

Risk hazard: increase the risk of private key leakage.

Risk file:

Audit process: Check whether the private key/mnemonic phrase is used in an unnecessary place..

Knownsec

Audit results: After testing, the use of private keys/mnemonic words in the public chain code complies with safe operation specifications, and no related risks have been found.

Safety advice: nothing.

4.4. Consensus mechanism

4.4.1. Implementation of consensus verification【PASS】

Audit description: Check whether consensus such as pow/poc can be used to construct legal blocks with less than expected cost.

Risk hazard: forged computing power, centralization of computing power, severe cases can lead to 51% attacks, chain bifurcation and other problems.

Risk file:

Audit process: Audit the design and implementation of the consensus mechanism.

```
/**
 * Update the validator sets.
 */
protected void updateValidators() {
        int maxValidators = config.spec().getNumberOfValidators(height);

        validators = chain.getValidators();
        // if the chain is reporting a larger number of validators
        // then a configuration change has occurred (like a stuck testnet)
        // so honor the configuration value
        if (validators.size() > maxValidators) {
                validators = validators.subList(0, maxValidators);
```

Knownsec

```
        }
        activeValidators = channelMgr.getActiveChannels(validators);
        lastUpdate = TimeUtil.currentTimeMillis();
    }

    /**
     * Check if this node is a validator.
     *
     * @return
     */
    protected boolean isValidator() {
        return validators.contains(coinbase.toAddressString());
    }

     /**
     * Create a block for BFT proposal.
     *
     * @return the proposed block
     */
    protected Block proposeBlock() {
        AccountState asTrack = chain.getAccountState().track();
        DelegateState dsTrack = chain.getDelegateState().track();

        long t1 = TimeUtil.currentTimeMillis();

        // construct block template
        BlockHeader parent = chain.getBlockHeader(height - 1);
        long number = height;
        byte[] prevHash = parent.getHash();
        long timestamp = TimeUtil.currentTimeMillis();
        timestamp = timestamp > parent.getTimestamp() ? timestamp : parent.getTimestamp() + 1;
        byte[] data = chain.constructBlockHeaderDataField();
```

Knownsec

```
        BlockHeader tempHeader = new BlockHeader(height, coinbase.toAddress(), prevHash,
timestamp, new byte[0],
                new byte[0], new byte[0], data);

        // fetch pending transactions
        final List<PendingManager.PendingTransaction> pendingTxs = pendingMgr
                .getPendingTransactions(config.poolBlockGasLimit());
        final List<Transaction> includedTxs = new ArrayList<>();
        final List<TransactionResult> includedResults = new ArrayList<>();

        TransactionExecutor exec = new TransactionExecutor(config, blockStore,
chain.isVMEnabled(),
                chain.isVotingPrecompiledUpgraded());
        BohrBlock bohrBlock = new BohrBlock(tempHeader, config.spec().maxBlockGasLimit());

        // only propose gas used up to configured block gas limit
        long remainingBlockGas = config.poolBlockGasLimit();
        long gasUsedInBlock = 0;
        for (PendingManager.PendingTransaction pendingTx : pendingTxs) {
            Transaction tx = pendingTx.transaction;

            // check if the remaining gas covers the declared gas limit
            long gas = tx.isVMTransaction() ? tx.getGas() :
config.spec().nonVMTransactionGasCost();
            if (gas > remainingBlockGas) {
                break;
            }

            // re-evaluate the transaction
            TransactionResult result = exec.execute(tx, asTrack, dsTrack, bohrBlock,
gasUsedInBlock);
            if (result.getCode().isAcceptable()) {
                includedTxs.add(tx);
```

Knownsec

```
                includedResults.add(result);

                // update counter
                long gasUsed = tx.isVMTransaction() ? result.getGasUsed() :
config.spec().nonVMTransactionGasCost();
                remainingBlockGas -= gasUsed;
                gasUsedInBlock += gasUsed;
            }
        }

        // compute roots
        byte[] transactionsRoot = MerkleUtil.computeTransactionsRoot(includedTxs);
        byte[] resultsRoot = MerkleUtil.computeResultsRoot(includedResults);
        byte[] stateRoot = Bytes.EMPTY_HASH;

        BlockHeader header = new BlockHeader(number, coinbase.toAddress(), prevHash,
timestamp, transactionsRoot,
                resultsRoot, stateRoot, data);
        Block block = new Block(header, includedTxs, includedResults);

        long t2 = TimeUtil.currentTimeMillis();
        logger.debug("Block creation: # txs = {}, time = {} ms", includedTxs.size(), t2 - t1);

        return block;
    }
    /**
     * Check if a block proposal is valid.
     */
    protected boolean validateBlockProposal(BlockHeader header, List<Transaction> transactions) {
        try {
            AccountState asTrack = chain.getAccountState().track();
            DelegateState dsTrack = chain.getDelegateState().track();
```

knownsec

```
                Block block = new Block(header, transactions);
                long t1 = TimeUtil.currentTimeMillis();

                // [1] check block header
                Block latest = chain.getLatestBlock();
                if (!block.validateHeader(header, latest.getHeader())) {
                    logger.warn("Invalid block header");
                    return false;
                }

                // [?] additional checks by consensus
                // - disallow block time drifting;
                // - restrict the coinbase to be the proposer
                if (header.getTimestamp() - TimeUtil.currentTimeMillis() >
config.bftMaxBlockTimeDrift()) {
                    logger.warn("A block in the future is not allowed");
                    return false;
                }

                if (!Arrays.equals(header.getCoinbase(), proposal.getSignature().getAddress())) {
                    logger.warn("The coinbase should always equal to the proposer's address");
                    return false;
                }

                // [2] check transactions
                List<Transaction> unvalidatedTransactions = getUnvalidatedTransactions(transactions);
                if (!block.validateTransactions(header, unvalidatedTransactions, transactions,
config.network())) {
                    logger.warn("Invalid transactions");
                    return false;
                }
                if (transactions.stream().anyMatch(tx -> chain.hasTransaction(tx.getHash()))) {
                    logger.warn("Duplicated transaction hash is not allowed");
```

Knownsec

```
                return false;
            }

            // [3] evaluate transactions
            TransactionExecutor transactionExecutor = new TransactionExecutor(config,
blockStore, chain.isVMEnabled(),
                    chain.isVotingPrecompiledUpgraded());
            List<TransactionResult> results = transactionExecutor.execute(transactions, asTrack,
dsTrack,
                    new BohrBlock(header, config.spec().maxBlockGasLimit()), 0);
            if (!block.validateResults(header, results)) {
                logger.error("Invalid transaction results");
                return false;
            }
            block.setResults(results); // overwrite the results

            long t2 = TimeUtil.currentTimeMillis();
            logger.debug("Block validation: # txs = {}, time = {} ms", transactions.size(), t2 - t1);

            validBlocks.put(ByteArray.of(block.getHash()), block);
            return true;
        } catch (Exception e) {
            logger.error("Unexpected exception during block proposal validation", e);
            return false;
        }
    }
```

Audit results: After auditing, BOHR public chain adopts COSH consensus algorithm, using this algorithm to periodically select a number of super nodes, give them notarization rights of data units, and will issue effective notarization units to obtain block rewards based on them.

Knownsec

Safety advice: nothing.

4.4.2. Consensus mechanism design【PASS】

Audit description: In view of the new type of consensus mechanism, check whether there are any security risks in the design based on its documents, such as missing transactions/transactions not verified/forged proof of computing power/centralization/BP doing evil.

Risk hazard: analysis based on specific conditions.

Risk file:

Audit process: Audit the consensus mechanism design of the BOHR public chain.

```
/**
     * Update the validator sets.
     */
    protected void updateValidators() {
        int maxValidators = config.spec().getNumberOfValidators(height);

        validators = chain.getValidators();
        // if the chain is reporting a larger number of validators
        // then a configuration change has occurred (like a stuck testnet)
        // so honor the configuration value
        if (validators.size() > maxValidators) {
            validators = validators.subList(0, maxValidators);
        }
        activeValidators = channelMgr.getActiveChannels(validators);
        lastUpdate = TimeUtil.currentTimeMillis();
    }
```

knownsec

```
    /**
     * Check if this node is a validator.
     *
     * @return
     */
    protected boolean isValidator() {
        return validators.contains(coinbase.toAddressString());
    }

    /**
     * Create a block for BFT proposal.
     *
     * @return the proposed block
     */
    protected Block proposeBlock() {
        AccountState asTrack = chain.getAccountState().track();
        DelegateState dsTrack = chain.getDelegateState().track();

        long t1 = TimeUtil.currentTimeMillis();

        // construct block template
        BlockHeader parent = chain.getBlockHeader(height - 1);
        long number = height;
        byte[] prevHash = parent.getHash();
        long timestamp = TimeUtil.currentTimeMillis();
        timestamp = timestamp > parent.getTimestamp() ? timestamp : parent.getTimestamp() + 1;
        byte[] data = chain.constructBlockHeaderDataField();
        BlockHeader tempHeader = new BlockHeader(height, coinbase.toAddress(), prevHash,
timestamp, new byte[0],
                new byte[0], new byte[0], data);

        // fetch pending transactions
```

Knownsec

```
            final List<PendingManager.PendingTransaction> pendingTxs = pendingMgr
                    .getPendingTransactions(config.poolBlockGasLimit());
            final List<Transaction> includedTxs = new ArrayList<>();
            final List<TransactionResult> includedResults = new ArrayList<>();

            TransactionExecutor exec = new TransactionExecutor(config, blockStore,
chain.isVMEnabled(),
                    chain.isVotingPrecompiledUpgraded());
            BohrBlock bohrBlock = new BohrBlock(tempHeader, config.spec().maxBlockGasLimit());

            // only propose gas used up to configured block gas limit
            long remainingBlockGas = config.poolBlockGasLimit();
            long gasUsedInBlock = 0;
            for (PendingManager.PendingTransaction pendingTx : pendingTxs) {
                Transaction tx = pendingTx.transaction;

                // check if the remaining gas covers the declared gas limit
                long gas = tx.isVMTransaction() ? tx.getGas() :
config.spec().nonVMTransactionGasCost();
                if (gas > remainingBlockGas) {
                    break;
                }

                // re-evaluate the transaction
                TransactionResult result = exec.execute(tx, asTrack, dsTrack, bohrBlock,
gasUsedInBlock);
                if (result.getCode().isAcceptable()) {
                    includedTxs.add(tx);
                    includedResults.add(result);

                    // update counter
                    long gasUsed = tx.isVMTransaction() ? result.getGasUsed() :
config.spec().nonVMTransactionGasCost();
```

knownsec

```
                remainingBlockGas -= gasUsed;
                gasUsedInBlock += gasUsed;
            }
        }

        // compute roots
        byte[] transactionsRoot = MerkleUtil.computeTransactionsRoot(includedTxs);
        byte[] resultsRoot = MerkleUtil.computeResultsRoot(includedResults);
        byte[] stateRoot = Bytes.EMPTY_HASH;

        BlockHeader header = new BlockHeader(number, coinbase.toAddress(), prevHash,
timestamp, transactionsRoot,
                resultsRoot, stateRoot, data);
        Block block = new Block(header, includedTxs, includedResults);

        long t2 = TimeUtil.currentTimeMillis();
        logger.debug("Block creation: # txs = {}, time = {} ms", includedTxs.size(), t2 - t1);

        return block;
    }
    /**
     * Check if a block proposal is valid.
     */
    protected boolean validateBlockProposal(BlockHeader header, List<Transaction> transactions) {
        try {
            AccountState asTrack = chain.getAccountState().track();
            DelegateState dsTrack = chain.getDelegateState().track();

            Block block = new Block(header, transactions);
            long t1 = TimeUtil.currentTimeMillis();

            // [1] check block header
            Block latest = chain.getLatestBlock();
```

Knownsec

```
            if (!block.validateHeader(header, latest.getHeader())) {
                logger.warn("Invalid block header");
                return false;
            }

            // [?] additional checks by consensus
            // - disallow block time drifting;
            // - restrict the coinbase to be the proposer
            if (header.getTimestamp() - TimeUtil.currentTimeMillis() >
config.bftMaxBlockTimeDrift()) {
                logger.warn("A block in the future is not allowed");
                return false;
            }

            if (!Arrays.equals(header.getCoinbase(), proposal.getSignature().getAddress())) {
                logger.warn("The coinbase should always equal to the proposer's address");
                return false;
            }

            // [2] check transactions
            List<Transaction> unvalidatedTransactions = getUnvalidatedTransactions(transactions);
            if (!block.validateTransactions(header, unvalidatedTransactions, transactions,
config.network())) {
                logger.warn("Invalid transactions");
                return false;
            }
            if (transactions.stream().anyMatch(tx -> chain.hasTransaction(tx.getHash()))) {
                logger.warn("Duplicated transaction hash is not allowed");
                return false;
            }

            // [3] evaluate transactions
```

Knownsec

```
                TransactionExecutor transactionExecutor = new TransactionExecutor(config,
blockStore, chain.isVMEnabled(),
                        chain.isVotingPrecompiledUpgraded());
                List<TransactionResult> results = transactionExecutor.execute(transactions, asTrack,
dsTrack,
                        new BohrBlock(header, config.spec().maxBlockGasLimit()), 0);
                if (!block.validateResults(header, results)) {
                    logger.error("Invalid transaction results");
                    return false;
                }
                block.setResults(results); // overwrite the results

                long t2 = TimeUtil.currentTimeMillis();
                logger.debug("Block validation: # txs = {}, time = {} ms", transactions.size(), t2 - t1);

                validBlocks.put(ByteArray.of(block.getHash()), block);
                return true;
            } catch (Exception e) {
                logger.error("Unexpected exception during block proposal validation", e);
                return false;
            }
        }
```

Audit results: After auditing, after auditing, BOHR public chain adopts COSH consensus algorithm, using this algorithm to periodically select a number of super nodes, give them notarization rights for data units, and obtain block rewards based on their effective notarization units.

Safety advice: nothing.

knownsec

4.4.3. Voting verification logic【PASS】

Audit description: Audit the voting logic to check whether there are hidden safety hazards or uncontrollable risk factors in the design.

Risk hazard: analysis based on specific conditions.

Risk file:

Audit process: Audit the voting logic and check whether there is a security risk in the logic design.

```
/**
 * validate the vote format and signature while ignoring any cached validation
 * value.
 * @return
 */
public boolean revalidate() {
    return (validated = (type != null
            && height > 0
            && view >= 0
            && blockHash != null && blockHash.length == 32
            && encoded != null
            && signature != null && Key.verify(encoded, signature)));
}
/**
 * Verifies a signature.
 * @param message
 *        message
 * @param signature
 *        signature
 * @return True if the signature is valid, otherwise false
 */
public static boolean verify(byte[] message, Signature signature) {
```

Knownsec

```
        if (message != null && signature != null) { // avoid null pointer exception
            try {
                if (Native.isEnabled()) {
                    return Native.verify(message, signature.getS(), signature.getA());
                } else {
                    EdDSAEngine engine = new EdDSAEngine();
                   engine.initVerify(PublicKeyCache.computeIfAbsent(signature.getPublicKey()));
                    return engine.verifyOneShot(message, signature.getS());
                }
            } catch (Exception e) {
                // do nothing
            }
        }
        return false;
    }
```

Audit results: After testing, voting to verify that the logic design is reasonable, and no related security risks were found.

Safety advice: nothing.

4.4.4. Proposal verification logic【PASS】

Audit description: Audit the proposal verification logic to check whether there are hidden safety hazards or uncontrollable risk factors in the design..

Risk hazard: analysis based on specific conditions.

Risk file:

Audit process: Audit the proposal verification logic to check whether there is a security risk in the logic design.

```
    /**
```

Knownsec

```
     * Check if a block proposal is valid.
     */
    protected boolean validateBlockProposal(BlockHeader header, List<Transaction> transactions) {
        try {
            AccountState asTrack = chain.getAccountState().track();
            DelegateState dsTrack = chain.getDelegateState().track();

            Block block = new Block(header, transactions);
            long t1 = TimeUtil.currentTimeMillis();

            // [1] check block header
            Block latest = chain.getLatestBlock();
            if (!block.validateHeader(header, latest.getHeader())) {
                logger.warn("Invalid block header");
                return false;
            }

            // [?] additional checks by consensus
            // - disallow block time drifting;
            // - restrict the coinbase to be the proposer
            if (header.getTimestamp() - TimeUtil.currentTimeMillis() >
config.bftMaxBlockTimeDrift()) {
                logger.warn("A block in the future is not allowed");
                return false;
            }

            if (!Arrays.equals(header.getCoinbase(), proposal.getSignature().getAddress())) {
                logger.warn("The coinbase should always equal to the proposer's address");
                return false;
            }

            // [2] check transactions
            List<Transaction> unvalidatedTransactions = getUnvalidatedTransactions(transactions);
```

Knownsec

```
                if (!block.validateTransactions(header, unvalidatedTransactions, transactions,
config.network())) {
                    logger.warn("Invalid transactions");
                    return false;
                }
                if (transactions.stream().anyMatch(tx -> chain.hasTransaction(tx.getHash()))) {
                    logger.warn("Duplicated transaction hash is not allowed");
                    return false;
                }

                // [3] evaluate transactions
                TransactionExecutor transactionExecutor = new TransactionExecutor(config,
blockStore, chain.isVMEnabled(),
                        chain.isVotingPrecompiledUpgraded());
                List<TransactionResult> results = transactionExecutor.execute(transactions, asTrack,
dsTrack,
                        new BohrBlock(header, config.spec().maxBlockGasLimit()), 0);
                if (!block.validateResults(header, results)) {
                    logger.error("Invalid transaction results");
                    return false;
                }
                block.setResults(results); // overwrite the results
                long t2 = TimeUtil.currentTimeMillis();
                logger.debug("Block validation: # txs = {}, time = {} ms", transactions.size(), t2 - t1);
                validBlocks.put(ByteArray.of(block.getHash()), block);
                return true;
            } catch (Exception e) {
                logger.error("Unexpected exception during block proposal validation", e);
                return false;
            }
        }
```

knownsec

Audit results: After testing, the logic design of the proposal verification is reasonable, and no related security risks have been found.

Safety advice: nothing.

4.5. P2P

4.5.1. Differential response to sensitive information【PASS】

Audit description: Check the response data logic of various responses, and test whether there are different responses to sensitive information in various responses.

Risk hazard: leakage of sensitive information.

Risk file:

Audit process: Check whether there is any abnormality in the logic processing of the response data of various responses.

```
/**
     * Validates this message.
     * <p>
     * NOTE: only data format and signature is checked here.
     * </p>
     * @param config
     * @return true if success, otherwise false
     */
    public boolean validate(Config config) {
        if (peer != null && validatePeer(peer)
                && Math.abs(TimeUtil.currentTimeMillis() - timestamp) <=
config.netHandshakeExpiry()
                && signature != null
```

Knownsec

```
                && peer.getPeerId().equals(Hex.encode(signature.getAddress()))) {

            SimpleEncoder enc = new SimpleEncoder();
            enc.writeBytes(encodePeer(peer));
            enc.writeLong(timestamp);

            return Key.verify(enc.toBytes(), signature);
        } else {
            return false;
        }
    }
    /**
     * Verifies a signature.
     *
     * @param message
     *            message
     * @param signature
     *            signature
     * @return True if the signature is valid, otherwise false
     */
    public static boolean verify(byte[] message, Signature signature) {
        if (message != null && signature != null) { // avoid null pointer exception
            try {
                if (Native.isEnabled()) {
                    return Native.verify(message, signature.getS(), signature.getA());
                } else {
                    EdDSAEngine engine = new EdDSAEngine();

engine.initVerify(PublicKeyCache.computeIfAbsent(signature.getPublicKey()));

                    return engine.verifyOneShot(message, signature.getS());
                }
            } catch (Exception e) {
```

knownsec

```
                    // do nothing
                }
            }
            return false;
        }
```

Audit results: After testing, the public chain code involved in the P2P connection response function only responds to the standard msg protocol, which complies with the security coding specification.

Safety advice: nothing.

4.5.2. Fuzz malformed data test【PASS】

Audit description: Check the response data logic of various responses, and test whether the nodes respond normally in various responses.

Risk hazard: Malformed data attacks may lead to node denial of service or disclosure of chain-related information.

Risk file:

Audit process: Check whether there are any abnormalities in the logic design of the response data of various responses.

```
/**
     * Validates this message.
     * <p>
     * NOTE: only data format and signature is checked here.
     * </p>
     * @param config
     * @return true if success, otherwise false
     */
```

knownsec

```
    public boolean validate(Config config) {
        if (peer != null && validatePeer(peer)
                && Math.abs(TimeUtil.currentTimeMillis() - timestamp) <=
config.netHandshakeExpiry()
                && signature != null
                && peer.getPeerId().equals(Hex.encode(signature.getAddress()))) {

            SimpleEncoder enc = new SimpleEncoder();
            enc.writeBytes(encodePeer(peer));
            enc.writeLong(timestamp);

            return Key.verify(enc.toBytes(), signature);
        } else {
            return false;
        }
    }
    /**
     * Verifies a signature.
     *
     * @param message
     *            message
     * @param signature
     *            signature
     * @return True if the signature is valid, otherwise false
     */
    public static boolean verify(byte[] message, Signature signature) {
        if (message != null && signature != null) { // avoid null pointer exception
            try {
                if (Native.isEnabled()) {
                    return Native.verify(message, signature.getS(), signature.getA());
                } else {
                    EdDSAEngine engine = new EdDSAEngine();
```

Knownsec

```
engine.initVerify(PublicKeyCache.computeIfAbsent(signature.getPublicKey()));

                    return engine.verifyOneShot(message, signature.getS());
                }
            } catch (Exception e) {
                // do nothing
            }
        }
        return false;
    }
```

Audit results: After testing, the public chain code involved in the P2P connection response function only responds to the standard msg protocol, and there is a logic processing mechanism for other protocols, which conforms to the security coding specification.

Safety advice: nothing.

4.5.3. Node Discovery Algorithm【PASS】

Audit description: Check whether the node finds that the algorithm is balanced and unpredictable, such as the imbalance of the distance algorithm.

Risk hazard: analysis based on specific conditions.

Risk file:

Audit process: Audit the design of the node discovery algorithm to check whether the balance is unpredictable.

Audit results: After the audit, no relevant risks were found..

Safety advice: nothing.

Knownsec

4.5.4. Node communication protocol【PASS】

Audit description: Check whether there is a design/implementation layer problem in the communication protocol between nodes, perform fuzz testing on the nodes, and test the response of the nodes to malformed data packets.

Risk hazard: analysis based on specific conditions.

Risk file:

Audit process: Check whether there is error handling logic in the design/implementation layer of the communication protocol between nodes.

Audit results: After auditing, no relevant risks were found.

Safety advice: nothing.

4.5.5. Routing table pollution【PASS】

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

Risk hazard: reduce the difficulty of solar eclipse attacks..

Risk file:

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

Audit results: After auditing, no relevant risks were found.

Safety advice: nothing.

Knownsec

4.5.6. Traffic Limit【PASS】

Audit description: Check the restrictions on the number of connections and data packet size in the various p2p protocols in the code logic, try to establish a large number of connections with the target node, and construct and send data packets of various sizes.

Risk hazard: DoS nodes occupy local storage resources. In severe cases, the node program may crash.

Risk file:

Audit process: Check whether there are restrictions on the number of connections and data packet size in the various p2p protocols in the public chain code logic.

```
// ========================
// Network
// ========================
protected int netMaxOutboundConnections = 128;
protected int netMaxInboundConnections = 512;
protected int netMaxInboundConnectionsPerIp = 5;
protected int netMaxMessageQueueSize = 4096;
protected int netMaxFrameBodySize = 128 * 1024;
protected int netMaxPacketSize = 16 * 1024 * 1024;
protected int netRelayRedundancy = 8;
protected int netHandshakeExpiry = 5 * 60 * 1000;
protected int netChannelIdleTimeout = 2 * 60 * 1000;
protected Amount maxVoteAmount = Amount.of(10000000, BOHR);
protected int maxVoteCount = 1000;
protected Amount maxUnVoteAmount = Amount.of(10000000, BOHR);
protected int maxUnVoteCount = 1000;
```

Knownsec

Audit results: After testing, whether the core p2p network communication layer involved in the public chain code has maximum restrictions on traffic and other related communication parameters.

Safety advice: nothing.

4.5.7. Node penalty mechanism【PASS】

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

Risk hazard: Isolate normal nodes.

Risk file:

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

Audit results: After auditing, no relevant risks were found.

Safety advice: nothing.

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 hazard: chain rollback, chain bifurcation.

Risk file:

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

Knownsec

Audit results: After auditing, no relevant risks were found.

Safety advice: nothing.

4.6.2. Chain synchronization logic【PASS】

Audit description: Check whether the scope of the synchronous block, the verification logic, and other asynchronous operations during the period are reasonable.

Risk hazard: chain bifurcation.

Risk file:

Audit process: After testing, the relevant code of the chain synchronization in the public chain code complies with the security coding operation, and no obvious security problems have been found.

Audit results: After auditing, no relevant risks were found.

Safety advice: nothing.

4.7. Block processing

4.7.1. Block hash collision【PASS】

Audit description: Check the construction method of block hash collision and whether the handling of collision is reasonable.

Risk hazard: DoS nodes occupy local storage resources.

Risk file:

knownsec

Audit process: Check the construction method of block hash collision and whether the handling of collision is reasonable.

Audit results: After testing, the core data encryption operations involved in the public chain code all use the sha256 library, which conforms to the secure encoding operation.

Safety advice: nothing.

4.7.2. Block processing resource limit【PASS】

Audit description: Check whether the resource limits of orphan block pool, verification calculation, hard disk addressing, etc. are reasonable.

Risk hazard: DoS nodes occupy local storage and CPU resources. In severe cases, the node program may crash.

Risk file:

Audit process: Check whether the resource limits of orphan block pool, verification calculation, hard disk addressing, etc. are reasonable.

```
protected int netMaxOutboundConnections = 128;
    protected int netMaxInboundConnections = 512;
    protected int netMaxInboundConnectionsPerIp = 5;
    protected int netMaxMessageQueueSize = 4096;
    protected int netMaxFrameBodySize = 128 * 1024;
    protected int netMaxPacketSize = 16 * 1024 * 1024;
    protected int netRelayRedundancy = 8;
    protected int netHandshakeExpiry = 5 * 60 * 1000;
    protected int netChannelIdleTimeout = 2 * 60 * 1000;
    protected Amount maxVoteAmount = Amount.of(10000000, BOHR);
```

Knownsec

```
    protected int maxVoteCount = 1000;
    protected Amount maxUnVoteAmount = Amount.of(10000000, BOHR);
    protected int maxUnVoteCount = 1000;
```

Audit results: After testing, no obvious security issues were found in the logic related to block processing in the public chain code.

Safety advice: nothing.

4.7.3. Block signature processing【PASS】

Audit description: Check whether the block signature content items and verification logic are sufficient.

Risk hazard: Forged blocks can produce unexpected benefits. In severe cases, it can lead to problems such as chain forks and data tampering.

Risk file:

Audit process: Check whether the block signature content items and verification logic are sufficient.

```
@Override
    public synchronized void addBlock(Block block) {
        long number = block.getNumber();
        byte[] hash = block.getHash();

        if (number != genesis.getNumber() && number != latestBlock.getNumber() + 1) {
            logger.error("Adding wrong block: number = {}, expected = {}", number,
latestBlock.getNumber() + 1);
            throw new BlockchainException("Blocks can only be added sequentially");
        }

        // [1] update block
```

Knownsec

```
            blockDB.put(Bytes.merge(TYPE_BLOCK_HEADER_BY_NUMBER, Bytes.of(number)),
block.getEncodedHeader());
            blockDB.put(Bytes.merge(TYPE_BLOCK_TRANSACTIONS_BY_NUMBER,
Bytes.of(number)), block.getEncodedTransactions());
            blockDB.put(Bytes.merge(TYPE_BLOCK_RESULTS_BY_NUMBER, Bytes.of(number)),
block.getEncodedResults());
            blockDB.put(Bytes.merge(TYPE_BLOCK_VOTES_BY_NUMBER, Bytes.of(number)),
block.getEncodedVotes());

            indexDB.put(Bytes.merge(TYPE_BLOCK_NUMBER_BY_HASH, hash),
Bytes.of(number));

            // [2] update transaction indices
            List<Transaction> txs = block.getTransactions();
            Pair<byte[], List<Integer>> transactionIndices =
block.getEncodedTransactionsAndIndices();
            Pair<byte[], List<Integer>> resultIndices = block.getEncodedResultsAndIndices();
            Amount feeReward = Block.getFeeReward(block);

            for (int i = 0; i < txs.size(); i++) {
                Transaction tx = txs.get(i);
                TransactionResult result = block.getResults().get(i);

                TransactionIndex index = new TransactionIndex(number,
transactionIndices.getRight().get(i),
                        resultIndices.getRight().get(i));
                indexDB.put(Bytes.merge(TYPE_TRANSACTION_INDEX_BY_HASH,
tx.getHash()), index.toBytes());

                // [3] update transaction_by_account index
                addTransactionToAccount(tx, tx.getFrom());
                if (!Arrays.equals(tx.getFrom(), tx.getTo())) {
                    addTransactionToAccount(tx, tx.getTo());
```

```
            }

            // index internal transactions
            for (BohrInternalTransaction internalTx : result.getInternalTransactions()) {
                addInternalTransactionToAccount(tx, internalTx, internalTx.getFrom());
                if (!Arrays.equals(internalTx.getFrom(), internalTx.getTo())) {
                    addInternalTransactionToAccount(tx, internalTx, internalTx.getTo());
                }
            }
        }

        if (number != genesis.getNumber()) {
            // [4] coinbase fee reward transaction
            Transaction tx = new Transaction(config.network(),
                    TransactionType.COINBASE,
                    block.getCoinbase(),
                    feeReward,
                    Amount.ZERO,
                    block.getNumber(),
                    block.getTimestamp(),
                    Bytes.EMPTY_BYTES);
            tx.sign(Constants.COINBASE_KEY);
            indexDB.put(Bytes.merge(TYPE_TRANSACTION_INDEX_BY_HASH,
tx.getHash()), tx.toBytes());
            indexDB.put(Bytes.merge(TYPE_BLOCK_COINBASE_BY_NUMBER,
Bytes.of(block.getNumber())), tx.getHash());
            addTransactionToAccount(tx, block.getCoinbase());

            //[5] coinbase block reward transaction , per 28800 generate
            Amount blockReward = Block.getBlockReward(block, config);
            if (blockReward.isPositive()) {
                Transaction blockRewardTx = new Transaction(config.network(),
                        TransactionType.REWARD,
```


Knownsec

```
                        Constants.BOHR_GAME_REWARD_GENERATE_ADDRESS,
                        blockReward,
                        Amount.ZERO,
                        block.getNumber(),
                        block.getTimestamp(),
                        Bytes.EMPTY_BYTES);

                blockRewardTx.sign(Constants.COINBASE_KEY);
                indexDB.put(Bytes.merge(TYPE_TRANSACTION_INDEX_BY_HASH,
blockRewardTx.getHash()), blockRewardTx.toBytes());

indexDB.put(Bytes.merge(TYPE_BLOCK_COINBASE_REWARD_BY_NUMBER,
Bytes.of(block.getNumber())), blockRewardTx.getHash());
                addTransactionToAccount(blockRewardTx,
Constants.BOHR_GAME_REWARD_GENERATE_ADDRESS);
            }

            // [6] update validator statistics
            List<String> validators = getValidators();
            String primary = config.spec().getPrimaryValidator(validators, number, 0,
                    this.isForkActivated(UNIFORM_DISTRIBUTION));
            adjustValidatorStats(block.getCoinbase(), StatsType.FORGED, 1);
            if (primary.equals(Hex.encode(block.getCoinbase()))) {
                adjustValidatorStats(Hex.decode0x(primary), StatsType.HIT, 1);
            } else {
                adjustValidatorStats(Hex.decode0x(primary), StatsType.MISSED, 1);
            }
        }

        // [7] update validator set
        if (number % config.spec().getValidatorUpdateInterval() == 0) {
            updateValidators(block.getNumber());
        }
```

Knownsec

```
        // [8] update latest_block
        latestBlock = block;
        indexDB.put(Bytes.of(TYPE_LATEST_BLOCK_NUMBER), Bytes.of(number));

        for (BlockchainListener listener : listeners) {
            listener.onBlockAdded(block);
        }

        activateForks();
    }
    /**
     * Sign this transaction.
     *
     * @param key
     * @return
     */
    public Transaction sign(Key key) {
        this.signature = key.sign(this.hash);
        return this;
    }
    /**
     * Signs a message.
     *
     * @param message
     *                message
     * @return
     */
    public Signature sign(byte[] message) {
        try {
            byte[] sig;
            if (Native.isEnabled()) {
                sig = Native.sign(message, Bytes.merge(sk.getSeed(), sk.getAbyte()));
```

knownsec

```
            } else {
                EdDSAEngine engine = new EdDSAEngine();
                engine.initSign(sk);
                sig = engine.signOneShot(message);
            }

            return new Signature(sig, pk.getAbyte());
        } catch (InvalidKeyException | SignatureException e) {
            throw new CryptoException(e);
        }
    }
```

Audit results: After testing, no obvious security issues were found in the logic related to block processing in the public chain code.

Safety advice: nothing.

4.7.4. Block verification processing【PASS】

Audit description: Check whether there is a security risk in the block verification processing logic.

Risk hazard: chain bifurcation.

Risk file:

Audit process: Audit the logic code related to the block verification process.

```
/**
     * Validate the block. Votes are validated only if validateVotes is true.
     * @param block
     * @param asTrack
     * @param dsTrack
     * @param validateVotes
     * @return
```

Knownsec

```
     */
    protected boolean validateBlock(Block block, AccountState asTrack, DelegateState dsTrack,
                                    boolean validateVotes) {
        try {
            BlockHeader header = block.getHeader();
            List<Transaction> transactions = block.getTransactions();

            // [1] check block header
            Block latest = this.getLatestBlock();
            if (!block.validateHeader(header, latest.getHeader())) {
                logger.error("Invalid block header");
                return false;
            }

            // [?] additional checks by block importer
            // - check points
            if (config.checkpoints().containsKey(header.getNumber()) &&
                    !Arrays.equals(header.getHash(),
config.checkpoints().get(header.getNumber()))) {
                logger.error("Checkpoint validation failed, checkpoint is {} => {}, getting {}",
header.getNumber(),
                        Hex.encode0x(config.checkpoints().get(header.getNumber())),
                        Hex.encode0x(header.getHash()));
                return false;
            }

            // [2] check transactions
            if (!block.validateTransactions(header, transactions, config.network())) {
                logger.error("Invalid transactions");
                return false;
            }
            if (transactions.stream().anyMatch(tx -> this.hasTransaction(tx.getHash()))) {
                logger.error("Duplicated transaction hash is not allowed");
```

Knownsec

```
                return false;
            }

            // [3] evaluate transactions
            TransactionExecutor transactionExecutor = new TransactionExecutor(config,
blockStore,isVMEnabled(),isVotingPrecompiledUpgraded());
            List<TransactionResult> results = transactionExecutor.execute(transactions, asTrack,
dsTrack,
                    new BohrBlock(block.getHeader(), config.spec().maxBlockGasLimit()),0);

            if (!block.validateResults(header, results)) {
                logger.error("Invalid transaction results");
                return false;
            }
            block.setResults(results); // overwrite the results

            // [4] evaluate votes
            if (validateVotes) {
                return validateBlockVotes(block);
            }

            return true;
        } catch (Exception e) {
            logger.error("Unexpected exception during block validation", e);
            return false;
        }
    }
    @Override
    public synchronized void addBlock(Block block) {
        long number = block.getNumber();
        byte[] hash = block.getHash();

        if (number != genesis.getNumber() && number != latestBlock.getNumber() + 1) {
```

Knownsec

```
            logger.error("Adding wrong block: number = {}, expected = {}", number,
latestBlock.getNumber() + 1);
            throw new BlockchainException("Blocks can only be added sequentially");
        }

        // [1] update block
        blockDB.put(Bytes.merge(TYPE_BLOCK_HEADER_BY_NUMBER, Bytes.of(number)),
block.getEncodedHeader());
        blockDB.put(Bytes.merge(TYPE_BLOCK_TRANSACTIONS_BY_NUMBER,
Bytes.of(number)), block.getEncodedTransactions());
        blockDB.put(Bytes.merge(TYPE_BLOCK_RESULTS_BY_NUMBER, Bytes.of(number)),
block.getEncodedResults());
        blockDB.put(Bytes.merge(TYPE_BLOCK_VOTES_BY_NUMBER, Bytes.of(number)),
block.getEncodedVotes());

        indexDB.put(Bytes.merge(TYPE_BLOCK_NUMBER_BY_HASH, hash),
Bytes.of(number));

        // [2] update transaction indices
        List<Transaction> txs = block.getTransactions();
        Pair<byte[], List<Integer>> transactionIndices =
block.getEncodedTransactionsAndIndices();
        Pair<byte[], List<Integer>> resultIndices = block.getEncodedResultsAndIndices();
        Amount feeReward = Block.getFeeReward(block);

        for (int i = 0; i < txs.size(); i++) {
            Transaction tx = txs.get(i);
            TransactionResult result = block.getResults().get(i);

            TransactionIndex index = new TransactionIndex(number,
transactionIndices.getRight().get(i),
                    resultIndices.getRight().get(i));
```

Knownsec

```
                indexDB.put(Bytes.merge(TYPE_TRANSACTION_INDEX_BY_HASH,
tx.getHash()), index.toBytes());

                // [3] update transaction_by_account index
                addTransactionToAccount(tx, tx.getFrom());
                if (!Arrays.equals(tx.getFrom(), tx.getTo())) {
                    addTransactionToAccount(tx, tx.getTo());
                }

                // index internal transactions
                for (BohrInternalTransaction internalTx : result.getInternalTransactions()) {
                    addInternalTransactionToAccount(tx, internalTx, internalTx.getFrom());
                    if (!Arrays.equals(internalTx.getFrom(), internalTx.getTo())) {
                        addInternalTransactionToAccount(tx, internalTx, internalTx.getTo());
                    }
                }
            }

            if (number != genesis.getNumber()) {
                // [4] coinbase fee reward transaction
                Transaction tx = new Transaction(config.network(),
                        TransactionType.COINBASE,
                        block.getCoinbase(),
                        feeReward,
                        Amount.ZERO,
                        block.getNumber(),
                        block.getTimestamp(),
                        Bytes.EMPTY_BYTES);
                tx.sign(Constants.COINBASE_KEY);
                indexDB.put(Bytes.merge(TYPE_TRANSACTION_INDEX_BY_HASH,
tx.getHash()), tx.toBytes());
                indexDB.put(Bytes.merge(TYPE_BLOCK_COINBASE_BY_NUMBER,
Bytes.of(block.getNumber())), tx.getHash());
```

Knownsec

```
            addTransactionToAccount(tx, block.getCoinbase());

            //[5] coinbase block reward transaction , per 28800 generate
            Amount blockReward = Block.getBlockReward(block, config);
            if (blockReward.isPositive()) {
                Transaction blockRewardTx = new Transaction(config.network(),
                        TransactionType.REWARD,
                        Constants.BOHR_GAME_REWARD_GENERATE_ADDRESS,
                        blockReward,
                        Amount.ZERO,
                        block.getNumber(),
                        block.getTimestamp(),
                        Bytes.EMPTY_BYTES);

                blockRewardTx.sign(Constants.COINBASE_KEY);
                indexDB.put(Bytes.merge(TYPE_TRANSACTION_INDEX_BY_HASH,
blockRewardTx.getHash()), blockRewardTx.toBytes());

indexDB.put(Bytes.merge(TYPE_BLOCK_COINBASE_REWARD_BY_NUMBER,
Bytes.of(block.getNumber())), blockRewardTx.getHash());
                addTransactionToAccount(blockRewardTx,
Constants.BOHR_GAME_REWARD_GENERATE_ADDRESS);
            }

            // [6] update validator statistics
            List<String> validators = getValidators();
            String primary = config.spec().getPrimaryValidator(validators, number, 0,
                    this.isForkActivated(UNIFORM_DISTRIBUTION));
            adjustValidatorStats(block.getCoinbase(), StatsType.FORGED, 1);
            if (primary.equals(Hex.encode(block.getCoinbase()))) {
                adjustValidatorStats(Hex.decode0x(primary), StatsType.HIT, 1);
            } else {
                adjustValidatorStats(Hex.decode0x(primary), StatsType.MISSED, 1);
```

```
            }
        }

        // [7] update validator set
        if (number % config.spec().getValidatorUpdateInterval() == 0) {
            updateValidators(block.getNumber());
        }
        // [8] update latest_block
        latestBlock = block;
        indexDB.put(Bytes.of(TYPE_LATEST_BLOCK_NUMBER), Bytes.of(number));

        for (BlockchainListener listener : listeners) {
            listener.onBlockAdded(block);
        }
        activateForks();
    }
```

Audit results: After the audit, no relevant risks were found.

Safety advice: nothing

4.7.5. Merkle tree root construction method【PASS】

Audit description: Check whether the Merkle tree root construction method is reasonable.

Risk hazard: forging blocks, causing problems such as chain forks.

Risk file:

Audit process: Check whether the Merkle tree root construction method is reasonable.

```
    private Node build(List<Node> nodes) {
        if (nodes.isEmpty()) {
```

Knownsec

```
            return new Node(Bytes.EMPTY_HASH);
        }
        while (nodes.size() > 1) {
            List<Node> list = new ArrayList<>();

            for (int i = 0; i < nodes.size(); i += 2) {
                Node left = nodes.get(i);
                if (i + 1 < nodes.size()) {
                    Node right = nodes.get(i + 1);
                    list.add(new Node(Hash.h256(left.value, right.value), left, right));
                } else {
                    list.add(new Node(left.value, left, null));
                }
            }
            levels++;
            nodes = list;
        }

        levels++;
        return nodes.get(0);
    }
```

Audit results: Audit results: After the audit, no relevant risks were found.

Safety advice: nothing.

4.8. Transaction processing

4.8.1. Transaction signature【PASS】

Audit description: Check whether the signature content items and verification logic of various types of transactions are sufficient.

Knownsec

Risk hazard: same-chain or cross-chain transaction replay, forgery of legal transactions, affecting asset security.

Risk file:

Audit process: After testing, no obvious security issues were found in the transaction signature logic in the public chain code.

```
/**
     * Sign this transaction.
     * @param key
     * @return
     */
    public Transaction sign(Key key) {
        this.signature = key.sign(this.hash);
        return this;
}
    /**
     * Signs a message.
     * @param message
     *        message
     * @return
     */
    public Signature sign(byte[] message) {
        try {
            byte[] sig;
            if (Native.isEnabled()) {
                sig = Native.sign(message, Bytes.merge(sk.getSeed(), sk.getAbyte()));
            } else {
                EdDSAEngine engine = new EdDSAEngine();
                engine.initSign(sk);
                sig = engine.signOneShot(message);
            }
```

Knownsec

```
            return new Signature(sig, pk.getAbyte());
        } catch (InvalidKeyException | SignatureException e) {
            throw new CryptoException(e);
        }
    }
```

Audit results: After auditing, no relevant risks were found.

Safety advice: nothing.

4.8.2. Transaction processing logic【PASS】

Audit description: Check the detection and processing methods for double-spending transactions, repeated transactions, and illegal transactions.

Risk hazard: double spending of assets in the same chain, replay of transactions.

Risk file:

Audit process: Check whether there are security risks in the transaction processing logic design in the public chain.

```
/**
     * Validates the given transaction and add to pool if success.
     *
     * @param tx
     *              a transaction
     * @param isIncludedBefore
     *              whether the transaction is included before
     * @param isFromThisNode
     *              whether the transaction is from this node
     * @return the number of transactions that have been included
     */
    protected ProcessingResult processTransaction(Transaction tx, boolean isIncludedBefore, boolean
isFromThisNode) {
```

Knownsec

```
            int cnt = 0;
            long now = TimeUtil.currentTimeMillis();

            // reject VM transactions that come in before fork
            //if (tx.isVMTransaction()
&& !kernel.getBlockchain().isForkActivated(Fork.VIRTUAL_MACHINE)) {
            //        return new ProcessingResult(0, TransactionResult.Code.INVALID_TYPE);
            //}

            // reject VM transaction with low gas price
            if (tx.isVMTransaction() &&
tx.getGasPrice().lessThan(kernel.getConfig().poolMinGasPrice())) {
                return new ProcessingResult(0, TransactionResult.Code.INVALID_FEE);
            }

            // reject transactions with a duplicated tx hash
            if (kernel.getBlockchain().hasTransaction(tx.getHash())) {
                return new ProcessingResult(0, TransactionResult.Code.INVALID);
            }

            // check transaction timestamp if this is a fresh transaction:
            // a time drift of 2 hours is allowed by default
            if (tx.getTimestamp() < now - kernel.getConfig().poolMaxTransactionTimeDrift()
                    || tx.getTimestamp() > now + kernel.getConfig().poolMaxTransactionTimeDrift())
{
                return new ProcessingResult(0, TransactionResult.Code.INVALID_TIMESTAMP);
            }

            // report INVALID_NONCE error to prevent the transaction from being
            // silently ignored due to a low nonce
            if (tx.getNonce() < getNonce(tx.getFrom())) {
                return new ProcessingResult(0, TransactionResult.Code.INVALID_NONCE);
```

Knownsec

```
        }

        // Check transaction nonce: pending transactions must be executed sequentially
        // by nonce in ascending order. In case of a nonce jump, the transaction is
        // delayed for the next event loop of PendingManager.
        while (tx != null && tx.getNonce() == getNonce(tx.getFrom())) {

            // execute transactions
            AccountState as = pendingAS.track();
            DelegateState ds = pendingDS.track();
            TransactionResult result = new TransactionExecutor(kernel.getConfig(), blockStore,
                    kernel.getBlockchain().isVMEnabled(),
kernel.getBlockchain().isVotingPrecompiledUpgraded())
                    .execute(tx, as, ds, dummyBlock, 0);

            if (result.getCode().isAcceptable()) {
                // commit state updates
                as.commit();
                ds.commit();

                // Add the successfully processed transaction into the pool of transactions
                // which are ready to be proposed to the network.
                PendingTransaction pendingTransaction = new PendingTransaction(tx, result);
                validTxs.add(pendingTransaction);
                cnt++;

                // If a transaction is not included before, send it to the network now
                if (!isIncludedBefore) {
                    // if it is from myself, broadcast it to everyone
                    broadcastTransaction(tx, isFromThisNode);
                }
            } else {
                // exit immediately if invalid
```

Knownsec

```
                return new ProcessingResult(cnt, result.getCode());
            }

            tx = largeNonceTxs.getIfPresent(createKey(tx.getFrom(), getNonce(tx.getFrom())));
            isIncludedBefore = false; // A large-nonce transaction is not included before
        }

        // Delay the transaction for the next event loop of PendingManager. The delayed
        // transaction is expected to be processed once PendingManager has received all
        // all of its preceding transactions from the same address.
        if (tx != null && tx.getNonce() > getNonce(tx.getFrom())) {
            largeNonceTxs.put(createKey(tx), tx);
        }

        return new ProcessingResult(cnt);
    }

    private void broadcastTransaction(Transaction tx, boolean toAllPeers) {
        List<Channel> channels = kernel.getChannelManager().getActiveChannels();

        // If not to all peers, randomly pick n channels
        int n = kernel.getConfig().netRelayRedundancy();
        if (!toAllPeers && channels.size() > n) {
            Collections.shuffle(channels);
            channels = channels.subList(0, n);
        }

        // Send the message
        TransactionMessage msg = new TransactionMessage(tx);
        for (Channel c : channels) {
            if (c.isActive()) {
                c.getMessageQueue().sendMessage(msg);
            }
```

knownsec

```
        }
    }
```

Audit results: After testing, no obvious security issues were found in the transaction processing logic in the public chain code.

Safety advice: nothing.

4.8.3. Transaction processing resource limit【PASS】

Audit description: Check whether resource restrictions such as transaction pools, verification calculations, and hard disk addressing are reasonable..

Risk hazard: DoS nodes occupy local storage and CPU resources. In severe cases, the node program may crash..

Risk file:

Audit process: Check whether resource restrictions such as transaction pools, verification calculations, hard disk addressing, etc. are reasonable

```
const (
    // chainHeadChanSize is the size of channel listening to ChainHeadEvent.
    chainHeadChanSize = 10

    // txSlotSize is used to calculate how many data slots a single transaction
    // takes up based on its size. The slots are used as DoS protection, ensuring
    // that validating a new transaction remains a constant operation (in reality
    // O(maxslots), where max slots are 4 currently).
    txSlotSize = 32 * 1024

    // txMaxSize is the maximum size a single transaction can have. This field has
    // non-trivial consequences: larger transactions are significantly harder and
    // more expensive to propagate; larger transactions also take more resources
```

Knownsec

```
    // to validate whether they fit into the pool or not.
    txMaxSize = 4 * txSlotSize // 128KB
)
```

Audit results: After testing, no obvious security issues were found in the transaction processing logic in the public chain code.

Safety advice: nothing.

4.8.4. Transaction fee setting【PASS】

Audit description: Check whether the transaction fee corresponding to each atomic operation executed by the transaction/contract is proportional to the resource consumption.

Risk hazard: DoS the entire network at a lower cost.

Risk file:

Audit process: Check whether the handling fee corresponding to each atomic operation of transaction processing/contract execution is proportional to resource consumption.

```
  // ========================
    // Chain spec
    // ========================
    protected long maxBlockGasLimit = 30_000_000L; // 30m gas
    protected Amount minTransactionFee = Amount.of(100, MICRO_BOHR);
//0.0001Bohr
    protected Amount minDelegateBurnAmount = Amount.of(1000, BOHR);
    protected long nonVMTransactionGasCost = 5_000L;
```

Audit results: After testing, the transaction fee in the public chain code is designed reasonably, and no related security risks are found.

Safety advice: nothing.

4.9. Contract virtual machine

4.9.1. Contract virtual machine sandbox escape【PASS】

Audit description: Check the virtual machine sandbox logic to determine whether key objects are completely intercepted, test whether attacks such as common character escape are effective, and whether they can access objects outside the sandbox environment.

Risk hazard: Bypassing sandbox restrictions, severe cases may affect the execution logic of other contracts, or cause DoS and remote command execution.

Risk file:

Audit process: Check whether the contract virtual machine intercepts key objects thoroughly, and test whether common character escape and other attacks are effective.

Audit results: After auditing, the contract virtual machine was designed with reference to the official source code of Ethereum, and no suspicious security risks were found.

Safety advice: nothing.

Knownsec

4.9.2. Contract execution logic【PASS】

Audit description: Check whether there is unexpected control flow in the execution logic of the contract, and whether reasonable restrictions are made in each link.

Risk hazard: Unexpected behavior in contract execution, severe cases may lead to data tampering or node program crash.

Risk file:

Audit process: Check whether there is unexpected control flow in the execution logic of the contract, and whether there are reasonable restrictions.

```
/**
     * Create contract for {@link OpCode#CREATE}
     */
    public ProgramResult createContract(DataWord value, DataWord memStart, DataWord memSize, long gas) {
        resetReturnDataBuffer();

        byte[] senderAddress = this.getOwnerAddress().getLast20Bytes();
        BigInteger endowment = value.value();
        RuntimeException exception = verifyCall(senderAddress, endowment);
        if (exception != null) {
            // in case of insufficient balance or call is too deep,
            // throw an exception
            return ProgramResult.createExceptionResult(getGasLeft(), exception);
        }

        long nonce = getRepository().getNonce(senderAddress);
        byte[] contractAddress = HashUtil.calcNewAddress(senderAddress, nonce);
        byte[] programCode = memoryChunk(memStart.intValue(), memSize.intValue());
```

Knownsec

```
        ProgramResult callResult = createContractImpl(value, programCode,
contractAddress, gas);
        setReturnDataBuffer(callResult.getReturnData());
        return callResult;
    }
    /**
     * All stages required to create contract on provided address after initial
     * check
     *
     * @param value
     *                  Endowment
     * @param programCode
     *                  Contract code
     * @param newAddress
     *                  Contract address
     */
    private ProgramResult createContractImpl(DataWord value, byte[] programCode, byte[]
newAddress, long gas) {
        byte[] senderAddress = this.getOwnerAddress().getLast20Bytes();
        boolean contractAlreadyExists = getRepository().exists(newAddress);

        // [1] SPEND GAS
        spendGas(gas, "internal call - create");

        // [2] UPDATE THE NONCE
        // (THIS STAGE IS NOT REVERTED BY ANY EXCEPTION)
        getRepository().increaseNonce(senderAddress);

        // track for reversibility when failure
        Repository track = getRepository().startTracking();

        // [3] SET THE NONCE OF NEW CONTRACT TO ONE?????
```

Knownsec

```
            track.increaseNonce(newAddress);

            // [4] TRANSFER THE BALANCE
            transfer(track, senderAddress, newAddress, value.value());

            // [5] COOK AN INTERNAL TRANSACTION
            InternalTransaction internalTx = addInternalTx(OpCode.CREATE, senderAddress,
EMPTY_BYTE_ARRAY,
                    getRepository().getNonce(senderAddress), value.value(), programCode,
gas);
            if (logger.isDebugEnabled()) {
                logger.debug("CREATE: {}", internalTx);
            }

            // [6] EXECUTE THE CODE
            ProgramResult result;
            if (contractAlreadyExists) {
                result = ProgramResult.createExceptionResult(gas,
                        new BytecodeExecutionException("Account already exists: 0x"
                                + HexUtil.toHexString(newAddress)));
            } else if (isNotEmpty(programCode)) {
                ProgramInvoke programInvoke =
programInvokeFactory.createProgramInvoke(this,
                        getOwnerAddress(),
                        DataWord.of(newAddress),
                        gas,
                        value,
                        EMPTY_BYTE_ARRAY,
                        track,
                        this.invoke.getBlockStore(),
                        false);
                Program program = new Program(programCode, programInvoke, spec);
```

Knownsec

```
                new VM(spec).play(program);
                result = program.getResult();
            } else {
                result = ProgramResult.createEmptyResult(gas);
            }

            // [7] SAVE THE CONTRACT CODE
            if (result.getException() == null && !result.isRevert()) {
                byte[] code = result.getReturnData();
                long storageCost = getLength(code) *
spec.getFeeSchedule().getCREATE_DATA();

                if (result.getGasLeft() < storageCost) {
                    if (!spec.createEmptyContractOnOOG()) {
                        result.setReturnData(EMPTY_BYTE_ARRAY);
                        result.setException(ExceptionFactory.notEnoughSpendingGas("No
gas to return just created contract",
                                storageCost, this));
                    } else {
                        // free of charge; return data un-touched
                        track.saveCode(newAddress, EMPTY_BYTE_ARRAY);
                    }
                } else if (getLength(code) > spec.maxContractSize()) {
                    result.setReturnData(EMPTY_BYTE_ARRAY);
                    result.setException(ExceptionFactory.notEnoughSpendingGas("Contract
size too large: "
                            + getLength(result.getReturnData()), storageCost, this));
                } else {
                    result.spendGas(storageCost);
                    track.saveCode(newAddress, code);
                }
            }
```

Knownsec

```
        // [8] POST EXECUTION PROCESSING
        if (result.getException() == null && !result.isRevert()) {
            // commit changes
            track.commit();
            // in success push the address into the stack
            stackPush(DataWord.of(newAddress));
        } else {
            if (logger.isDebugEnabled()) {
                logger.debug("Contract run halted by Exception: contract: [{}],
exception: [{}]",
                        HexUtil.toHexString(newAddress),
                        result.getException());
            }
            internalTx.reject();
            result.rejectInternalTransactions();

            track.rollback();
            stackPushZero();
        }

        // [9] REFUND THE REMAIN GAS
        if (result.getException() == null) {
            long refundGas = result.getGasLeft();
            if (refundGas > 0) {
                refundGas(refundGas, "remaining gas from create");
            }
        }
        // [10] MERGE RESULT INTO PARENT
        getResult().merge(result);
        return result;
    }
```

Audit results: After auditing, no relevant risks were found.

Knownsec

Safety advice: nothing.

4.10. Account system

4.10.1. Account system authentication【PASS】

Audit description: Check whether there is a problem with the authentication logic.

Risk hazard: Forged identities bypass permission checks, and severe cases can threaten asset security.

Risk file:

Audit process: Check whether the logic design of the relevant authority operation of the wallet account system in the public chain code is reasonable.

Audit results: After auditing, it was found that the API is closed by default and not open to the outside world. Only local access is allowed and identity authentication is required. When operating an account through the UI, you need to enter the wallet password for authentication.

Safety advice: nothing.

4.10.2. Account password policy【Low】

Audit description: Check the account password policy level in the public chain code and whether weak passwords are allowed.

Risk hazard: Weak password strategy brings the risk of password blasting, and it is recommended to use a high-strength password strategy.

Knownsec

Risk file: bohr-core\src\main\java\org\bohr\cli\BohrCli.java

Audit process: Audit the account creation logic and check the corresponding password security policy.

Audit results: After auditing, it was found that the password complexity was not verified when the wallet was created, and the password passed by the user was directly used as the wallet account password, allowing the setting of weak passwords.

```
395     /**
396      * Create a new wallet with a new password from input and save the wallet file
397      * to disk
398      *
399      * @return created new wallet, or null if it failed to create the wallet
400      */
401     protected Wallet createNewWallet() {
402         String newPassword = readNewPassword("EnterNewPassword", "ReEnterNewPassword");
403         if (newPassword == null) {
404             return null;
405         }
406
407         setPassword(newPassword);
408         Wallet wallet = loadWallet();
409         if (!wallet.unlock(newPassword) || !wallet.flush()) {
410             logger.error("CreateNewWalletError");
411             exit(SystemUtil.Code.FAILED_TO_WRITE_WALLET_FILE);
412             return null;
413         }
414
415         return wallet;
416     }
```

image: New wallet logic design

```
269     /**
270      * Sets the password.
271      *
272      * @param password
273      */
274     protected void setPassword(String password) {
275         this.password = password;
276     }
277
```

image: Directly set without verifying the password complexity

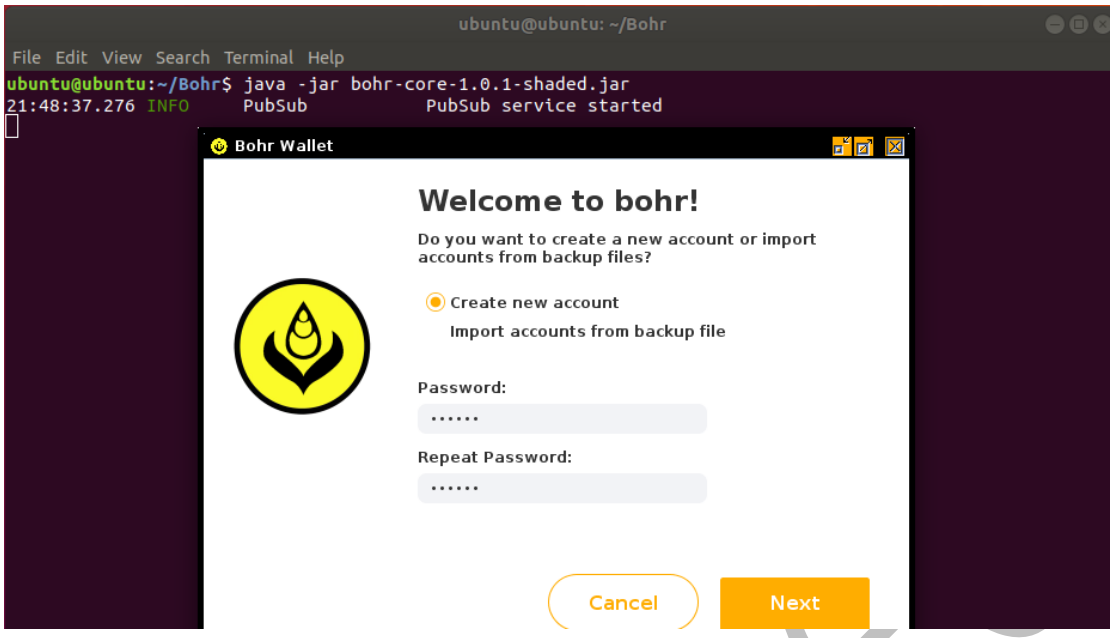


image: Allow weak passwords like 123456 as account passwords

Safety advice: Use a strong password strategy, such as: uppercase and lowercase letters + numbers + special symbols.

4.10.3. Account CRUD logic【PASS】

Audit description: Check whether there are illegal boundary conditions in the CRUD logic of the account system, and when the operation takes effect, and test whether there are loopholes and the impact on nodes and data.

Risk hazard: Unauthorized unauthorized operation of account CRUD operations.

Risk file:

Audit process: Check whether there are illegal boundary conditions in the CRUD logic of the account system.

Audit results: After auditing, it was found that the API is closed by default and not open to the outside world. Only local access is allowed and identity authentication

is required. When operating an account through the UI, you need to enter the wallet password for authentication.

Safety advice: nothing.

4.11. *other*

4.11.1.Database Security【PASS】

Audit description: Check whether there are injection points in all database statements, test whether there are injection points and the impact that can be caused. Test the database operation method for logical errors.

Risk hazard: tampering with the database, severe cases may lead to remote command execution.

Risk file:

Audit process: Check whether there are injection points in all database statements.

Audit results: After auditing, no relevant risks were found.

Safety advice: nothing.

4.11.2.Message Processing Security【PASS】

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

Risk hazard: remote command execution, occupation of local storage resources, and node program crashes, etc.

Risk file:

Knownsec

Audit process: check the deserialization of messages and other operations.

Audit results: After auditing, no relevant risks were found.

Safety advice: nothing.

4.11.3. Thread safety【PASS】

Audit description: Analyze multi-threaded shared resources and check the lock status of shared resources.

Risk hazard: data competition, atomic destruction of things, node process crashes, etc.

Risk file:

Audit process: Analyze multi-threaded shared resources and check the lock status of shared resources.

Audit results: After auditing, no relevant risks were found.

Safety advice: nothing.

4.11.4. Security of environment variables【PASS】

Audit description: Check the dependence on environment variables, and test the influence of controllable environment variables on the node program.

Risk hazard: Affect the safety of funds, etc., analyze according to specific conditions.

Risk file:

Knownsec

Audit process: Audit and check the logic related to environment variables in the public chain code.

Audit results: After testing, no obvious security problems were found in the logic related to environment variables in the public chain code.

Safety advice: nothing.

4.11.5.Centralization degree detection【PASS】

Audit description: Identify whether there is an over-centralized design in the system design.

Risk hazard: Centralized design is easy to concentrate risks. Once a single point of failure occurs, it can lead to serious problems such as data tampering and chain bifurcation.

Risk file:

Audit process: Identify whether there is an over-centralized design in the system design.

Audit results: After testing, no related security issues were found in the centralized logic of the public chain code.

Safety advice: nothing.

4.11.6.File permission security【PASS】

Audit description: Check the relevant log files and whether the permissions of the keystore file are correctly set.

Knownsec

Risk hazard: File permissions are one of the lowest level security settings in the system to ensure that files can be operated by available users. Improper file permissions settings may lead to risks such as file tampering and information leakage.

Risk file:

Audit process: Check whether the file permission setting in the public chain code is reasonable.

Audit results: After auditing, no relevant risks were found.

Safety advice: nothing.

4.11.7.Concurrency security risks【PASS】

Audit description: Check whether there are concurrent security risks in the map, slice and other types of operations in the public chain code.

Risk hazard: When multiple threads share data, results that do not match expectations may occur, which may cause data loss, value overwriting, and other issues.

Risk file:

Audit process: Check whether there are concurrent security risks in operations such as map and slice in the public chain code.

Audit results: After auditing, no relevant risks were found.

Safety advice: nothing.

Knownsec

4.11.8. Third-party dependent libraries【Low】

Audit description: Check whether there are outdated third-party dependent risk libraries in the public chain code.

Risk hazard: The use of outdated third-party dependent risk libraries will increase the risk of being attacked.

Risk file:

Audit process: Check whether the third-party dependent libraries in the pom file have outdated risk dependent libraries.

```
<!-- JSON -->
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-core</artifactId>
        <version>2.9.9</version>
    </dependency>
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
        <version>2.9.9.2</version>
    </dependency>
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-annotations</artifactId>
        <version>2.9.9</version>
    </dependency>
```

Audit results: After audit, it was found that the outdated Jackson-databind risk library was used.

Safety advice: Use the latest version of Jackson-databind library.

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