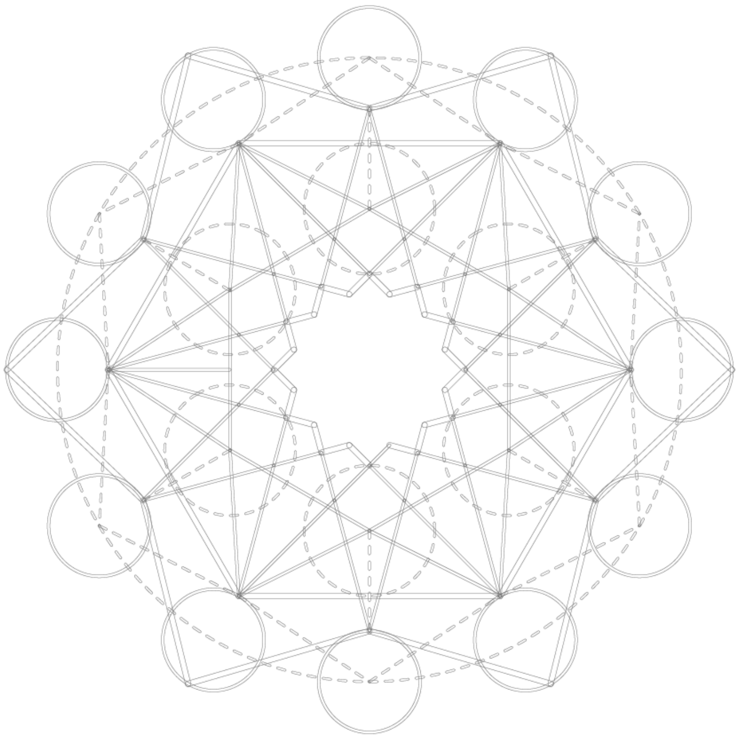
KNOWNSEC
Blockchain Lab

Blockchain Security Audit Report



Version description

The revision	Date	Revised	Version
Write documentation	**20211230**	**Knownsec blockchain security team**	**V1.0**

Document information

Title	Version	Document Number	Type
KORTHO Blockchain Security Audit Report	**V1.0**	**124cf8cb006e4dd7af521a70fc9252c5**	**Open to project team**

Statement

KNOWNSEC Blockchain Lab only issues this report for facts that have occurred or existed before the issuance of this report, and assumes corresponding responsibilities for this. KNOWNSEC Blockchain Lab is unable to determine the security status of its smart contracts 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 Blockchain Lab '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 Blockchain Lab shall not be liable for any losses and adverse effects caused thereby.

Directory

1. Summarize

The effective test time of this report is **from November 23, 2021 to December 30, 2021**. During this period, the security and standardization of the **KORTHO public chain** will be audited and used as the statistical basis for the report.

KNOWNSEC blockchain system security test method:

White box test

Through the source code of the program, SDK conducts security audit and dynamically debugs vulnerabilities to node p2p and RPC.

Gray box test

Tool security scans of source code to identify vulnerabilities that could lead to anomalies.

Black box test

Simulate an attacker performing a security test attack on a node to see if the node is responding properly.

Through this code audit, KNOWNSEC engineers found that there are 0 kinds of common security risks in the target code, a total of 0 security problem, of which 0 serious-risk code files or functions, 0 high-risk code files or functions, 0 medium-risk code files or functions, 0 low-risk code files or functions, **comprehensive assessment as Pass.**

Since this testing process is conducted in a non-production environment, all code is up-to-date, the testing process communicates with the relevant interface person, and the relevant test operation is carried out under the control of operational risks to avoid production and operational risks and code security risks in the testing process.

Report information for this audit:

Report No: 124cf8cb006e4dd7af521a70fc9252c5

Report query address link:

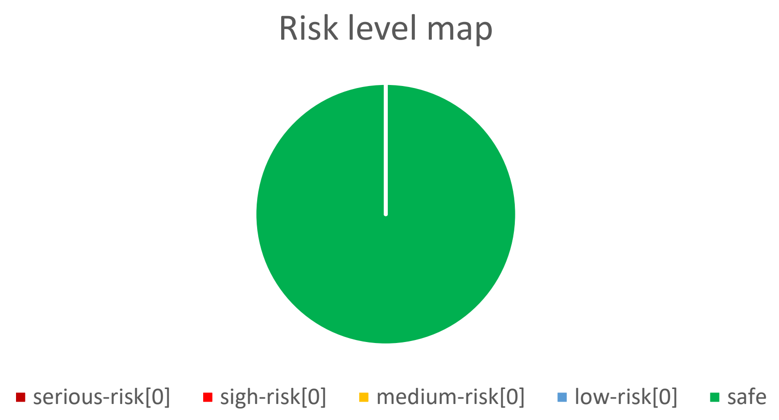
https://attest.im/attestation/searchResult?qurey=124cf8cb006e4dd7af521a70fc9252c5

2. Vulnerability analysis

2.1. Vulnerability level distribution

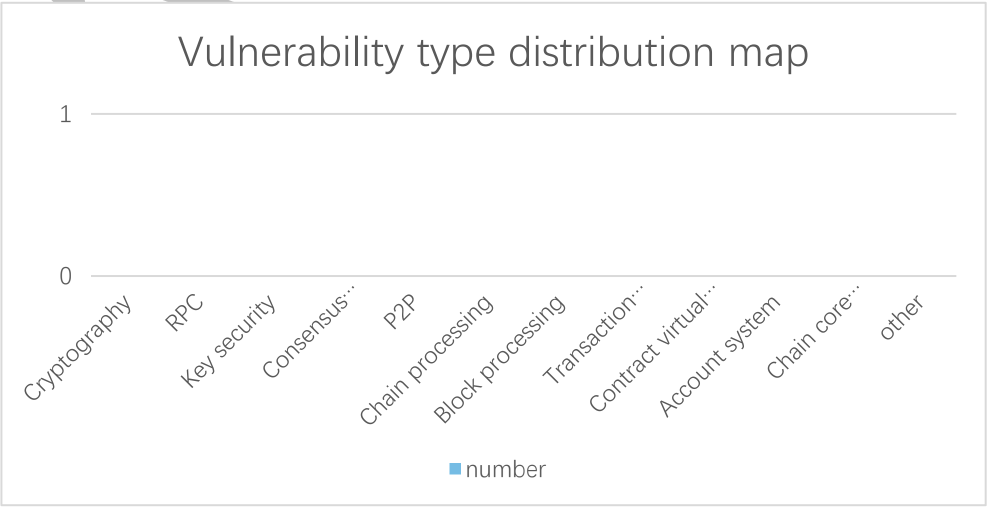
The vulnerability risk is stated by rating:

Number of vulnerability risk levels			
Serious-risk	high-risk	medium-risk	low-risk
0	0	0	0



2.2. The vulnerability type distribution

This vulnerability is counted by type:



3. Project analysis

3.1. Introduction to the project

Since human society entered the 21st century, information technology has developed rapidly, especially driven by computers and the Internet, people have stepped into the era of information Internet, so that human beings can share and connect information in all corners of the world. Work together under the family. However, with the continuous Internetization and the convenience brought by technology to mankind, the problem of data security has become increasingly serious. Nowadays, information access permissions are chaotic, data information leakage is serious, smart terminals are compromised, and other problems follow one after another. When faced with various information plagiarism and terminal data analysis, people will have no privacy and be seen. How to optimize the online world Data security and ownership confirmation will be important issues that people have to reexamine. To this end, the Kortho Chain blockchain ecosystem has emerged in due course, providing a brand new blockchain solution for data encryption and desensitization, transaction information storage and data confirmation in the Internet field. Ketu Chain uses the decentralization of the blockchain and distributed access to the database and the token economy to promote the ecologicalization of data confirmation, including data information traceability, data chaining, and equity token incentives. Data verification can be applied to various fields of the Internet and even the physical industry. For example, in the field of e-commerce, it can effectively ensure the quality and safety of goods. From manufacturers to merchants to consumers, the production and circulation data of every product is uploaded to On the blockchain, while ensuring the quality of the goods purchased by consumers, the transaction timestamp is put on the chain to ensure the rights and interests of consumers, promote the quality of consumer goods and the integrity of merchants, and thereby increase the production capacity of the consumer ecology. Participants in the ecology measure the incentives of their equity tokens by their contribution value. The higher the contribution value, the more equity token rewards they will receive.

Official address: https://www.kortho.org/

Project White Paper: https://www.kortho.org/file/kortho.pdf

Main coding language: Go

3. 2. The scope of the audit

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. The directory structure

Core code directory structure:

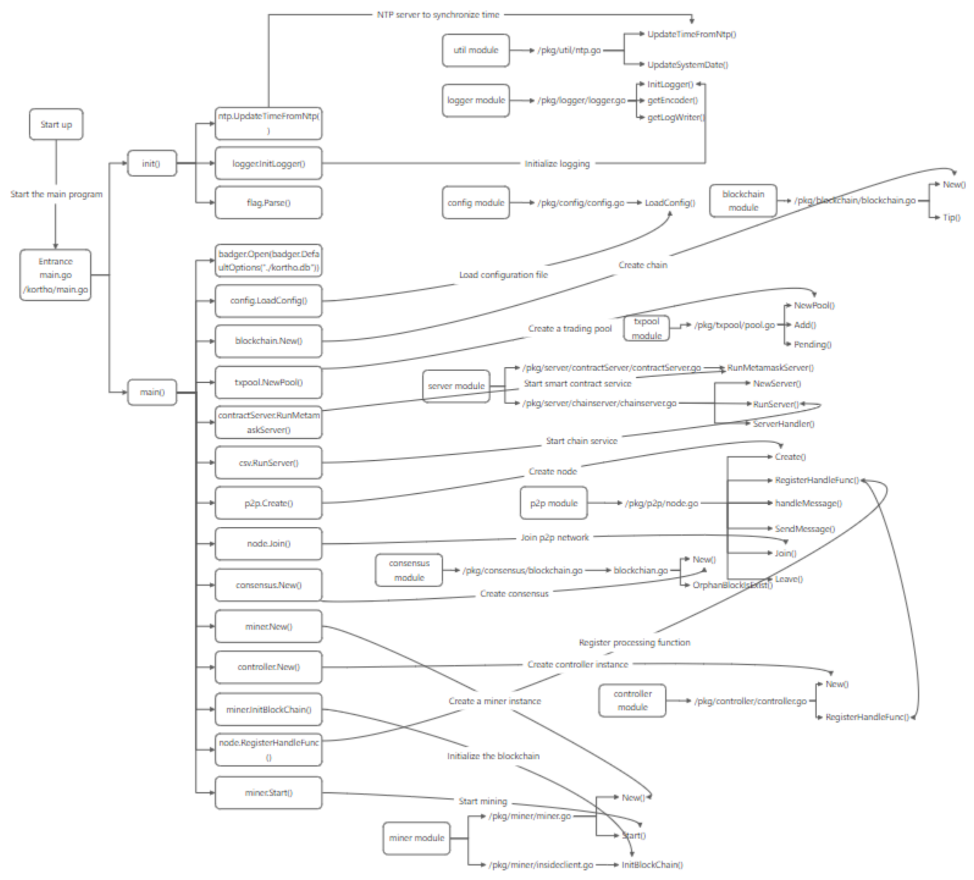
```
.
├── main.go
└── pkg
    ├── address
    │   ├── address.go
    │   ├── address_test.go
    │   └── default.go
    ├── block
    │   ├── block_test.go
    │   └── types.go
    ├── blockchain
    │   ├── addAndDel.go
    │   ├── blockchain.go
    │   ├── blockchain_test.go
    │   ├── contract.go
    │   ├── distribute.go
    │   ├── freeze.go
    │   ├── interface.go
    │   ├── pledge.go
    │   ├── state.go
    │   ├── token.go
    │   ├── types.go
    │   └── writeState.go
    ├── config
    │   ├── config.go
    │   └── config_test.go
    ├── consensus
    │   ├── blockchain.go
    │   ├── chain.go
```

```
|    └─── syncblock.go
├─── contract
|    ├─── evm
|    |    └─── evm.go
|    ├─── exec
|    |    ├─── exec.go
|    |    ├─── exec_test.go
|    |    └─── types.go
|    ├─── main.go
|    └─── parser
|         ├─── ctypes.go
|         ├─── operate.go
|         ├─── otypes.go
|         ├─── parser.go
|         ├─── parser_test.go
|         ├─── ptypes.go
|         ├─── word.go
|         └─── wtypes.go
├─── controller
|    ├─── controller.go
|    └─── pool.go
├─── crypto
|    ├─── signature.go
|    └─── sigs
|         ├─── ed25519
|         |    ├─── ed25519_bench_test.go
|         |    ├─── ed25519.go
|         |    └─── ed25519_test.go
|         ├─── secp
|         |    ├─── secp.go
|         |    └─── secp_test.go
|         └─── sigs.go
├─── logger
```

```
│   ├── config.go
│   └── logger.go
├── miner
│   ├── cx
│   │   └── main.go
│   ├── grpcclient.go
│   ├── hash
│   │   ├── group.go
│   │   ├── hash.go
│   │   └── types.go
│   ├── insideclient.go
│   ├── miner.go
│   └── pool.go
├── p2p
│   ├── broadcast.go
│   ├── config.go
│   ├── delegate.go
│   ├── eventDelegate.go
│   ├── lamport.go
│   ├── messages.go
│   ├── messages_test.go
│   └── node.go
├── server
│   ├── chainserver
│   │   ├── chainserver.go
│   │   └── types.go
│   ├── contractServer
│   │   ├── api
│   │   │   ├── api.go
│   │   │   └── types.go
│   │   ├── client
│   │   │   ├── client.go
│   │   │   └── interface.go
```

```
|     |     └──── contractServer.go
|     ├──── grpcserver
|     |     ├──── grpcserver.go
|     |     ├──── grpcserver_test.go
|     |     └──── message
|     |           └──── message.pb.go
|     ├──── interceptor.go
|     ├──── limiter.go
|     └──── rpcserver
|           ├──── pb
|           |     └──── rpcserver.pb.go
|           └──── rpcserver.go
├──── storage
|     ├──── merkle
|     |     ├──── merkle.go
|     |     ├──── merkle_test.go
|     |     └──── types.go
|     ├──── miscellaneous
|     |     └──── miscellaneous.go
|     ├──── store
|     |     ├──── bg
|     |     |     ├──── bgdb
|     |     |     |     └──── bgdb.go
|     |     |     ├──── bg.go
|     |     |     ├──── iterator.go
|     |     |     └──── types.go
|     |     └──── types.go
|     └──── typeclass
|           ├──── eq.go
|           ├──── ordBytes.go
|           ├──── ordFloat.go
|           ├──── ord.go
|           ├──── ordInt.go
|           ├──── ordString.go
|           ├──── ordUint64.go
|           ├──── read.go
|           └──── show.go
├──── transaction
|     ├──── contracttransaction.go
|     ├──── finishedtransaction.go
|     ├──── signedtransaction.go
|     ├──── sign.go
|     ├──── transaction.go
|     └──── transaction_test.go
├──── txpool
|     ├──── config.go
|     ├──── pool.go
|     ├──── queue.go
|     └──── queue_test.go
└──── util
      ├──── addrcodec
      |     ├──── base58.go
      |     └──── ED25519.go
      ├──── difficulty
      |     ├──── difficuley_test.go
      |     └──── difficulty.go
      ├──── math
      |     └──── overflow.go
      └──── ntp
            ├──── ntp.go
            └──── ntp_test.go
```

3.4. Start the process



3.5. A list of P2P-related methods

func Create(conf Config)

*func (n *Node) RegisterHandleFunc(f func([]byte) error)*

*func (n *Node) handleMessage(msg *message)*

*func (n *Node) SendMessage(msgType messageType, buf []byte)*

*func (n *Node) Join(existing []string)*

*func (n *Node) Leave()*

*func (n *Node) State()*

*func (n *Node) Members()*

3.6. A list of API core interfaces

```
func NewMetamaskServer(bc blockchain.Blockchains, tp *txpool.Pool, cfg *config.CfgInfo)
 *Server
func (s *Server) HandRequest(w http.ResponseWriter, req *http.Request)
func (s *Server) eth_chainId()
func (s *Server) net_version()
func (s *Server) eth_accounts()
func (s *Server) eth_signTransaction(mp map[string]interface{})
func (s *Server) eth_sendRawTransaction(rawTx string)
func (s *Server) eth_call(mp map[string]interface{})
func (s *Server) eth_blockNumber()
func (s *Server) eth_getBalance(from string)
func (s *Server) eth_getBlockByHash(hash string, bl bool)
func (s *Server) eth_getBlockByNumber(num uint64, bl bool)
func (s *Server) eth_getTransactionByHash(hash string)
func (s *Server) eth_getCode(addr string)
func (s *Server) eth_getTransactionCount(addr string)
func (s *Server) eth_gasPrice()
func (s *Server) eth_estimateGas(mp map[string]interface{})
func (s *Server) eth_getTransactionReceipt(hash string)
func (s *Server) eth_getLogs(mp map[string]interface{})
func (s *Server) web3_clientVersion()
func (s *Server) eth_getStorageAt(mp map[string]interface{})
func (s *Server) ktoBlockToEthBlock(b *block.Block, bl bool)
func (s *Server) ktoTxsToEthTxs(block *block.Block, ktxs []*transaction.FinishedTransaction)
func (s *Server) getTxsHashes(ktxs []*transaction.FinishedTransaction)
func (s *Server) ktoTxToTxReceipt(tx *transaction.FinishedTransaction, block *block.Block)
```

3.7. The list of third-party library references

```
    github.com/BurntSushi/toml v0.4.0 // indirect
    github.com/StackExchange/wmi v1.2.1 // indirect
```

```
github.com/beevik/ntp v0.3.0
github.com/bluele/gcache v0.0.2
github.com/btcsuite/btcd v0.22.0-beta // indirect
github.com/btcsuite/btcutil v1.0.3-0.20201208143702-a53e38424cce
github.com/buaazp/fasthttprouter v0.1.1
github.com/cespare/xxhash/v2 v2.1.2 // indirect
github.com/deckarep/golang-set v1.7.1 // indirect
github.com/dgraph-io/badger v1.6.2
github.com/dgraph-io/ristretto v0.1.0 // indirect
github.com/ethereum/go-ethereum v1.10.8
github.com/fxamacker/cbor/v2 v2.3.0
github.com/go-stack/stack v1.8.1 // indirect
github.com/gofrs/uuid v4.1.0+incompatible
github.com/gogf/gf v1.16.6
github.com/goinggo/mapstructure v0.0.0-20140717182941-194205d9b4a9
github.com/golang/glog v1.0.0
github.com/golang/protobuf v1.5.2
github.com/golang/snappy v0.0.4 // indirect
github.com/hashicorp/memberlist v0.2.4
github.com/mattn/go-runewidth v0.0.13 // indirect
github.com/mr-tron/base58 v1.1.3
github.com/prometheus/tsdb v0.10.0 // indirect
github.com/shirou/gopsutil v3.21.7+incompatible // indirect
github.com/spf13/viper v1.8.1
github.com/stretchr/testify v1.7.0
github.com/syndtr/goleveldb v1.0.1-0.20210305035536-64b5b1c73954
github.com/thep0y/go-logger v1.1.1
github.com/tklauser/go-sysconf v0.3.9 // indirect
github.com/valyala/fasthttp v1.30.0
github.com/whyrusleeping/cbor-gen v0.0.0-20210713220151-be142a5ae1a8
go.uber.org/zap v1.18.1
golang.org/x/crypto v0.0.0-20210817164053-32db794688a5
golang.org/x/net v0.0.0-20210825183410-e898025ed96a // indirect
golang.org/x/sys v0.0.0-20210921065528-437939a70204 // indirect
golang.org/x/text v0.3.7 // indirect
golang.org/x/time v0.0.0-20210723032227-1f47c861a9ac
golang.org/x/tools v0.1.6 // indirect
google.golang.org/genproto v0.0.0-20210830153122-0bac4d21c8ea // indirect
google.golang.org/grpc v1.40.0
google.golang.org/protobuf v1.27.1
gopkg.in/natefinch/lumberjack.v2 v2.0.0
```

4. Cryptology

4.1. Cryptology

4.1.1. Random number range and probability distribution 【SAFE】

Audit Note: 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 even distribution).

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

Risk file:

Audit process: After testing, the random number in the public chain code is generated by the Reader in the crypto/rand package, which is a global and shared strong random number generator.

```
func main() {
        t := time.Now()
        data := make([]byte, 1<<10)
        bits := uint32(0x20000000)
        //bits := uint32(0x207fffff)
        target := CompactToBig(bits)
        cnt := 0
        for {


                _, err := rand.Read(data)
                if err != nil {
                        log.Fatal(err)
                }
```

```
            if HashToBig(hash.Hash(data)).Cmp(target) <= 0 {
                    fmt.Printf("sucess: %v\n", time.Now().Sub(t))
                    cnt++
                    if cnt == Cycle {
                            bits = calcNextRequiredDifficulty(t.Unix(), time.Now().Unix(), bits)
                            {
                                    fmt.Printf("begin reset %v: %v\n", time.Now().Sub(t), bits)
                            }
                            cnt = 0
                            t = time.Now()
                    }
            }
    }
    // ci()
}
```

Audit results: After audit, no related risks were found.

Security advice: None.

4.1.2. Cryptography algorithm implementation/use【SAFE】

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

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

Risk file:

Audit process: After testing, the public chain code uses crypto/sha256, crypto/sha3, crypto/ed25519, and crypto/ecdsa libraries when core data encryption operations are involved, which complies with secure encoding operations.

Audit results: After audit, no related risks were found.

Security advice: None.

4.2. RPC

4.2.1. Sensitive interface permissions【SAFE】

Audit Note: Check access to the RPC sensitive interface, whether it is exposed, and test whether the RPC interface has access to sensitive operations or data.

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

Risk file:

Audit process: after detection, the public chain code involves the core data operation has the relevant permissions to verify, in line with the security coding operation.

```
func (g *Greeter) RunGrpc() {
	lis, err := net.Listen("tcp", g.Cfg.SververCfg.GRpcAddress)
	if err != nil {
		logger.Error("net.Listen", zap.Error(err))
		os.Exit(-1)
	}

	server := grpc.NewServer(grpc.UnaryInterceptor(server.IpInterceptor))

	message.RegisterGreeterServer(server, g)

	if err := server.Serve(lis); err != nil {
		panic(err)
	}
}
```

```
func (g *Greeter) GetBalance(ctx context.Context, in *message.ReqBalance) (*message.ResBalance, 
error) {
	addr, err := address.StringToAddress(in.Address)
	if err != nil {
		logger.Info("StringToAddress error", zap.Error(err))
		return nil, fmt.Errorf("eth addr to kto addr error:%s", err.Error())
	}
	balance, err := g.Bc.GetBalance(addr)
	if err != nil {
		logger.Error("g.Bc.GetBalance", zap.Error(err), zap.String("address", in.Address))
		return nil, err
	}
	return &message.ResBalance{Balance: balance}, nil
}

func (g *Greeter) SendTransaction(ctx context.Context, in *message.ReqTransaction) 
(*message.ResTransaction, error) {

	from, err := address.NewAddrFromString(in.From)
	if err != nil {
		return nil, err
	}

	to, err := address.NewAddrFromString(in.To)
	if err != nil {
		return nil, err
	}

	balance, err := g.Bc.GetAvailableBalance(from)
	if err != nil {
		return nil, err
	}
	if in.Amount+in.GasLimit*in.GasPrice > balance {
		return nil, fmt.Errorf("from(%v) balance(%v) is not enough or out of gas.", from, balance)
	}
	sign, err := crypto.DeserializeSignature(in.Sign)
	if err != nil {
		return nil, err
	}

	if in.GasLimit*in.GasPrice == 0 {
		return nil, fmt.Errorf("error: one of gasprice[%v] and gaslimit[%v] is 0", in.GasLimit,
in.GasPrice)
	}

	tx := &transaction.SignedTransaction{
		Transaction: transaction.Transaction{
			From:      from,
			To:        to,
			Amount:    in.Amount,
			Nonce:     in.Nonce,
			GasLimit:  in.GasLimit,
			GasPrice:  in.GasPrice,
			GasFeeCap: in.GasFeeCap,
			Input:     in.Input,
			Type:      transaction.TransferTransaction,
		},
		Signature: *sign,
	}

	err = g.Tp.Add(tx)
	if err != nil {
		return nil, err
	}
	data, err := tx.Serialize()
	if err != nil {
		return nil, err
	}

	if g.Node != nil {
		g.Node.SendMessage(p2p.PayloadMessageType, append([]byte{0}, data...))
	}

	hash := hex.EncodeToString(tx.Hash())

	return &message.ResTransaction{Hash: hash}, nil
}

func (g *Greeter) SendLockTransaction(ctx context.Context, in *message.ReqTransaction)
(*message.ResTransaction, error) {

	from, err := address.NewAddrFromString(in.From)
	if err != nil {
		return nil, err
	}

	to, err := address.NewAddrFromString(in.To)
	if err != nil {
		return nil, err
	}
	balance, err := g.Bc.GetAvailableBalance(from)
	if err != nil {
		return nil, err
	}
	if in.Amount+in.GasLimit*in.GasPrice > balance {
		return nil, errors.New("freeze balance is not enough or out of gas.")
	}
	sign, err := crypto.DeserializeSignature(in.Sign)
	if err != nil {
		return nil, err
	}

	if in.GasLimit*in.GasPrice == 0 {
		return nil, fmt.Errorf("error: one of gasprice[%v] and gaslimit[%v] is 0", in.GasLimit,
in.GasPrice)
	}

	tx := &transaction.SignedTransaction{
		Transaction: transaction.Transaction{
			From:      from,
			To:        to,
			Amount:    in.Amount,
			Nonce:     in.Nonce,
			GasLimit:  in.GasLimit,
			GasPrice:  in.GasPrice,
			GasFeeCap: in.GasFeeCap,
			Input:     in.Input,
			Type:      transaction.LockTransaction,
		},
		Signature: *sign,
	}

	err = g.Tp.Add(tx)
	if err != nil {
		return nil, err
	}
	hash := hex.EncodeToString(tx.Hash())

	return &message.ResTransaction{Hash: hash}, nil
```

```
}

func (g *Greeter) SendUnlockTransaction(ctx context.Context, in *message.ReqTransaction)
(*message.ResTransaction, error) {
        from, err := address.NewAddrFromString(in.From)
        if err != nil {
                return nil, err
        }

        to, err := address.NewAddrFromString(in.To)
        if err != nil {
                return nil, err
        }
        balance, err := g.Bc.GetAllFreezeBalance(from)
        if err != nil {
                return nil, err
        }

        if in.Amount+in.GasLimit*in.GasPrice > balance {
                return nil, errors.New("freeze balance is not enough or out of gas.")
        }

        sign, err := crypto.DeserializeSignature(in.Sign)
        if err != nil {
                return nil, err
        }

        if in.GasLimit*in.GasPrice == 0 {
                return nil, fmt.Errorf("error: one of gasprice[%v] and gaslimit[%v] is 0", in.GasLimit,
in.GasPrice)
        }

        tx := &transaction.SignedTransaction{
```

```
		Transaction: transaction.Transaction{
			From:         from,
			To:           to,
			Amount:       in.Amount,
			Nonce:        in.Nonce,
			GasLimit:     in.GasLimit,
			GasPrice:     in.GasPrice,
			GasFeeCap: in.GasFeeCap,
			Input:        in.Input,
			Type:         transaction.UnlockTransaction,
		},
		Signature: *sign,
	}

	err = g.Tp.Add(tx)
	if err != nil {
		return nil, err
	}
	hash := hex.EncodeToString(tx.Hash())

	return &message.ResTransaction{Hash: hash}, nil
}

func (g *Greeter) GetBlockByNum(ctx context.Context, in *message.ReqBlockByNumber)
(*message.RespBlock, error) {
	b, err := g.Bc.GetBlockByHeight(in.Height)
	if err != nil {
		logger.Error("GetBlockByHeight", zap.Error(fmt.Errorf("error:%v,height %v", err.Error(),
in.Height)))
		return &message.RespBlock{Data: []byte{}, Code: -1, Message: err.Error()}, nil
	}
	blockbyte, err := b.Serialize()
	if err != nil {
		logger.Error("Serialize", zap.String("error", err.Error()))
		return &message.RespBlock{Data: []byte{}, Code: -1, Message: err.Error()}, nil
	}
	return &message.RespBlock{Data: blockbyte, Code: 0}, nil


}

func (g *Greeter) GetBlockByHash(ctx context.Context, in *message.ReqBlockByHash)
(*message.RespBlockDate, error) {
	hash, err := transaction.StringToHash(blockchain.Check0x(in.Hash))
	if err != nil {
		logger.Error("StringToHash", zap.String("error", err.Error()), zap.String("hash", in.Hash))
		return &message.RespBlockDate{Data: []byte{}, Code: -1, Message: err.Error()}, nil
	}
	block, err := g.Bc.GetBlockByHash(hash)
	if err != nil {
		//logger.Error("GetBlockByHash", zap.Error(fmt.Errorf("error:%v,hash: %v", err.Error(),
in.Hash)))
		return &message.RespBlockDate{Data: []byte{}, Code: -1, Message: err.Error()}, nil
	}
	blockbyte, err := block.Serialize()
	if err != nil {
		logger.Error("Serialize", zap.String("error", err.Error()))
		return &message.RespBlockDate{Data: []byte{}, Code: -1, Message: err.Error()}, nil
	}
	return &message.RespBlockDate{Data: blockbyte, Code: 0}, nil
}

func (g *Greeter) GetTxByHash(ctx context.Context, in *message.ReqTxByHash)
(*message.RespTxByHash, error) {
	hash, _ := transaction.StringToHash(blockchain.Check0x(in.Hash))
	tx, err := g.Bc.GetTransactionByHash(hash)
	if err != nil {
		//logger.Error("GetTransactionByHash", zap.Error(fmt.Errorf("error:%v,hash:%v", 
err.Error(), in.Hash)))
		return &message.RespTxByHash{Data: []byte{}, Code: -1, Message: err.Error()}, nil
	}

	txbytes, err := tx.Serialize()
	if err != nil {
		logger.Error("tx Serialize", zap.String("error", err.Error()))
		return &message.RespTxByHash{Data: []byte{}, Code: -1, Message: err.Error()}, nil
	}

	return &message.RespTxByHash{Data: txbytes, Code: 0}, nil
}

func (g *Greeter) GetAddressNonceAt(ctx context.Context, in *message.ReqNonce) 
(*message.ResposeNonce, error) {
	addr, err := address.StringToAddress(in.Address)
	if err != nil {
		logger.Info("StringToAddress error", zap.Error(err))
		return nil, fmt.Errorf("eth addr to kto addr error:%s", err.Error())
	}
	resp, err := g.Bc.GetNonce(addr)
	if err != nil {
		return nil, err
	}

	return &message.ResposeNonce{Nonce: resp}, nil
}

//send eth signed transaction
func (g *Greeter) SendEthSignedRawTransaction(ctx context.Context, in 
*message.ReqEthSignTransaction) (*message.ResEthSignTransaction, error) {
	ethFrom := common.HexToAddress(in.EthFrom)
	ethData := in.EthData

	l := len(ethData)
	if l <= 0 {
		logger.Error("Wrong eth signed data", zap.Error(fmt.Errorf("ethData length[%v] <= 0",
len(ethData))))
		return nil, fmt.Errorf("Wrong eth signed data:%s", fmt.Errorf("ethData length[%v] <= 0",
len(ethData)).Error())
	}
	logger.Info("rpc Into SendEthSignedRawTransaction:", zap.String("from", in.EthFrom),
zap.Int32("data lenght", int32(l)))
	var from, to address.Address
	var evm transaction.EvmContract
	var amount, gasPrice, gasLimit uint64
	var tag transaction.TransactionType
	var signType crypto.SigType

	if len(in.KtoFrom) > 0 && len(in.MsgHash) > 0 {
		f, err := address.NewAddrFromString(in.KtoFrom)
		if err != nil {
			logger.Error("NewAddrFromString error", zap.Error(err))
			return nil, err
		}
		from = f
		signType = crypto.TypeED25519
		evm.MsgHash = in.MsgHash
	} else {
		addr, err := address.StringToAddress(in.EthFrom)
		if err != nil {
			logger.Info("StringToAddress error", zap.Error(err))
			return nil, fmt.Errorf("eth addr to kto addr error:%s", err.Error())
		}
		from = addr
```

```
        signType = crypto.TypeSecp256k1
}

ethTx, err := transaction.DecodeEthData(ethData)
if err != nil {
        logger.Info("SendEthSignedTransaction decodeData error", zap.Error(err))
        return nil, fmt.Errorf("decodeData error:%s", err.Error())
}
evm.EthData = ethData

if len(ethTx.Data()) > 0 {
        to = address.ZeroAddress
        amount = 0
        if ethTx.To() == nil {
                evm.Origin = ethFrom
                evm.CreateCode = ethTx.Data()
                evm.Operation = blockchain.CREATECONTRACT
        } else {
                evm.ContractAddr = *ethTx.To()
                evm.CallInput = ethTx.Data()
                evm.Operation = blockchain.CALLCONTRACT
                evm.Origin = ethFrom
        }
        tag = transaction.EvmContractTransaction
} else {
        ethTo := *ethTx.To()
        addr, err := address.StringToAddress(ethTo.Hex())
        if err != nil {
                logger.Info("StringToAddress error", zap.Error(err))
                return nil, fmt.Errorf("eth addr to kto addr error:%s", err.Error())
        }
        to = addr
```

```
		deci := new(big.Int).SetUint64(blockchain.ETHDECIMAL)
		value := ethTx.Value().Div(ethTx.Value(), deci)
		amount = value.Uint64()
		tag = transaction.EvmKtoTransaction
		evm.Status = true
	}

	if ethTx.GasPrice().Uint64() == 0 {
		gasPrice = g.Cfg.ChainCfg.GasPrice
	} else {
		gasPrice = ethTx.GasPrice().Uint64()
	}

	if ethTx.Gas() == 0 {
		gasLimit = g.Cfg.ChainCfg.GasLimit
	} else {
		gasLimit = ethTx.Gas()
	}

	input, err := transaction.EncodeEvmData(&evm)
	if err != nil {
		logger.Info("EncodeEvmData error", zap.Error(err))
		return nil, fmt.Errorf("EncodeEvmData error:%s", err.Error())
	}

	nonce, err := g.Bc.GetNonce(from)
	if err != nil {
		logger.Info("GetNonce error", zap.Error(err), zap.String("address", from.String()))
		return nil, fmt.Errorf("GetNonce error:%s,from:%v", err.Error(), from)
	}
	if nonce != ethTx.Nonce() {
		return nil, fmt.Errorf("error: from [%v] nonce[%v] not equal ethTx.Nonce[%v]", from,
nonce, ethTx.Nonce())
```

```
	}
	tx := &transaction.SignedTransaction{
		Transaction: transaction.Transaction{
			From:       from,
			To:         to,
			Amount:     amount,
			Nonce:      ethTx.Nonce(),
			GasLimit:   gasLimit,
			GasPrice:   gasPrice,
			GasFeeCap: gasLimit,
			Type:       tag,
			Input:      input,
		},

		Signature: crypto.Signature{
			SigType: signType,
			Data:    transaction.ParseEthSignature(&ethTx),
		},
	}
	err = g.Tp.Add(tx)
	if err != nil {
		logger.Info("EthSignedRawTransaction add tx pool error:", zap.Error(err))
		return nil, fmt.Errorf(err.Error())
	}
	//g.n.Broadcast(tx)

	logger.Info("rpc End SendEthSignedRawTransaction:", zap.String("hash", tx.HashToString()))
	return &message.ResEthSignTransaction{Hash: tx.HashToString()}, nil
}

//send pledge transaction
func (g *Greeter) SendSignedPledgeTransaction(ctx context.Context, in
*message.ReqPledgeTransaction) (*message.ResPledgeTransaction, error) {
	from, err := address.StringToAddress(in.From)
	if err != nil {
		return nil, err
	}

	avib, err := g.Bc.GetAvailableBalance(from)
	if err != nil {
		return nil, err
	}

	if avib < in.Amount+in.GasLimit*in.GasPrice {
		return nil, fmt.Errorf("address[%v] no enough balance to pledge or out of
gas,available[%v],amount[%v],gas[%v]", from, avib, in.Amount, in.GasLimit*in.GasPrice)
	}

	sign, err := crypto.DeserializeSignature(in.Signature)
	if err != nil {
		return nil, err
	}

	var txtype uint8 = 0
	if in.Type == "pledge" {
		if in.Amount <= 0 {
			return nil, fmt.Errorf("pledge amount couldn't <= 0,amount[%v]", in.Amount)
		}
		txtype = transaction.PledgeTrasnaction
	} else if in.Type == "break" {
		if in.Amount != 0 {
			return nil, fmt.Errorf("pledge amount should be 0,amount[%v]", in.Amount)
		}
		totP, err := g.Bc.GetTotalPledge(from)
		if err != nil {
			return nil, err
		}
		if totP == 0 {
			return nil, fmt.Errorf("total pledge[%v] is 0", totP)
		}
		txtype = transaction.PledgeBreakTransaction
	} else {
		return nil, fmt.Errorf("unsupport pledge type[%v]", in.Type)
	}
	tx := &transaction.SignedTransaction{
		Transaction: transaction.Transaction{
			From:       from,
			To:         address.ZeroAddress,
			Amount:     in.Amount,
			Nonce:      in.Nonce,
			GasLimit:   in.GasLimit,
			GasPrice:   in.GasPrice,
			GasFeeCap: in.GasFeeCap,
			Type:       txtype,
		},
		Signature: *sign,
	}

	if err := g.Tp.Add(tx); err != nil {
		return nil, err
	}

	//g.n.Broadcast(tx)
	logger.Info("rpc End SendSignedPledgeTransaction:", zap.String("hash", tx.HashToString()))
	return &message.ResPledgeTransaction{Hash: tx.HashToString()}, nil
}

//get eth address by kto address
```

```
func (g *Greeter) GetETHAddress(ctx context.Context, in *message.ReqKtoAddress)
(*message.ResEthAddress, error) {
	kaddr, err := address.NewAddrFromString(in.Ktoaddress)
	if err != nil {
		logger.Error("rpc NewAddrFromString", zap.String("error", err.Error()))
		return &message.ResEthAddress{Message: err.Error(), Code: -1}, nil
	}
	res, err := g.Bc.GetBindingEthAddress(kaddr)
	if err != nil {
		//logger.Error("rpc GetBindingEthAddress", zap.String("error", err.Error()))
		return &message.ResEthAddress{Message: err.Error(), Code: -1}, nil
	}
	return &message.ResEthAddress{Ethaddress: res, Code: 0}, nil
}

//get kto address    by eth address
func (g *Greeter) GetKTOAddress(ctx context.Context, in *message.ReqEthAddress)
(*message.ResKtoAddress, error) {
	res, err := g.Bc.GetBindingKtoAddress(in.Ethaddress)
	if err != nil {
		//logger.Error("rpc GetBindingKtoAddress", zap.String("error", err.Error()))
		return &message.ResKtoAddress{Message: err.Error(), Code: -1}, nil
	}
	return &message.ResKtoAddress{Ktoaddress: res.String(), Code: 0}, nil
}

//call contract
func (g *Greeter) CallSmartContract(ctx context.Context, in *message.ReqCallContract)
(*message.ResCallContract, error) {
	res, gas, err := g.Bc.CallSmartContract(in.Contractaddress, in.Origin, in.Inputcode, in.Value)
	if err != nil {
		logger.Error("rpc CallSmartContract", zap.String("Result", res), zap.String("error",
err.Error()))
```

```
		return &message.ResCallContract{Result: res, Msg: err.Error(), Code: -1}, nil
	}
	return &message.ResCallContract{Result: res, Gas: gas, Code: 0}, nil
}

//get code by contract address
func (g *Greeter) GetCode(ctx context.Context, in *message.ReqEvmGetcode)
(*message.ResEvmGetcode, error) {
	code := g.Bc.GetCode(in.Contract)
	if len(code) <= 0 {
		return nil, fmt.Errorf("code not exist")
	}
	return &message.ResEvmGetcode{Code: common.Bytes2Hex(code)}, nil
}

//get storage by hash
func (g *Greeter) GetStorageAt(ctx context.Context, in *message.ReqGetstorage)
(*message.ResGetstorage, error) {
	ret := g.Bc.GetStorageAt(in.Addr, blockchain.Check0x(in.Hash))
	return &message.ResGetstorage{Result: ret.String()}, nil
}

//get max block height
func (g *Greeter) GetMaxBlockHeight(ctx context.Context, in *message.ReqMaxBlockHeight)
(*message.ResMaxBlockHeight, error) {
	maxH, err := g.Bc.GetMaxBlockHeight()
	if err != nil {
		logger.Error("rpc GetMaxBlockHeight", zap.String("error", err.Error()))
		return nil, err
	}
	return &message.ResMaxBlockHeight{MaxHeight: maxH}, nil
}
```

```
//get evm logs
func (g *Greeter) GetLogs(ctx context.Context, in *message.ReqLogs) (*message.ResLogs, error) {
	if in.FromBlock > in.ToBlock {
		return nil, fmt.Errorf("Wrong fromBlock[%v] and to toBlock[%v]", in.FromBlock,
in.ToBlock)
	}
	logs := g.Bc.GetLogs()
	var resLogs []*evmtypes.Log
	for _, log := range logs {
		if common.HexToHash(in.BlockHash) == log.BlockHash {
			resLogs = append(resLogs, log)
		} else if log.BlockNumber >= in.FromBlock && log.BlockNumber <= in.ToBlock {
			if common.HexToAddress(in.Address) == log.Address {
				resLogs = append(resLogs, log)
			}
		}
	}
	bslog, err := json.Marshal(&resLogs)
	if err != nil {
		return nil, err
	}
	return &message.ResLogs{Evmlogs: bslog}, nil
}

//get total transaction pledge and mined pledge by address
func (g *Greeter) GetTotalPledge(ctx context.Context, in *message.ReqKtoAddress)
(*message.ResPledge, error) {
	addr, err := address.StringToAddress(in.Ktoaddress)
	if err != nil {
		logger.Info("GetTotalPledge string address to Address error", zap.Error(err))
		return &message.ResPledge{Code: -1, Message: err.Error()}, nil
	}
```

```
    totalP, err := g.Bc.GetTotalPledge(addr)
    if err != nil {
        logger.Info("GetTotalPledge error", zap.Error(err))
        return &message.ResPledge{Code: -1, Message: err.Error()}, nil
    }

    totalM, err := g.Bc.GetTotalMined(addr)
    if err != nil {
        logger.Info("GetTotalMined error", zap.Error(err))
        return &message.ResPledge{Code: -1, Message: err.Error()}, nil
    }
    return &message.ResPledge{TotalPledge: totalP, TotalMined: totalM, Code: 0}, nil
}

func (g *Greeter) CreateToken(ctx context.Context, in *message.ReqTokenCreate)
(*message.HashMsg, error) {

    from, err := address.NewAddrFromString(in.From)
    if err != nil {
        return nil, err
    }

    to, err := address.NewAddrFromString(in.To)
    if err != nil {
        return nil, err
    }

    balance, err := g.Bc.GetAvailableBalance(from)
    if err != nil {
        return nil, err
    }
    if in.GasLimit*in.GasPrice > balance {
        return nil, errors.New("freeze balance is not enough or out of gas.")
```

```
	}
	sign, err := crypto.DeserializeSignature(in.Signature)
	if err != nil {
		return nil, err
	}

	if in.GasLimit*in.GasPrice == 0 {
		return nil, fmt.Errorf("error: one of gasprice[%v] and gaslimit[%v] is 0", in.GasLimit,
in.GasPrice)
	}

	tx := &transaction.SignedTransaction{
		Transaction: transaction.Transaction{
			From:       from,
			To:         to,
			Nonce:      in.Nonce,
			GasLimit:   in.GasLimit,
			GasPrice:   in.GasPrice,
			GasFeeCap: in.GasFeeCap,
			Type:       transaction.TransferTransaction,
			Input:      in.Input,
		},
		Signature: *sign,
	}

	script := string(tx.Input)
	_, err = g.Bc.GetTokenRoot(from, script)
	if err != nil {
		return nil, errors.New("get token root err")
	}

	err = g.Tp.Add(tx)
	if err != nil {
		return nil, err
	}

	data, err := tx.Serialize()
	if err != nil {
		return nil, err
	}

	g.Node.SendMessage(p2p.PayloadMessageType, append([]byte{0}, data...))
	hash := hex.EncodeToString(tx.Hash())

	return &message.HashMsg{Hash: hash}, nil

}
func (g *Greeter) MintToken(ctx context.Context, in *message.ReqTokenCreate) (*message.HashMsg,
error) {

	from, err := address.NewAddrFromString(in.From)
	if err != nil {
		return nil, err
	}

	to, err := address.NewAddrFromString(in.To)
	if err != nil {
		return nil, err
	}

	if len(in.Symbol) == 0 {
		logger.Info("symbol", zap.String("symbol", in.Symbol))
		return &message.HashMsg{Code: -1, Message: "invalid symbol"}, nil
	}

	sign, err := crypto.DeserializeSignature(in.Signature)
```

```
if err != nil {
        return nil, err
}

tx := &transaction.SignedTransaction{
        Transaction: transaction.Transaction{
                From:          from,
                To:            to,
                Nonce:         in.Nonce,
                GasLimit:   in.GasLimit,
                GasPrice:   in.GasPrice,
                GasFeeCap: in.GasFeeCap,
                Type:          transaction.TransferTransaction,
                Input:        in.Input,
        },
        Signature: *sign,
}

script := string(tx.Input)
_, err = g.Bc.GetTokenRoot(from, script)
if err != nil {
        return nil, errors.New("get token root err")
}

err = g.Tp.Add(tx)
if err != nil {
        return nil, err
}

data, err := tx.Serialize()
if err != nil {
        return nil, err
}
```

```
        g.Node.SendMessage(p2p.PayloadMessageType, append([]byte{0}, data...))
        hash := hex.EncodeToString(tx.Hash())

        return &message.HashMsg{Hash: hash}, nil

}
func (g *Greeter) SendToken(ctx context.Context, in *message.ReqTokenTransaction)
(*message.RespTokenTransaction, error) {
        from, err := address.NewAddrFromString(in.From)
        if err != nil {
                return nil, err
        }

        to, err := address.NewAddrFromString(in.To)
        if err != nil {
                return nil, err
        }
        if len(in.Input) == 0 {
                logger.Info("symbol", zap.String("symbol", string(in.Input)))
                return nil, fmt.Errorf("symbol:%s", in.Input)
        }

        sign, err := crypto.DeserializeSignature(in.Signature)
        if err != nil {
                return nil, err
        }

        if in.GasLimit*in.GasPrice == 0 {
                return nil, fmt.Errorf("error: one of gasprice[%v] and gaslimit[%v] is 0", in.GasLimit,
in.GasPrice)
        }
```

```
tx := &transaction.SignedTransaction{
        Transaction: transaction.Transaction{
                From:           from,
                To:             to,
                Nonce:          in.Nonce,
                GasLimit:    in.GasLimit,
                GasPrice:    in.GasPrice,
                GasFeeCap: in.GasFeeCap,
                Type:           transaction.TransferTransaction,
                Input:         in.Input,
        },
        Signature: *sign,
}

err = g.Tp.Add(tx)
if err != nil {
        return nil, err
}

script := string(in.Input)
_, err = g.Bc.GetTokenRoot(from, script)
if err != nil {
        logger.Error("Failed to get token root", zap.String("from", from.String()),
                zap.String("script", string(in.Input)))
        return nil, grpc.Errorf(codes.InvalidArgument, "get root failed")
}

data, err := tx.Serialize()
if err != nil {
        return nil, err
}

g.Node.SendMessage(p2p.PayloadMessageType, append([]byte{0}, data...))
```

```
	hash := hex.EncodeToString(tx.Hash())


	return &message.RespTokenTransaction{Hash: hash}, nil


}
func (g *Greeter) GetBalanceToken(ctx context.Context, in *message.ReqTokenBalance)
(*message.RespTokenBalance, error) {
	addr, err := address.NewAddrFromString(in.Address)
	if err != nil {
		return nil, err
	}


	balance, err := g.Bc.GetTokenBalance(addr, []byte(in.Symbol))
	if err != nil {
		logger.Error("g.Bc.GetTokenBalance", zap.Error(err), zap.String("address", in.Address),
zap.String("symbol", in.Symbol))
		return nil, grpc.Errorf(codes.InvalidArgument, "symbol:\"%s\",address:%s", in.Symbol,
in.Address)
	}


	demic, err := g.Bc.GetTokenDemic([]byte(in.Symbol))
	if err != nil {
		logger.Error("get demic:", zap.String("symbol", in.Symbol), zap.Error(err))
		return nil, grpc.Errorf(codes.InvalidArgument, "symbol:\"%s\",address:%s", in.Symbol,
in.Address)
	}


	return &message.RespTokenBalance{Demic: demic, Balnce: balance}, nil
}


func (g *Greeter) GetSingleFreezeBalance(ctx context.Context, in *message.ReqSignBalance)
(*message.ResBalance, error) {
	from, err := address.NewAddrFromString(in.From)
	if err != nil {
		return nil, err
	}
	to, err := address.NewAddrFromString(in.To)
	if err != nil {
		return nil, err
	}

	bal, err := g.Bc.GetSingleFreezeBalance(from, to)
	if err != nil {
		return nil, err
	}
	return &message.ResBalance{Balance: bal}, nil
}

func (g *Greeter) GetAllFreezeBalance(ctx context.Context, in *message.ReqBalance)
(*message.ResBalance, error) {
	from, err := address.NewAddrFromString(in.Address)
	if err != nil {
		return nil, err
	}

	bal, err := g.Bc.GetAllFreezeBalance(from)
	if err != nil {
		return nil, err
	}
	return &message.ResBalance{Balance: bal}, nil
}

func (g *Greeter) GetWholeNetworkPledge(ctx context.Context, in
*message.ReqWholeNetworkPledge) (*message.ResWholeNetworkPledge, error) {
	total, err := g.Bc.GetWholeNetWorkTotalPledge()
	if err != nil {
		return nil, err
	}
	return &message.ResWholeNetworkPledge{WholeNetworkPledge: total}, nil
}

func (g *Greeter) GetAvailableBalance(ctx context.Context, in *message.ReqGetAvailableBalance)
(*message.ResGetAvailableBalance, error) {
	addr, err := address.StringToAddress(in.Address)
	if err != nil {
		logger.Info("StringToAddress error", zap.Error(err))
		return nil, err
	}
	avi, err := g.Bc.GetAvailableBalance(addr)
	if err != nil {
		return nil, err
	}
	return &message.ResGetAvailableBalance{AvailableBalance: avi}, nil
}

func (g *Greeter) GetHasherPerSecond(ctx context.Context, req *message.ReqHasherPerSecond)
(*message.ResHasherPerSecond, error) {
	resp := &message.ResHasherPerSecond{}
	if g.Miner == nil {
		resp.HasherPerSecond = 0
		return resp, nil
	}
	h := g.Miner.HashesPerSecond()
	resp.Address = h.MinerAddr.String()
	resp.HasherPerSecond = float32(h.HasherPerSecond)
	resp.Uuid = g.NodeName
	return resp, nil
}
```

Audit results: After audit, no related risks were found.

Security advice: None.

4.2.2. Business logic【SAFE】

Audit Notes: Check access to RPC interfaces, export wallets, back up wallets and other functions for overstepping authority, logical errors, etc.

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

Risk file:

Audit process: After testing, the public chain code involves the core operation use authentication and related processing logic is correct, in line with the security coding operation.

Audit results: After audit, no related risks were found.

Security advice: None.

4.2.3. Traditional Web security【SAFE】

Audit Notes: Check/test traditional web security items such as XXE injection, CSRF, SSRF, path crossing, and sensitive information clear text transfer.

Risk hazards: remote command execution, exposure of RPC interfaces, CSRF, SSRF attacks, arbitrary file reads, sensitive information leakage, etc.

Risk file:

Audit process: It has been detected that there is no common web attack risk such as XXE injection in public chain code.

Audit results: After audit, no related risks were found.

Security advice: None.

4.3. Key security

4.3.1. Private key/mnemonic generation algorithm【SAFE】

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

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

Risk file:

Audit process: After testing, no mnemonic words are involved in the public chain code. The private key is generated by the ed25519 or secp256k1 algorithm.

Audit results: After audit, no related risks were found.

```
func Generate(sigType crypto.SigType) ([]byte, error) { //knownsec// Generate private key      s, ok :=
sigs[sigType]
        if !ok {
                return nil, fmt.Errorf("cannot generate private key with signature of unsupported type:%v",
sigType)
        }


        return s.Generate()
}
```

Security advice: None.

4.3.2. Private key/mnemonic clear text storage【SAFE】

Audit Note: Check how the files are stored, whether they are encrypted, or can be decrypted

Risk hazards: Working with other vulnerabilities can lead to private key disclosure, affect asset security, serious can lead to chain fork and other issues.

Risk file:

Audit process: Check that the private key/mnemonics are stored in clear text.

Audit results: After auditing, the public chain code does not involve the storage of private keys/mnemonics for the time being.

Security advice: None.

4.3.3. Private key/mnemonics use traces【SAFE】

Audit Notes: Check whether the private key/mnemonics in the code logic clean up traces in time after use, check whether sensitive data can be used, can be found from logs, memory dump traces can leak information traces.

Risk hazards: Working with other vulnerabilities can lead to private key disclosure, affect asset security, serious can lead to chain fork and other issues.

Risk file:

Audit process: After testing, the use of private keys/mnemonics in public chain code conforms to the safety practices and no associated risks are found.

Audit results: After audit, no related risks were found.

Security advice: None.

4.3.4. Private key/mnemonic residue【SAFE】

Audit Note: Check that the private key/mnemonic deletion is thorough.

Risk hazards: Working with other vulnerabilities can lead to private key disclosure, affect asset security, serious can lead to chain fork and other issues.

Risk file:

Audit process: After testing, the use of private keys/mnemonics in public chain code conforms to the safety practices and no associated risks are found.

Audit results: After audit, no related risks were found.

Security advice: None.

4.3.5. Private key/mnemonic abuse【SAFE】

Audit Notes: Check that the private key/mnemonic is being used in a non-essential location.

Risk Hazards: Increase the risk of private key leakage.

Risk file:

Audit process: After testing, the use of private keys/mnemonics in public chain code conforms to the safety practices and no associated risks are found.

Audit results: After audit, no related risks were found.

Security advice: None.

4.4. Consensus mechanisms

4.4.1. Consensus validation implementation【SAFE】

Audit Note: Check that consensus such as pow/poc can construct legal blocks at less than expected cost.

Risk hazards: calculation force forgery, calculation force centralization, serious can lead to 51% attack, chain fork and other issues.

Risk file:

Audit process: Audit the consensus verification implementation logic of the public chain.

Audit results: After testing, the KORTHO public chain consensus verification mechanism adopts a PoW+PoS hybrid consensus. The consensus still requires the nodes participating in the block production to perform a certain amount of hash value calculation, and the block is generated in a similar way to proof of work, but each node calculates The probability of a legal block is related to the stake held by the node. No related security risks were found.

```
func NextMinerDifficulty(baseDifficulty *big.Int, minerPledge, basePledge uint64) (*big.Int, error)
{ //knownsec// Next miner difficulty
    target, baseTargetCopy := big.NewInt(0), big.NewInt(0)

    minerTarget := big.NewInt(0)
    target.Add(target, baseDifficulty)
    baseTargetCopy.Add(baseTargetCopy, baseDifficulty)

    if minerPledge > basePledge {
        minerPledge = basePledge
```

```
	}


	a := big.NewInt(0).SetUint64((minerPledge) * 900)
	a = a.Div(a, big.NewInt(0).SetUint64(basePledge))


	minerTarget.Add(baseDifficulty, target.Mul(target, a).Div(target, big.NewInt(100)))


	if baseTargetCopy.Cmp(minerTarget) > 0 {
		return nil, fmt.Errorf("miners too difficult,old difficulty:%s new difficulty:%s miner
pledge:%d,basfe pledeg:%d",
			baseTargetCopy, minerTarget, minerPledge, basePledge)
	} else if baseTargetCopy.Mul(baseTargetCopy, big.NewInt(10)).Cmp(minerTarget) < 0 {
		return nil, fmt.Errorf("miners too little difficulty,old difficulty:%s new difficulty:%s miner
pledge:%d,basfe pledeg:%d",
			baseTargetCopy, minerTarget, minerPledge, basePledge)
	}


	return minerTarget, nil
}
```

Security advice: None.

4.4.2. Consensus mechanism design【SAFE】

Audit Note: For the new type of consensus mechanism, check the design for design security risks based on its documentation, such as missing transactions/transactions not verified/falsifying calculation proof/centrization/BP evil.

Risk hazards: Analysed on a case-by-case basis.

Risk file:

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

Audit results: After testing, the KORTHO public chain consensus verification mechanism adopts a PoW+PoS hybrid consensus. The consensus still requires the nodes participating in the block production to perform a certain amount of hash value calculation, and the block is generated in a similar way to proof of work, but each node calculates The probability of a legal block is related to the stake held by the node. No related security risks were found.

Security advice: None.

4.5. P2P

4.5.1. Differentiated responses to sensitive information【SAFE】

Audit Notes: Check the response data logic for each response and test whether there are different responses to sensitive information in each response.

Risk hazards: Disclosure of sensitive information.

Risk file:

Audit process: After testing, the public chain code involves P2P connection response function only to meet the standard msg protocol to answer, in line with the security coding specifications.

Audit results: After audit, no related risks were found.

Security advice: None.

4.5.2. Fuzz deformity data test【SAFE】

Audit Notes: Check the response data logic of the various responses and test whether the nodes in the various responses are responding properly.

Risk hazards: A data attack that is deformed can cause nodes to deny service or disclose chain-related information.

Risk file:

Audit process: After testing, the public chain code involves P2P connection response function only to meet the standard msg protocol to answer, in line with the security coding specifications.

Audit results: After audit, no related risks were found.

Security advice: None.

4.5.3. The node discovery algorithm【SAFE】

Audit Note: Check whether the node discovery algorithm is balanced and unpredictable, such as distance algorithm imbalance and other issues.

Risk hazards: Analysed on a case-by-case basis.

Risk file:

Audit process: Check whether the node discovery algorithm is balanced and unpredictable, such as distance algorithm imbalance and other issues.

Audit results: After audit, no related risks were found.

Security advice: None.

4.5.4. The node communication protocol【SAFE】

Audit Note: Check the inter-node communication protocol for design/ implementation layer problems, fuzzy test the node, test the node's reaction to the deformed packet.

Risk hazards: Analysed on a case-by-case basis.

Risk file:

Audit process: Check the inter-node communication protocol for design/ implementation layer problems, fuzzy test of nodes, test the node's reaction to malformed packets.

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

Security advice: None.

4.5.5. Route table contamination【SAFE】

Audit Note: Check that the routing table can be inserted or overwritten at will.

Risk hazards: Reduce the difficulty of eclipse attacks, etc.

Risk file:

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

Audit results: After audit, no related risks were found.

Security advice: None.

4.5.6. Traffic restrictions【SAFE】

Audit Note: Check the limits on the number of connections, packet size, etc. in p2p protocols in code logic, try to establish a large number of connections with the target node, and construct packets that send various volumes.

Risk hazard: Take up the local storage resource DoS node, which can cause the node program to crash.

Risk file:

Audit process: The size of the requested data is detected and security checked, and the block form of maxNeighbors is sent to the neighboring node for processing to stay below the packet size limit.

```
func (n *Node) handleMessage(msg *message) bool {
	msgc := *msg

	n.messageClock.Witness(msgc.LTime)
	// Check if this message is too old
	currTime := n.messageClock.Time()
	if currTime > LamportTime(len(n.messageBuffer)) && msgc.LTime <
currTime-LamportTime(len(n.messageBuffer)) {
		//n.logger.Printf("node received old message from message:%s,message ltime:%d (cuttent
ltime:%d)",
		//	hex.EncodeToString(msgc.id()), msgc.LTime, currTime)
		return false
	}

	msgid := msgc.id()
	idx := msgc.LTime % LamportTime(len(n.messageBuffer))

```

```
    n.messageBufferMutex.Lock()
    seen := n.messageBuffer[idx]
    if seen != nil && seen.LTime == msgc.LTime {
        for _, id := range seen.IDs {
            if bytes.Equal(msgid, id) {
                n.messageBufferMutex.Unlock()
                return false
            }
        }
    } else {
        seen = &receivedMessage{LTime: msgc.LTime}
        n.messageBuffer[idx] = seen
    }
    seen.IDs = append(seen.IDs, msgid)
    n.messageBufferMutex.Unlock()

    if n.HandleFunc != nil {
        go n.HandleFunc(msgc.Payload)
    }

    return true
}
```

Audit results: After audit, no related risks were found.

Security advice: None.

4.5.7. Node penalty mechanism【SAFE】

Audit Notes: Check that the punishment mechanism for nodes is reasonable.

Risk hazard: Isolated normal nodes.

Risk file:

Audit process: Check that the punishment mechanism for nodes is reasonable.

Audit results: After audit, no related risks were found.

Security advice: None.

4.5.8. Solar eclipse attack【SAFE】

Audit Notes: Assess the cost and hazards of an eclipse attack and provide quantitative analysis if necessary.

Risk hazards: isolation of normal nodes, asset double flowers, serious can lead to chain rollback, chain fork and other issues.

Risk file:

Audit process: Assess the cost and hazards of an eclipse attack and provide quantitative analysis if necessary.

Audit results: After audit, no related risks were found.

Security advice: None.

4.6. Chain processing

4.6.1. Chain synchronization logic【SAFE】

Audit Note: Check the scope of the synchronization chunk, verify the logic, and whether other asynchronous operations during the period are reasonable.

Risk hazards: Chain forks.

Risk file:

Audit process: Check the scope of synchronization blocks, verify logic, and whether other asynchronous operations during the period are reasonable.

Audit results: After testing, the chain synchronization related code in the male chain code conforms to the security coding operation, and no obvious security problems are found.

```
//ProcessBlock is management block function
func (b *BlockChain) ProcessBlock(newblock *block.Block, globalDifficulty *big.Int, basePledge
uint64) bool {//knownsec// Processing block

    //newblcok hash is exist
    defer logger.Info(" ProcessBlock    end ", zap.Uint64("height", newblock.Height),
zap.String("hash", hex.EncodeToString(newblock.Hash)))

    if b.BlockExists(newblock.Hash) {
        fmt.Println("Block is exist", hex.EncodeToString(newblock.Hash))
        return false
    }

    hash := BytesToHash(newblock.Hash)
    if _, exist := b.Oranphs[hash]; exist {
        logger.Info("orphan is exist", zap.String("hash", hex.EncodeToString(newblock.Hash)))
        return false
    }

    h, err := b.Bc.GetMaxBlockHeight()
    if err != nil {
        logger.SugarLogger.Error("GetBlockByHeight err", err)
        return false
    }
    if newblock.Height == 1 && h == 0 {
```

```
		logger.SugarLogger.Error("==========first blcok============",
hex.EncodeToString(newblock.Hash))
		err = b.Bc.AddBlock(newblock)
		if err != nil {
			return false
		}
		return true
	}
	//Check block data sanity
	err = checkBlockRegular(newblock, b.Bc, globalDifficulty, basePledge)
	if err != nil {
		logger.Error("checkBlockRegular err", zap.Error(err))
		return false
	}

	//Determine whether prevhash exists
	if !b.BlockExists(newblock.PrevHash) {
		logger.Info("prevhash not exist")
		b.AddOrphanBlock(newblock)
		return false
	}

	//maybeAcceptBlock return longest chain flag
	succ, mainChain := b.maybeAcceptBlock(newblock)
	if !succ {
		return false
	}
	ok := b.ProcessOrphan(newblock)
	if ok {
		mainChain = ok
	}

	return mainChain
```

```
}
```

Security advice: None.

4.6.2. Longest chain selection/switching【SAFE】

Audit Notes: Check the longest chain selection and switching algorithm and whether the implementation is reasonable.

Risk hazards: chain rollback, chain fork.

Risk file:

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

Audit results: After testing, chain switching and selection logic in the male chain code conforms to the security logic, and no obvious security problems are found.

```
//Determine whether the PrevBlock of the current block is bestChain 的 tip,Returns the result of adding data
// and block whether or not bestchain
func (bc *BlockChain) maybeAcceptBlock(b *block.Block) (bool, bool) {

	tipblock, err := bc.Bc.Tip()
	if err != nil {
		logger.Error("GetMaxBlockHeight err")
		return false, false
	}
	// prevhash := BytesToHash(b.PrevHash)
	// tiphash := BytesToHash(tipblock.Hash)

	//add blcok to bestchain
	if bytes.Equal(b.PrevHash, tipblock.Hash) { //knownsec// Block prevHash is consistent with the
current highest block hash, insert the block normally and return
		//if prevhash.IsEqual(&tiphash) {
		//		logger.SugarLogger.Error("tipHash:%s\npreHash:%s\ntip_height:%d\n",
		//			hex.EncodeToString(tipblock.Hash), hex.EncodeToString(b.PrevHash),
tipblock.Height)
		//The current block and the main chain block need to verify the difficulty
		if b.Height%10 != 2 || b.Height == 2 {
			if tipblock.GlobalDifficulty.Cmp(b.GlobalDifficulty) < 0 {
				logger.SugarLogger.Error("block Difficulty is easy", "myself : ",
tipblock.GlobalDifficulty, "block : ", b.GlobalDifficulty)
				return false, false
			}
		}

		if b.Height != 1 {
			if err := bc.Bc.DifficultDetection(b); err != nil {
				return false, false
			}
		}

		err := bc.Bc.AddBlock(b) //knownsec// Insert block
		if err != nil {
			logger.SugarLogger.Errorf("------------maybeAcceptBlock,hash:%s,height:%d\n",
hex.EncodeToString(b.Hash), b.Height, zap.Error(err))
			return false, false
		}
		return true, true
	}

	parent, err := bc.Bc.GetBlockByHash(b.PrevHash) //knownsec// Get parent block
	if err != nil {
```

```
            logger.Error("AddUncleBlock", zap.String("PrevHash", hex.EncodeToString(b.PrevHash)),
zap.Error(err))
            return false, false
      }


      //
      if parent.Height == tipblock.Height { //knownsec// Fork situation
            err := bc.Bc.AddUncleBlock(b)
            if err != nil {
                  logger.Error("AddUncleBlock", zap.Uint64("height", b.Height), zap.Error(err))
                  return false, false
            }
            //The side chain is converted to the main chain
            //1. Find the bifurcation point
            hashList := make([][]byte, 0)
            hashList = append(hashList, b.Hash)
            branchhash, delHeight, err := bc.FindBranchPoint(parent.Hash, tipblock.Hash) //knownsec//
Find the bifurcation point
            if err != nil {
                  logger.Error("FindBranchPoint", zap.Uint64("height", b.Height), zap.Error(err))
                  return false, false
            }

            if len(branchhash) > 0 {
                  hashList = append(hashList, branchhash...)
            }

            err = bc.Bc.ReorganizeChain(hashList, delHeight) //knownsec// Chain reorganization
            if err != nil {
                  logger.Error("ReorganizeChain", zap.Uint64("height", b.Height), zap.Error(err))
                  bc.Bc.DeleteUncleBlock(b)
                  return false, false
            }
```

```
    } else if parent.Height < tipblock.Height { //knownsec// Orphan block situation
        err := bc.Bc.AddUncleBlock(b)
        if err != nil {
            return false, false
        }
        return true, false

    } else { //knownsec// Error handling in other cases
        logger.Error("blockchain db have heigher blockchain")
        return false, false
    }
    return true, true

}
```

Security advice: None.

4.7. Block processing

4.7.1. Block hash collision【SAFE】

Audit Notes: Check how the block hash collision is constructed and whether the handling at the time of the collision is reasonable.

Risk hazard: Block hash collision.

Risk file:

Audit process: after detection, the male chain code involved in the core data encryption operations are using sha3-256, in line with the security coding operation.

Audit results: After audit, no related risks were found.

```
//SetHash hash the block data
```

```
func (b *Block) SetHash() error {
        data, err := b.Serialize()
        if err != nil {
                return err
        }

        hash := sha3.Sum256(data)
        b.Hash = hash[:]
        return nil
}
```

Security advice: None.

4.7.2. Block generation logic【SAFE】

Audit Note: Check that the logical design of block generation is reasonable.

Risk hazards: block forgery, construction fork, double flower, etc.

Risk file:

Audit process: Check that the logical design of block generation is reasonable.

Audit results: After testing, the block generation logic design in the public chain code is reasonable.

```
// NewBlock create a new block for the blockchain
func (bc *Blockchain) NewBlock(txs []*transaction.SignedTransaction, minaddr address.Address)
(*block.Block, error) { //knownsec// Generate a new block
        //logger.Info("start to new block")
        var height, prevHeight uint64
        var prevHash []byte
        var gasUsed uint64
        prevHeight, err := bc.GetMaxBlockHeight()
```

```
	if err != nil {
		logger.Error("failed to get height", zap.Error(err))
		return nil, err
	}

	height = prevHeight + 1
	if height > 1 { //knownsec// Get the previous block hash
		prevHash, err = bc.GetHash(prevHeight)
		if err != nil {
			logger.Error("failed to get hash", zap.Error(err), zap.Uint64("previous height",
prevHeight))
			return nil, err
		}
	} else { //knownsec// Genesis block
		prevHash = []byte{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0}
	}

	// init test (Test use only)
	//	if height == 1 {
	//		setInitTest(bc)
	//		txs = distrInitTest(txs)
	//		txs = distrInitWaterTest(txs)
	//		txs = distrInitWaterTest1(txs)
	//		txs = distrInitWaterTest2(txs)
	//		txs = distrInitWaterTest3(txs)
	//		txs = distrInitWaterTest4(txs)
	//	}

	// Currency distribution
	txs = distr(txs, minaddr, height) //knownsec// Processing miner coinbase transactions

	// Generate Merkel root, if there is no deal, calling GetMthash will painc
```

```
txBytesList := make([][]byte, 0, len(txs))
for _, tx := range txs {
        serialize, _ := tx.Serialize()
        txBytesList = append(txBytesList, serialize)
        gasUsed += tx.Transaction.GasLimit * tx.Transaction.GasPrice
}
tree := merkle.New(sha256.New(), txBytesList) //knownsec// Generate merkle tree
root := tree.GetMtHash()
//getRoot, err := getSnapRoot(bc.db)
getRoot, err := getSnapRootLock(bc.db)
if err != nil {
        logger.Error("failed to get getSnapRootLock", zap.Error(err))
        return nil, err

}
snapRoot := getRoot.Bytes()

// var difficulty *big.Int
// if prevHeight > 1 {
//      // Gets the data of the previous block
//      //              prevBlock, err := bc.getBlockByHeight(prevHeight)
//      prevBlock, err := bc.GetBlockByHeight(prevHeight)
//      if err != nil {
//              logger.Error("failed to get block", zap.Error(err))
//              return nil, err
//      }
//      difficulty = prevBlock.Difficulty
// } else {
//      difficulty = new(big.Int)
// }

ftxs := make([]*transaction.FinishedTransaction, len(txs)) //knownsec// Package transaction
for i, _ := range ftxs {
        ftxs[i] = &transaction.FinishedTransaction{SignedTransaction: *txs[i]}
        ftxs[i].BlockNum = height
    }

    timestamp := uint64(time.Now().Unix())
    block := &block.Block{ //knownsec// Building block
        Height:             height,
        PrevHash:           prevHash,
        Transactions:       ftxs,
        Root:               root,
        Version:            1,
        Timestamp:          timestamp,
        UsedTime:           0,
        Miner:              minaddr,
        SnapRoot:           snapRoot,
        Difficulty:         big.NewInt(0),
        GlobalDifficulty: big.NewInt(0),
        Nonce:              1,
        GasLimit:           bc.ChainCfg.GasLimit,
        GasUsed:            gasUsed,
    }
    //block.SetHash()
    //      logger.Info("end to new block")
    return block, nil
}
```

Security advice: None.

4.7.3. Block validation logic 【SAFE】

Audit Notes: Check that the block verification logic is adequate.

Risk hazards: forged blocks, resulting in unexpected gains, serious can lead to chain forking, data tampering and other issues.

Risk file:

Audit process: After testing, the block processing logic in the public chain code did not find obvious security problems.

Audit results: After audit, no related risks were found.

```
func checkBlockRegular(b *block.Block, bc blockchain.Blockchains, globalDifficulty *big.Int,
basePledge uint64) error { //knownsec// Block regular inspection
	if len(b.Transactions) > txpool.PendingLimit+1 {
		return errors.New("transnations out of PendingLimit")
	}

	// if err := bc.DifficultDetection(globalDifficulty, basePledge, b); err != nil {
	//	return err
	// }

	//checkNonceMp := make(map[address.Address]uint64)
	maxNum := 0
	for _, tx := range b.Transactions {
		switch tx.Type {
		case transaction.CoinBaseTransaction:
			if maxNum > 0 {
				return fmt.Errorf("a block should have only one Coin Base Transaction")
			}
			amount := blockchain.GetMinerAmount(tx.BlockNum)
			if amount != tx.Amount {
				return fmt.Errorf("error: transaction wrong miner amount[%v],expect[%v]",
tx.Amount, amount)
			}
			maxNum++
		case transaction.TransferTransaction, transaction.PledgeTrasnaction,
transaction.PledgeBreakTransaction:

		case transaction.EvmContractTransaction, transaction.EvmKtoTransaction:

		case transaction.LockTransaction, transaction.UnlockTransaction:

		default:
			return errors.New("invalid transaction type")
		}
	}
	return nil
}
```

Security advice: None.

4.7.4. Block processing resource restrictions【SAFE】

Audit Note: Check whether resource limitations such as orphan block pools, verification calculations, hard disk addressing, etc. are reasonable.

Risk hazard: Take up local storage, CPU resources DoS nodes, severe can cause node programs to crash.

Risk file:

Audit process: Check that resource limitations such as orphan block pools, verification calculations, hard disk addressing, etc. are reasonable.

Audit results: After audit, no related risks were found.

Security advice: None.

4.7.5. Block signature verification logic【SAFE】

Audit Note: Check whether the block signature verification logic is sufficient.

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

Risk file:

Audit process: Audit and check the block signature verification logic in the public chain.

Audit results: After testing, the Verify method in the Block data structure in the public chain code has no specific implementation. The block check processing is implemented in the checkblock.go file, and the difficulty of the block, gas fee, etc. will be verified.

```
func (bc *Blockchain) CheckBlockRegular(b *block.Block) error {//knownsec// Block check
	tx := bc.NewTransaction()
	defer tx.Cancel()
	return bc.checkBlockRegular(b, bc.db, tx)
}

func (bc *Blockchain) checkBlockRegular(b *block.Block, db store.DB, tx store.Transaction) error {
	if len(b.Transactions) > 101 {
		return errors.New("Too many transactions")
	}

	// checkout Difficulty
	if err := difficultDetection(b, db, tx); err != nil {//knownsec// Difficulty check
		return err
	}

	for _, tx := range b.Transactions {
		if tx.IsCoinBaseTransaction() {
			continue
		}

		if err := checkGas(tx.GasLimit, tx.GasPrice); err != nil {//knownsec// gas check
			return err
		}
	}

	//checkNonceMp := make(map[address.Address]uint64)
	return nil
}
```

Security advice: Improve the block data verification method to verify the block in detail.

4.7.6. Merkle tree root construction method【SAFE】

Audit Note: Check that the Merkle tree root was built in a reasonable manner.

Risk hazards: forged blocks, resulting in chain forks and other issues.

Risk file:

Audit process: Check that the Merkle tree root was built in a reasonable manner.

Audit results: No obvious security issues were found in the logic of block Merck tree construction in the male chain code.

```
// create a new merkle tree
func New(hash hash.Hash, data [][]byte) *MerkleTree {
	var n int

	if data == nil || len(data) == 0 {
        return nil
    }
    if n = len(data); n == 0 {
        return nil
    }
    r := &MerkleTree{
        hash: hash,
    }
    r.tree = r.mkMerkleTreeRoot(n, data)
    return r
}
......
// create a new merkle tree by slice
func (t *MerkleTree) mkMerkleTreeRoot(n int, data [][]byte) MKNode {
    switch n {
    case 1:
        return t.mkLeaf(data[0])
    default:
        i := powerOfTwo(n)
        return t.mkBranch(t.mkMerkleTreeRoot(i, data[:i]), t.mkMerkleTreeRoot((n-i), data[i:]))
    }
}
```

Security advice: None.

4.8. Transaction processing

4.8.1. The transaction signature logic 【SAFE】

Audit Notes: Check the content items of each type of transaction signature and verify that the logic is adequate.

Risk hazards: Replay of same-chain or cross-chain transactions, forge legitimate transactions, affect asset security.

Risk file:

Audit process: Check the content items of each type of transaction signature and verify that the logic is adequate.

Audit results: After testing, the transaction signature logic in the public chain code did not find obvious security problems.

```
func (tx *Transaction) MutilSign(fromPriv, toPriv []byte) (*SignedTransaction, error) { //knownsec//
Multi-signature
	if len(fromPriv) == 0 {
		return nil, fmt.Errorf("invalid from privte key")
	}

	if len(toPriv) == 0 {
		return nil, fmt.Errorf("invalid from privte key")
	}

	fsign, err := sigs.Sign(tx.From.Protocol(), fromPriv, tx.SignHash())
	if err != nil {
		return nil, err
	}

	tsign, err := sigs.Sign(tx.To.Protocol(), toPriv, tx.SignHash())
	if err != nil {
		return nil, err
	}

	mutilSign := crypto.Signature{SigType: crypto.TypeMutil, Data: append(fsign.Data,
tsign.Data...)}
```

```
	return &SignedTransaction{Transaction: *tx, Signature: mutilSign}, nil
}
```

Security advice: None.

4.8.2. Transaction validation logic【SAFE】

Audit Notes: Check that the validation logic for each type of transaction is adequate.

Risk hazards: Replay of same-chain or cross-chain transactions, forge legitimate transactions, affect asset security.

Risk file:

Audit process: Check that the validation logic for each type of transaction is adequate.

Audit results: After testing, the transaction verification logic in the public chain code did not find any obvious security problems.

```
func (st *SignedTransaction) VerifySign() error { //knownsec// Verify signature
	switch st.Type {
	case TransferTransaction, PledgeTrasnaction, PledgeBreakTransaction: //knownsec// Transfer and pledge transactions
		if err := sigs.Verify(&st.Signature, st.Caller().Payload(), st.SignHash()); err != nil {
			if err != nil {
				return err
			}
		}
	case LockTransaction, UnlockTransaction: //knownsec// Freeze/unfreeze transactions
		var l int
		if st.From.Protocol() == crypto.TypeED25519 {
```

```
            l = 64
        } else if st.From.Protocol() == crypto.TypeSecp256k1 {
            l = 65
        }

        if len(st.Signature.Data) < l {
            return fmt.Errorf("invalide mutil signature")
        }

        callerSign := &crypto.Signature{st.From.Protocol(), st.Signature.Data[:l]}
        if err := sigs.Verify(callerSign, st.Caller().Bytes(), st.SignHash()); err != nil {
            return err
        }

        receiverSign := &crypto.Signature{st.To.Protocol(), st.Signature.Data[l:]}
        if err := sigs.Verify(receiverSign, st.Receiver().Bytes(), st.SignHash()); err != nil {
            return err
        }
    case EvmContractTransaction, EvmKtoTransaction: //knownsec// EVM transaction
        evm, err := DecodeEvmData(st.Input)
        if err != nil {
            return err
        }
        if len(evm.MsgHash) > 0 {
            from := st.Caller()
            if !st.VerifyKtoSign(&from, evm.MsgHash, evm.EthData) {
                return errors.New("verify kto transaction failed!")
            }
        } else {
            if !st.VerifyEthSign(evm.EthData) {
                return errors.New("verify eth transaction failed!")
            }
        }
    }

    return nil
}
```

Security advice: None.

4.8.3. Signature verification logic【SAFE】

Audit Notes: Check that the signature verification logic is adequate.

Risk hazards: Replay of same-chain or cross-chain transactions, forge legitimate transactions, affect asset security.

Risk file:

Audit process: Check that the signature verification logic in the male chain code is adequate.

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

```
func (st *SignedTransaction) VerifySign() error { //knownsec// Verify signature
    switch st.Type {
    case TransferTransaction, PledgeTrasnaction, PledgeBreakTransaction: //knownsec// Transfer and
pledge transactions
        if err := sigs.Verify(&st.Signature, st.Caller().Payload(), st.SignHash()); err != nil {
            if err != nil {
                return err
            }
        }
    case LockTransaction, UnlockTransaction: //knownsec// Freeze/unfreeze transactions
        var l int
        if st.From.Protocol() == crypto.TypeED25519 {
            l = 64
        } else if st.From.Protocol() == crypto.TypeSecp256k1 {
            l = 65
        }

        if len(st.Signature.Data) < l {
            return fmt.Errorf("invalide mutil signature")
        }

        callerSign := &crypto.Signature{st.From.Protocol(), st.Signature.Data[:l]}
        if err := sigs.Verify(callerSign, st.Caller().Bytes(), st.SignHash()); err != nil {
            return err
        }

        receiverSign := &crypto.Signature{st.To.Protocol(), st.Signature.Data[l:]}
        if err := sigs.Verify(receiverSign, st.Receiver().Bytes(), st.SignHash()); err != nil {
            return err
        }
    case EvmContractTransaction, EvmKtoTransaction: //knownsec// EVM transaction
        evm, err := DecodeEvmData(st.Input)
        if err != nil {
            return err
        }
        if len(evm.MsgHash) > 0 {
            from := st.Caller()
            if !st.VerifyKtoSign(&from, evm.MsgHash, evm.EthData) {
                return errors.New("verify kto transaction failed!")
            }
        } else {
            if !st.VerifyEthSign(evm.EthData) {
                return errors.New("verify eth transaction failed!")
            }
        }
    }
```

```
	return nil
}
```

Security advice: None.

4.8.4. Trading logic【SAFE】

Audit Notes: Audit the logical design of transaction processing, check whether the design of priority, etc. is reasonable.

Risk hazard: double flower of assets in the same chain, trading replay.

Risk file:

Audit process: the transaction processing logic design audit, check the priority and other design is reasonable.

Audit results: After testing, the transaction processing logic in the public chain code did not find obvious security problems.

```
func (bc *Blockchain) AddBlock(block *block.Block) error { //knownsec// Add block
	......
	for index, tx := range block.Transactions { //knownsec// Traverse all transactions in the processing block
		if tx.Transaction.IsCoinBaseTransaction() { //knownsec// coinbase transaction
			txHash := tx.Hash()
			if err := setTxbyaddrKV(DBTransaction, tx.Transaction.To.Bytes(), txHash, height, uint64(index)); err != nil {
				logger.Error("Failed to set transaction", zap.Error(err), zap.String("from address", tx.Transaction.From.String()),
					zap.Uint64("amount", tx.Transaction.Amount))
				REVERT = err
				return err
			}

			//	gas := tx.Transaction.GasLimit * tx.Transaction.GasPrice
			tx.GasUsed = tx.Transaction.GasLimit * tx.Transaction.GasPrice
			blockGasU += tx.GasUsed
			if err := setMinerFee(bc, block.Miner, tx.GasUsed); err != nil { //knownsec// Miner fee
				logger.Error("Failed to set Minerfee", zap.Error(err), zap.String("from address",
block.Miner.String()), zap.Uint64("fee", tx.GasUsed))
				REVERT = err
				return err
			}
			if err := bc.setToAccount(block, &tx.Transaction); err != nil {
				logger.Error("Failed to set account", zap.Error(err), zap.String("from address",
tx.Transaction.From.String()),
					zap.Uint64("amount", tx.Transaction.Amount))
				REVERT = err
				return err
			}
		} else if tx.Transaction.IsLockTransaction() || tx.Transaction.IsUnlockTransaction()
{ //knownsec// Freeze/unfreeze transactions
			txHash := tx.Hash()
			if err := setTxbyaddrKV(DBTransaction, tx.Transaction.From.Bytes(), txHash, height,
uint64(index)); err != nil {
				logger.Error("Failed to set transaction", zap.Error(err), zap.String("from address",
tx.Transaction.From.String()),
					zap.String("to address", tx.Transaction.To.String()), zap.Uint64("amount",
tx.Transaction.Amount))
				REVERT = err
				return err
			}

			if err := setTxbyaddrKV(DBTransaction, tx.Transaction.To.Bytes(), txHash, height,
uint64(index)); err != nil {
```

```
                logger.Error("Failed to set transaction", zap.Error(err), zap.String("from address",
tx.Transaction.From.String()),
                    zap.String("to address", tx.Transaction.To.String()), zap.Uint64("amount",
tx.Transaction.Amount))
                REVERT = err
                return err
            }

            nonce := tx.Transaction.Nonce + 1
            if err := setNonce(bc.sdb, tx.Transaction.From, nonce); err != nil { //knownsec// Set up
Nonce+1
                logger.Error("Failed to set nonce", zap.Error(err), zap.String("from address",
tx.Transaction.From.String()),
                    zap.String("to address", tx.Transaction.To.String()), zap.Uint64("amount",
tx.Transaction.Amount))
                REVERT = err
                return err
            }

            tx.GasUsed = tx.Transaction.GasLimit * tx.Transaction.GasPrice //knownsec// gas fee
            blockGasU += tx.GasUsed
            if err := setMinerFee(bc, block.Miner, tx.GasUsed); err != nil { //knownsec// Set miner
fee
                logger.Error("Failed to set Minerfee", zap.Error(err), zap.String("from address",
block.Miner.String()), zap.Uint64("fee", tx.GasUsed))
                REVERT = err
                return err
            }

            if tx.Transaction.IsLockTransaction() { //knownsec// Frozen transaction
                if err := setFreezeAccount(bc, tx.Transaction.From, tx.Transaction.To,
tx.Transaction.Amount, tx.GasUsed, 1); err != nil {
```

```
                    logger.Error("Faile to setFreezeAccount", zap.String("address",
tx.Transaction.From.String()),
                        zap.Uint64("amount", tx.Transaction.Amount))
                    REVERT = err
                    return err
                }
            } else { //knownsec// Unfreeze transactions
                if err := setFreezeAccount(bc, tx.Transaction.From, tx.Transaction.To,
tx.Transaction.Amount, tx.GasUsed, 0); err != nil {
                    logger.Error("Faile to setFreezeAccount", zap.String("address",
tx.Transaction.From.String()),
                        zap.Uint64("amount", tx.Transaction.Amount))
                    REVERT = err
                    return err
                }
            }

        } else if tx.Transaction.IsPledgeTrasnaction() || tx.IsPledgeDefaultTransaction()
{ //knownsec// Pledge transaction
            logger.SugarLogger.Debug(">>>>>>>>>>>>>>>>>>>>>>Start IsPledgeTrasnaction")
            txHash := tx.Hash()
            if err := setTxbyaddrKV(DBTransaction, tx.Transaction.From.Bytes(), txHash, height,
uint64(index)); err != nil {
                logger.Error("Failed to set transaction", zap.Error(err), zap.String("hash",
transaction.HashToString(txHash)))
                REVERT = err
                return err
            }
            nonce := tx.Transaction.Nonce + 1
            if err := setNonce(bc.sdb, tx.Transaction.From, nonce); err != nil { //knownsec// Set up
Nonce
                logger.Error("Failed to set nonce", zap.Error(err), zap.String("hash",
transaction.HashToString(txHash)))
```

```
                REVERT = err
                return err
            }
            {
                bfp, _ := getTotalPledge(bc.sdb, tx.From) //knownsec// Total pledge amount
before processing
                logger.SugarLogger.Debugf("Before pledge from[%v] total pledge[%v]\n",
tx.From, bfp)
            }

            var destroy uint64
            if tx.Transaction.IsPledgeTrasnaction() {
                err := bc.handlePledgeTransaction(block.Height, &tx.Transaction)
                if err != nil {
                    logger.Error("Failed to Pledge", zap.Error(err), zap.String("hash",
transaction.HashToString(txHash)))
                    REVERT = err
                    return err
                }
            } else if tx.IsPledgeDefaultTransaction() {
                des, err := bc.handlePledgeBreakTransaction(block, &tx.Transaction)
                if err != nil {
                    logger.Error("handlePledgeBreakTransaction Failed", zap.Error(err),
zap.String("hash", transaction.HashToString(txHash)))
                    REVERT = err
                    return err
                }
                tx.Input = miscellaneous.EB64func(destroy)
                destroy = des
            }

            {
                afp, _ := getTotalPledge(bc.sdb, tx.From) //knownsec// Total pledge amount after
processing
                logger.SugarLogger.Debugf("After pledge from[%v] total pledge[%v]\n",
tx.From, afp)
            }

            tx.GasUsed = tx.Transaction.GasLimit * tx.Transaction.GasPrice
            blockGasU += tx.GasUsed

            if err := setMinerFee(bc, block.Miner, tx.GasUsed); err != nil { //knownsec// Set miner
fee
                logger.Error("Failed to set Minerfee", zap.Error(err), zap.String("hash",
transaction.HashToString(txHash)), zap.Uint64("gasUsed", tx.GasUsed))
                REVERT = err
                return err
            }
            if err := setAccount(bc, tx, destroy); err != nil { //knownsec// Set account information
                logger.Error("Failed to set balance", zap.Error(err), zap.String("from address",
tx.Transaction.From.String()),
                    zap.String("to address", tx.Transaction.To.String()), zap.Uint64("amount",
tx.Transaction.Amount))
                REVERT = err
                return err
            }
        } else if tx.Transaction.IsEvmContractTransaction() { //knownsec//EVM contract trading
            txHash := tx.Hash()
            if err := setTxbyaddrKV(DBTransaction, tx.Transaction.From.Bytes(), txHash, height,
uint64(index)); err != nil {
                logger.Error("Failed to set transaction", zap.Error(err), zap.String("hash",
transaction.HashToString(txHash)))
                REVERT = err
                return err
            }
```

```
		gasLeft, err := bc.handleContractTransaction(block, DBTransaction, tx, index)
		if err != nil {
			logger.Error("Failed to HandleContractTransaction", zap.Error(err),
zap.String("hash", transaction.HashToString(txHash)))
			REVERT = err
			return err
		}

		evmcfg := bc.evm.GetConfig()
		if evmcfg.GasLimit < gasLeft {
			logger.Error("Failed to HandleContractTransaction",
zap.Error(fmt.Errorf("hash[%v],evm gaslimit[%v] < gasLeft[%v]", transaction.HashToString(txHash),
evmcfg.GasLimit, gasLeft)))
			REVERT = fmt.Errorf("error: hash[%v],evm gaslimit[%v] < gasLeft[%v]",
transaction.HashToString(txHash), evmcfg.GasLimit, gasLeft)
			return REVERT
		}
		tx.GasUsed = evmcfg.GasLimit - gasLeft
		blockGasU += tx.GasUsed

		if err := setMinerFee(bc, block.Miner, tx.GasUsed); err != nil { //knownsec// Set miner
fee
			logger.Error("Failed to set Minerfee", zap.Error(err), zap.String("hash",
transaction.HashToString(txHash)), zap.Uint64("gasUsed", tx.GasUsed))
			REVERT = err
			return err
		}

		// update balance
		if err := setAccount(bc, tx); err != nil { //knownsec// Set up an account
			logger.Error("Failed to set balance", zap.Error(err), zap.String("from address",
tx.Transaction.From.String()),
                        zap.String("to address", tx.Transaction.To.String()), zap.Uint64("amount",
tx.Transaction.Amount))
                    REVERT = err
                    return err
                }
        } else { //knownsec// Other types of transactions
                if tx.Transaction.Input != nil || !bytes.Equal(tx.Transaction.Input, []byte("")) ||
len(tx.Transaction.Input) != 0 { //knownsec// There is input
                    ipt := string(tx.Input)
                    sc := parser.Parser([]byte(tx.Input))
                    coAddr, _ := tx.Transaction.From.NewCommonAddr()
                    e, err := exec.New(DBTransaction, bc.sdb, sc, coAddr.String()) //knownsec//
Generate new execution
                    if err != nil {
                        logger.Error("Failed to new exec", zap.String("script", ipt),
                            zap.String("from address", tx.From.String()))
                        ipt += " InsufficientBalance"
                        tx.Input = []byte(ipt)
                        block.Transactions[index] = tx
                        REVERT = err
                        return err
                    }

                    if e != nil {
                        if err := e.Flush(DBTransaction, bc.sdb); err != nil { //knownsec// flush
                            logger.Error("Failed to flush exec", zap.String("script", ipt),
                                zap.String("from address", tx.From.String()))
                            ipt += " ExecutionFailed"
                            tx.Input = []byte(ipt)
                            block.Transactions[index] = tx
                            REVERT = err
                            return err
                        }
```

```
			}
		}

		txHash := tx.Hash()
		if err := setTxbyaddrKV(DBTransaction, tx.Transaction.From.Bytes(), txHash, height,
uint64(index)); err != nil {
			logger.Error("Failed to set transaction", zap.Error(err), zap.String("from address",
tx.Transaction.From.String()),
				zap.String("to address", tx.Transaction.To.String()), zap.Uint64("amount",
tx.Transaction.Amount))
			REVERT = err
			return err
		}

		if err := setTxbyaddrKV(DBTransaction, tx.Transaction.To.Bytes(), txHash, height,
uint64(index)); err != nil {
			logger.Error("Failed to set transaction", zap.Error(err), zap.String("from address",
tx.Transaction.From.String()),
				zap.String("to address", tx.Transaction.To.String()), zap.Uint64("amount",
tx.Transaction.Amount))
			REVERT = err
			return err
		}
		// update nonce,txs in block must be ordered
		nonce := tx.Transaction.Nonce + 1
		if err := setNonce(bc.sdb, tx.Transaction.From, nonce); err != nil {
			logger.Error("Failed to set nonce", zap.Error(err), zap.String("from address",
tx.Transaction.From.String()),
				zap.String("to address", tx.Transaction.To.String()), zap.Uint64("amount",
tx.Transaction.Amount))
			REVERT = err
			return err
		}
```

```
                //err

                tx.GasUsed = tx.Transaction.GasLimit * tx.Transaction.GasPrice
                blockGasU += tx.GasUsed

                if err := setMinerFee(bc, block.Miner, tx.GasUsed); err != nil {
                    logger.Error("Failed to set Minerfee", zap.Error(err), zap.String("from address",
block.Miner.String()), zap.Uint64("fee", tx.GasUsed))
                    REVERT = err
                    return err
                }

                // update balance
                if err := setAccount(bc, tx); err != nil {
                    logger.Error("Failed to set balance", zap.Error(err), zap.String("from address",
tx.Transaction.From.String()),
                        zap.String("to address", tx.Transaction.To.String()), zap.Uint64("amount",
tx.Transaction.Amount))
                    REVERT = err
                    return err
                }
            }
        }
    ......
}
```

Security advice: None.

4.8.5. Transaction fee settings【SAFE】

Audit Note: Check that the fees corresponding to each atomic operation performed by the transaction processing/contract are proportional to the consumption of resources.

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

Risk file:

Audit process: Check that the fees corresponding to each atomic operation performed by the transaction processing/contract are proportional to the consumption of resources.

Audit results: After testing, the transaction fee design in the public chain code is reasonable, no obvious security problems have been found.

```
// Transaction
type Transaction struct {
	Version uint64
	Type      TransactionType
	From      address.Address
	To        address.Address
	Amount    uint64
	Nonce     uint64

	GasLimit   uint64
	GasFeeCap uint64
	GasPrice   uint64

	Input []byte
}
......
```

```
// GasCap gas fee upper limit
func (t *Transaction) GasCap() uint64 {
        return t.GasFeeCap * t.GasPrice
}
```

Security advice: None.

4.8.6. Transaction fee assessment【SAFE】

Audit Note: Check whether the fee assessment corresponding to each atomic operation performed by the transaction processing/contract is proportional to the consumption of resources.

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

Risk file:

Audit process: Check whether the fee assessment corresponding to each atomic operation performed by the transaction processing/contract is proportional to resource consumption.

Audit results: After testing, the transaction fee estimate in the public chain code is reasonable, no obvious security problems have been found.

Security advice: None.

4.8.7. Limits on transaction processing resources【SAFE】

Audit Notes: Check whether resource limitations such as trading pools, verification calculations, hard disk addressing, etc. are reasonable.

Risk hazard: Take up local storage, CPU resources DoS nodes, severe can cause node programs to crash.

Risk file:

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

Audit results: After audit, no related risks were found.

```
const (
	PendingLimit = 100
	poolCap      = 10000
	timesub      = 100
)
```

Security advice: None.

4.9. Contract virtual machines

4.9.1. Contract execution logic 【SAFE】

Audit notes: Check whether there is an unexpected control flow in the contract execution logic, and whether there are reasonable restrictions on each link.

Risk hazard: Unexpected behavior in contract execution can lead to data tampering or node program crashes.

Risk file:

Audit process: Check the contract execution logic for unexpected control flow, whether there are restrictions on each link.

Audit results: After audit, no related risks were found.

Security advice: None.

4.9.2. Contract deployment security【SAFE】

Audit Notes: Check for design flaws and security risks during smart contract deployment, and whether there are reasonable limits on each link.

Risk hazard: Unexpected behavior in contract deployment can cause program crashes.

Risk file:

Audit process: Check the smart contract deployment process for design defects and security risks, whether there are reasonable restrictions.

Audit results: After audit, no related risks were found.

Security advice: None.

4.9.3. Contract interface debugging【SAFE】

Audit notes: Debug the contract interface-related functions whether there are defects or security issues, check whether the links have made reasonable restrictions.

Risk hazards: Sensitive information disclosure, program crash.

Risk file:

Audit process: debugging contract interface-related functions whether there are defects or security issues, check whether the links have made reasonable restrictions.

Audit results: After audit, no related risks were found.

Security advice: None.

4.9.4. Contract virtual machine sandbox escape【SAFE】

Audit Note: Check the virtual machine sandbox logic, determine whether the critical object interception is thorough, test whether attacks such as common character escapes are effective, and whether objects outside the sandbox environment can be accessed.

Risk hazard: Bypassing sandbox restrictions can affect the execution logic of other contracts or lead to DoS and remote command execution.

Risk file:

Audit process: Check the virtual machine sandbox logic, determine whether the critical object interception is thorough, test whether attacks such as common character escapes are effective, and whether objects outside the sandbox environment can be accessed.

Audit results: After audit, no related risks were found.

Security advice: None.

4.10. Account system

4.10.1. Account system authentication【SAFE】

Audit Notes: Check the authentication logic for problems.

Risk hazards: Falsifying identities bypasses permission checks, which can pose a threat to asset security.

Risk file:

Audit process: Check for problems with authentication logic.

Audit results: After testing, there is no account wallet system related operation logic in the public chain code.

Security advice: None.

4.10.2.Account CRUD logic【SAFE】

Audit Note: Check the account system CRUD logic for illegal boundaries and when the operation takes effect, and test for vulnerabilities and impact on nodes and data.

Risk hazards: Attackers can maliciously delete user accounts on the premise of having access to the user's client and cause damage to user assets, or when the user misuses, there is no way to slow down the error, resulting in irreparable asset losses.

Risk file:

Audit process: Check whether there is a problem with the CRUD logic of the account system.

Audit result: After testing, there is no account wallet system related operation logic in the public chain code.

Safety advice: None.

4.11. Core logic

4.11.1. Block reward distribution logic【SAFE】

Audit Note: Check that the block reward distribution logic design in the male chain code is consistent with the description documentation.

Risk hazards: Subject to availability.

Risk file:

Audit process: Check that the block reward distribution logic design in the male chain code is consistent with the description documentation.

Audit results: After auditing, no related risks were found. The initial block reward is 8848000000000000000. After subtracting the old chain reward, every 3,153,600 blocks in the new chain, the block reward becomes 90% of the previous stage.

```
func GetMinerAmount(height uint64) uint64 { //knownsec// Block reward
	const newChainHeight uint64 = InitHeight // Need to initialize the block height of the old chain
	const ktoTotal uint64 = 8848000000000000000
	const oldPhase1 uint64 = 31536000 * 49460000000
	const oldPhase2 uint64 = (newChainHeight - 31536000) * 8 / 10 * 49460000000
	const newChainBalance uint64 = 1000000000000000000 + oldPhase1 + oldPhase2 // The old
chain has issued coins
	var newChantotal uint64 = ktoTotal - newChainBalance //knownsec// Total amount of new chains
	if height < newChainHeight {
		err := fmt.Errorf("error height < newChainHeight")
		panic(err)
	}

	height = height - newChainHeight // The actual block height of the new chain
```

```
        var total uint64 = newChantotal / 3153600 / 10
        x := height / 3153600
        for i := 0; uint64(i) < x; i++ {
                newChantotal = newChantotal - (3153600 * total) //knownsec// 90%
                total = newChantotal / 3153600 / 10
        }
        return total
}
```

Security advice: None.

4.11.2.Miner reward release logic【SAFE】

Audit Note: Check that the logical design of miner incentive release in the male chain code is consistent with the description documentation.

Risk hazards: Subject to availability.

Risk file:

Audit process: Check that the logical design of miner incentive release in the male chain code is consistent with the description documentation.

Audit results: After auditing, no related risks were found. Miner rewards are composed of block fixed rewards and transaction fees. The block fixed rewards will be released in 120 days.

```
//pledge 70% mined
func (bc *Blockchain) handleMinedPledge(block *block.Block, to address.Address, amount uint64)
error {
        totMinedPledge, err := bc.getTotalMined(to)
        if err != nil {
                return err
        }
        if block.Miner == to {
                minedPledge := amount * MINEDDIVID / DIVIDEND //knownsec//Block reward 70%
                totMinedPledge += minedPledge
                err := bc.setTotalMined(to, Uint64ToBigInt(totMinedPledge)) //knownsec// Include the total
mining volume of the address
                if err != nil {
                        return err
                }
        }
        return nil
}

//release mined pledge
func (bc *Blockchain) releaseMined(block *block.Block) error { //knownsec// Release mining rewards
        currHeight := block.Height
        var numbers uint64
        logger.Info("start releaseMined", zap.String("success", fmt.Sprintf("currentHeight:%v,release
times:%v", block.Height, numbers)))
        for control := PledgeCycle; control > 0; control-- {//knownsec// Traverse the first 120-day block
                totalBlocksNum := control * DayMinedBlocks
                if currHeight <= uint64(totalBlocksNum) { //knownsec// Skip beyond the current block
height                  continue
                }
                releaseBlock := currHeight - uint64(totalBlocksNum)

                miner, releaseValue, err := bc.getReleaseInfo(releaseBlock)
                if err != nil {
                        logger.Error("getReleaseInfo error", zap.Error(err))
                        return err
                }
                totMinedPledge, err := bc.getTotalMined(miner)
                if err != nil {
```

```
                logger.Error("getTotalMined error", zap.Error(err))
                return err
            }
            logger.Info("getReleaseInfo", zap.String("success",
fmt.Sprintf("miner:%v,releaseHeight:%v,releaseValue:%v,totMinedPledge:%v", miner.String(),
releaseBlock, releaseValue, totMinedPledge)))
            if totMinedPledge > 0 {

                if totMinedPledge >= releaseValue { //knownsec// Total mining volume minus release
amount
                    totMinedPledge -= releaseValue
                }
                if totMinedPledge < releaseValue && totMinedPledge > 0 {
                    totMinedPledge = 0
                }

                err = bc.setTotalMined(miner, Uint64ToBigInt(totMinedPledge))
                if err != nil {
                    logger.Error("setTotalMined error", zap.Error(err))
                    return err
                }
                numbers++
            }
        }
        logger.Info("end releaseMined", zap.String("success", fmt.Sprintf("currentHeight:%v,release
times:%v", currHeight, numbers)))
        return nil
}

func (bc *Blockchain) getReleaseInfo(releaseH uint64) (address.Address, uint64, error)
{ //knownsoec// Get block release information
        var releaseValue uint64
        addr := address.Address{}
```

```
	b, err := bc.getBlockByHeight(releaseH)
	if err != nil {
		return addr, releaseValue, err
	}

	for _, tx := range b.Transactions { //knownsec// Traverse block transactions
		if tx.IsCoinBaseTransaction() { //knownsec// coinbase trade
			releaseValue = tx.Amount * MINEDDIVID / DIVIDEND / PledgeCycle //knownsec// Block reward*70%/120d, i.e. the block reward pledge amount will be released in 120 days
			addr = b.Miner
			break
		}
	}
	if releaseValue == 0 {
		return addr, releaseValue, fmt.Errorf("getReleaseInfo value failed!")
	}
	//logger.Info("getReleaseInfo", zap.String("success", fmt.Sprintf("miner:%v,releaseHeight:%v,releaseValue:%v", b.Miner.String(), b.Height, releaseValue)))
	return addr, releaseValue, nil
}
```

Security advice: None.

4.11.3.The transaction validates the logic【SAFE】

Audit Note: Check that the transaction in the public chain code validates the logical design is reasonable.

Risk hazards: Subject to availability.

Risk file:

Audit process: Check that the transaction in the male chain code validates the logical design is reasonable.

Audit results: After audit, the transaction effective verification logic design is reasonable.

```
func (p *Pool) Add(st *transaction.SignedTransaction) error {//knownsec// Add a transaction to the transaction pool
	if st.GasLimit*st.GasPrice < blockchain.MINGASLIMIT || st.GasLimit*st.GasPrice > blockchain.MAXGASLIMIT {
		return fmt.Errorf("gas is too small or too big,gas limit:%d gas price:%d", st.GasLimit, st.GasPrice)
	}

	if st.Type == transaction.TransferTransaction && len(st.Input) != 0 {
		return fmt.Errorf("this is token transaction")
	}

	if err := st.VerifySign(); err != nil {//knownsec// Verify transaction signature
		return err
	}

	p.qlock.Lock()
	defer p.qlock.Unlock()

	return p.add(st)
}
```

Security advice: None.

4.11.4.Miners create logical designs【SAFE】

Audit note: Check whether the logical design of the miner's creation is reasonable.

Risk hazard: It depends on the specific situation.

Risk document:

Audit process: Check whether the logical design of the miner creation is reasonable.

Audit result: After audit, the logic design of miner creation is reasonable and correct.

```
func New(cfg *Config, bc *blockchain.Blockchain, tp *txpool.Pool, node *p2p.Node, cb
*consensus.BlockChain, mode int, AdvertiseAddr string) (*Miner, error) {//knownsec// Create a miner

	if cfg == nil {
		return nil, fmt.Errorf("configuration cannot be nil")
	}

	fmt.Println(cfg.MiningAddr)
	miningAddr, err := address.NewAddrFromString(cfg.MiningAddr)
	if err != nil {
		fmt.Println("NewAddrFromString", err)
		return nil, err
	}

	m := &Miner{ //knownsec// Create Miner instance
		coinbaseAddr: miningAddr,
		c:            make(chan block.Block, 100),
		p:            make(chan *block.Block, 100),
		r:            make(chan *block.Block, 100),
        download:         make(chan bool),
        started:          false,
        tp:               tp,
        bc:               bc,
        cbc:              cb,

        node:             node,
        AdvertiseAddr: AdvertiseAddr,
        GenesisHash:      cfg.GenesisHash,
}

fmt.Println("==========miner GenesisHash========", cfg.GenesisHash)

initBits, err := strconv.ParseUint(cfg.DifficultyBits, 0, 32)
if err != nil {
        fmt.Println("ParseUint", err)
        return nil, err
}

m.InitBits = uint32(initBits)
globalBits = m.InitBits

m.calcNextRequiredDifficulty(m.coinbaseAddr)

if !cfg.NoMining {
        m.MiningSig = make(chan struct{}, 1)
        m.BreakMiningSig = make(chan struct{}, 1)
        go m.MultiCalcDifficulty()
}

go m.KtocoinMiner(mode) //knownsec// Mining

//m.MiningSig <- struct{}{}
```

```
    return m, nil
}
```

Security advice: None.

4.12. Other

4.12.1.Database security【SAFE】

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 document:

Audit process: 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.

Audit result: After audit,the badger database is used in the code. No related risks were found.

Safety advice: None.

4.12.2.Message processing security【SAFE】

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

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

Risk document:

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

Audit result: After audit, CBOR encoding is mainly used for message serialization and deserialization operations. No related risks were found.

Safety advice: None.

4.12.3.Thread safe【SAFE】

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

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

Risk document:

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

Audit results: After auditing, no concurrent security related risks were found.

Safety advice: None.

4.12.4.Security of environment variables【SAFE】

Audit description: Check the dependency status of environment variables, and test the influence of controllable environment variables on node programs.

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

Risk document:

Audit process: Check the dependency status of environment variables, and test the influence of controllable environment variables on node programs.

Audit result: After testing, the public chain code does not involve modification of environmental variables except for the test code, and no obvious security issues have been found in related logic.

Safety advice: None.

4.12.5.Centralization detection【SAFE】

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

Risk hazards: 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 document:

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: None.

4.12.6.File permission security【SAFE】

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

Risks and hazards: 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 document:

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

Audit result: After testing, no related risks were found.

Safety advice: None.

4.12.7.Concurrency security risks【SAFE】

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 document:

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

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

Safety advice: None.

4.12.8.JSON oversized request packet DOS【SAFE】

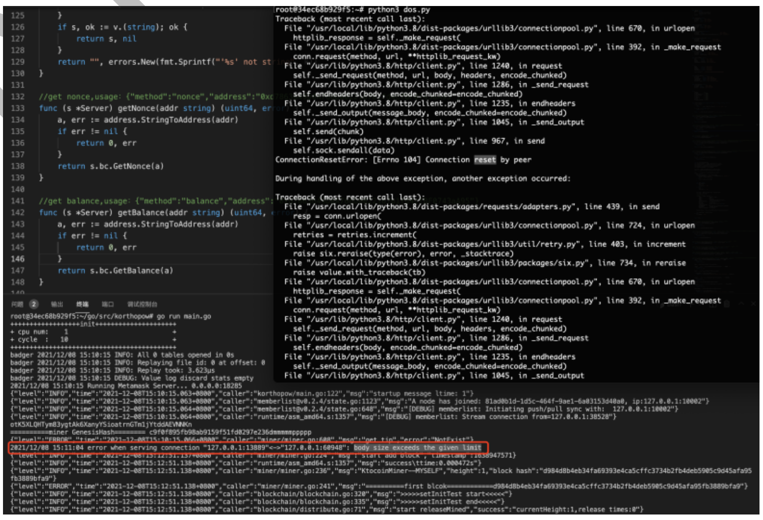
Audit description: Test by constructing malicious request data packets (malformed data packets, oversized request data packets, illegal parameters, illegal formats, etc.) to see if it will lead to the leakage of node DOS and sensitive information.

Risk hazard: An attacker can construct a larger data packet and send a malicious RPC request to destroy the node or obtain sensitive information.

Risk file:

Audit process: Testing is performed by passing in malformed data, and at the same time, analyzing and auditing the verification of address parameters in the code.

Audit result: constructing oversized, malformed, and illegal JSON request data packets will not cause node DOS



```
    }
    if s, ok := v.(string); ok {
        return s, nil
    }
    return "", errors.New(fmt.Sprintf("'%s' not st
}

//get nonce,usage: {"method":"nonce","address":"0x
func (s *Server) getNonce(addr string) (uint64, er
    a, err := address.StringToAddress(addr)
    if err != nil {
        return 0, err
    }
    return s.bc.GetNonce(a)
}

//get balance,usage: {"method":"balance","address":
func (s *Server) getBalance(addr string) (uint64,
    a, err := address.StringToAddress(addr)
    if err != nil {
        return 0, err
    }
    return s.bc.GetBalance(a)
}
```

```
root@34ec68b929f5:~# python3 dos.py
Traceback (most recent call last):
  File "/usr/local/lib/python3.8/dist-packages/urllib3/connectionpool.py", line 670, in urlopen
    httplib_response = self._make_request(
  File "/usr/local/lib/python3.8/dist-packages/urllib3/connectionpool.py", line 392, in _make_request
    conn.request(method, url, **httplib_request_kw)
  File "/usr/lib/python3.8/http/client.py", line 1240, in request
    self._send_request(method, url, body, headers, encode_chunked)
  File "/usr/lib/python3.8/http/client.py", line 1286, in _send_request
    self.endheaders(body, encode_chunked=encode_chunked)
  File "/usr/lib/python3.8/http/client.py", line 1235, in endheaders
    self._send_output(message_body, encode_chunked=encode_chunked)
  File "/usr/lib/python3.8/http/client.py", line 1045, in _send_output
    self.send(chunk)
  File "/usr/lib/python3.8/http/client.py", line 967, in send
    self.sock.sendall(data)
ConnectionResetError: [Errno 104] Connection reset by peer

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "/usr/local/lib/python3.8/dist-packages/requests/adapters.py", line 439, in send
    resp = conn.urlopen(
  File "/usr/local/lib/python3.8/dist-packages/urllib3/connectionpool.py", line 724, in urlopen
    retries = retries.increment(
  File "/usr/local/lib/python3.8/dist-packages/urllib3/util/retry.py", line 403, in increment
    raise six.reraise(type(error), error, _stacktrace)
  File "/usr/local/lib/python3.8/dist-packages/urllib3/packages/six.py", line 734, in reraise
    raise value.with_traceback(tb)
  File "/usr/local/lib/python3.8/dist-packages/urllib3/connectionpool.py", line 670, in urlopen
    httplib_response = self._make_request(
  File "/usr/local/lib/python3.8/dist-packages/urllib3/connectionpool.py", line 392, in _make_request
    conn.request(method, url, **httplib_request_kw)
  File "/usr/lib/python3.8/http/client.py", line 1240, in request
    self._send_request(method, url, body, headers, encode_chunked)
  File "/usr/lib/python3.8/http/client.py", line 1286, in _send_request
    self.endheaders(body, encode_chunked=encode_chunked)
  File "/usr/lib/python3.8/http/client.py", line 1235, in endheaders
    self._send_output(message_body, encode_chunked=encode_chunked)
  File "/usr/lib/python3.8/http/client.py", line 1045, in _send_output
```

```
root@34ec68b929f5:~/go/src/korthopow# go run main.go
+++++++++++++++++init++++++++++++++++++++
+ cpu num:    1                          +
+ cycle  :   10                          +
+++++++++++++++++++++++++++++++++++++++++
badger 2021/12/08 15:10:15 INFO: All 0 tables opened in 0s
badger 2021/12/08 15:10:15 INFO: Replaying file id: 0 at offset: 0
badger 2021/12/08 15:10:15 INFO: Replay took: 3.623µs
badger 2021/12/08 15:10:15 DEBUG: Value log discard stats empty
2021/12/08 15:10:15 Running Metamask Server... 0.0.0.0:18285
{"level":"INFO","time":"2021-12-08T15:10:15.063+0800","caller":"korthopow/main.go:122","msg":"startup message ltime: 1"}
{"level":"INFO","time":"2021-12-08T15:10:15.063+0800","caller":"memberlist@v0.2.4/state.go:1123","msg":"A node has joined: 81ad0b1d-1d5c-464f-9ae1-6a03153d40a0, ip:127.0.0.1:10002"}
{"level":"INFO","time":"2021-12-08T15:10:15.064+0800","caller":"memberlist@v0.2.4/state.go:648","msg":"[DEBUG] memberlist: Initiating push/pull sync with:  127.0.0.1:10002"}
{"level":"INFO","time":"2021-12-08T15:10:15.064+0800","caller":"runtime/asm_amd64.s:1357","msg":"[DEBUG] memberlist: Stream connection from=127.0.0.1:38528"}
otK5XLQHTym83ygtAk6XanyYSioatrnGTm1jYtddAEVNNKn
==========miner GenesisHash======== c9f0f895fb98ab9159f51fd0297e236dmmmmmppppp
{"level":"ERROR","time":"2021-12-08T15:10:15.066+0800","caller":"miner/miner.go:608","msg":"get tip","error":"NotExist"}
2021/12/08 15:11:04 error when serving connection "127.0.0.1:13889"<->"127.0.0.1:60948": body size exceeds the given limit
{"level":"INFO","time":"2021-12-08T15:12:51.137+0800","caller":"miner/miner.go:224","msg":"start add block","timestamp":1638947571}
{"level":"INFO","time":"2021-12-08T15:12:51.138+0800","caller":"runtime/asm_amd64.s:1357","msg":"success\ttime:0.000472s"}
{"level":"INFO","time":"2021-12-08T15:12:51.138+0800","caller":"miner/miner.go:236","msg":"KtocoinMiner--MYSELF","height":1,"block hash":"d984d8b4eb34fa69393e4ca5cffc3734b2fb4deb5905c9d45afa95
fb3889bfa9"}
{"level":"ERROR","time":"2021-12-08T15:12:51.138+0800","caller":"miner/miner.go:241","msg":"==========first blcok===========d984d8b4eb34fa69393e4ca5cffc3734b2fb4deb5905c9d45afa95fb3889bfa9"}
{"level":"INFO","time":"2021-12-08T15:12:51.138+0800","caller":"blockchain/blockchain.go:320","msg":">>>>>setInitTest start<<<<<"}
{"level":"INFO","time":"2021-12-08T15:12:51.138+0800","caller":"blockchain/blockchain.go:335","msg":">>>>>setInitTest end<<<<<"}
{"level":"INFO","time":"2021-12-08T15:12:51.138+0800","caller":"blockchain/distribute.go:71","msg":"start releaseMined","success":"currentHeight:1,release times:0"}
```

Safety advice: None.

4.12.9.Address parameter legality verification【SAFE】

Audit description: Audit and test whether the address parameters in the public chain have been verified for legitimacy.

Risk hazard: The lack of validity check logic of address parameters may lead to the leakage of sensitive information. In severe cases, it may lead to unrecoverable panic or even node crash.

Risk document:

Audit process: Testing is performed by passing in malformed data, and at the same time, analyzing and auditing the verification of address parameters in the code.

Audit result: After inspection, it was found that the legality of the address was verified.

```
func decode(str string) (Address, error) { //knownsec// Address decoding
        if len(str) < AddressSize { //knownsec// Length check
                return Undef, fmt.Errorf("invalid address string length")
        }

        if str[:AddresPrefixSize] != prefixSet[CurrentNetWork] { //knownsec// Network type verification
                return Undef, fmt.Errorf("unknow address network")
        }

        var err error
        var payload []byte
        var protocol crypto.SigType = crypto.TypeUnknown
        switch parseAddrStrProtocol(str[AddresPrefixSize:]) {
        case crypto.TypeSecp256k1:
```

```
        payload, err = hex.DecodeString(str[AddresPrefixSize+1:])
        if err != nil {
            return Undef, err
        }
        protocol = crypto.TypeSecp256k1
    case crypto.TypeED25519:
        payload = addrcodec.DecodeAddr(str[AddresPrefixSize:])
        protocol = crypto.TypeED25519
    default:
        return Undef, fmt.Errorf("unknow address type") //knownsec// Error handling unknown address type
    }

    return newAddress(protocol, payload)
}
```

Safety advice: None.

4.12.10. Store data cold and hot separation 【SAFE】

Audit note: Check the security of the cold and hot separation of stored data. Check whether there is a logic error in the method of cold and hot separation of test data.

Risk hazard: may lead to data leakage.

Risk document:

Audit process: Check the security of the cold and hot separation of stored data. Check whether there is a logic error in the method of cold and hot separation of test data.

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

Safety advice: None.

5. Appendix A: Common Security Coding Problems

5.1. Signed integer overflow

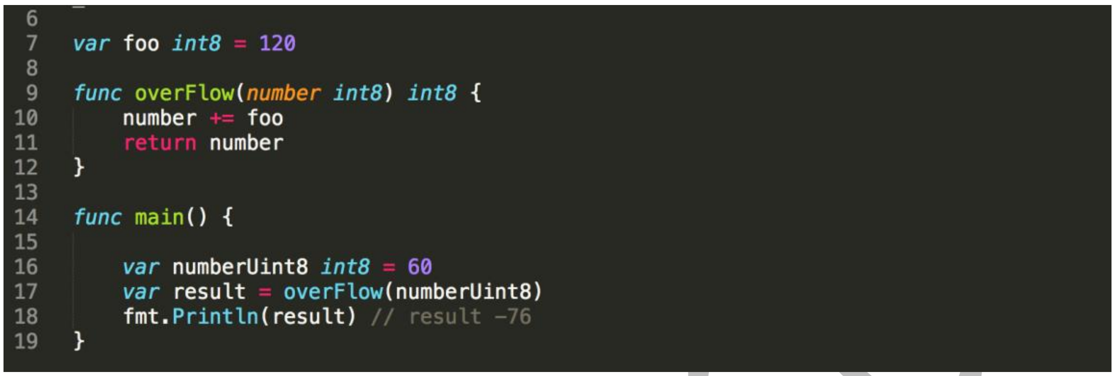
Integer types in Go are divided into signed integers and unsigned integers. The highest bit of a signed integer represents the sign (positive or negative), the remaining bits represent the magnitude of the value, and all the bits of the unsigned integer are used to represent the magnitude of the value. The value range of a signed integer is [−2n−1, 2n−1−1]. When the value of a signed integer exceeds the value range of a signed integer, an integer overflow will occur, which leads to the importance of signed integer overflow. One of the reasons is improper operation of signed integer operations, common operations "+", "-", "*", "/", "%", "++", "–", etc., if there is no value The range is judged and limited, which can easily lead to the overflow of signed integers.

5.1.1. Signed integer overflow hazards

Integer overflow will cause numerical error. Depending on the location where the wrong value is used (for example, wrong data is used for memory operations, or the wrong value causes the loop to become true, etc.), it may cause different security issues, including rejection Security issues such as service attacks, memory corruption, and incorrect blockchain transfer results.

5.1.2. Sample code

The sample code used in this chapter is derived from example/interger_overflow_int8.go



```go
var foo int8 = 120

func overFlow(number int8) int8 {
    number += foo
    return number
}

func main() {

    var numberUint8 int8 = 60
    var result = overFlow(numberUint8)
    fmt.Println(result) // result -76
}
```

In the above code, the overFlow function adds 120 to the incoming parameter and then returns. In the Go language, the value range represented by int8 is -128-127, that is, when the value exceeds 127, an overflow will occur. When the parameter is passed in When the value is 60, the result of the operation is 180, which exceeds the maximum value range represented by int8, which leads to integer overflow problems. The final result is -76.

5.1.3. How to avoid signed integer overflow

(1) When performing signed integer operations, it is necessary to effectively judge the value range of signed integers.

(2) When performing operations on signed integers from untrusted sources, additional attention is required.

(3) Automated detection using source code static analysis tools can effectively find the signed integer overflow problem in the source code.

5.2. Cross-border access

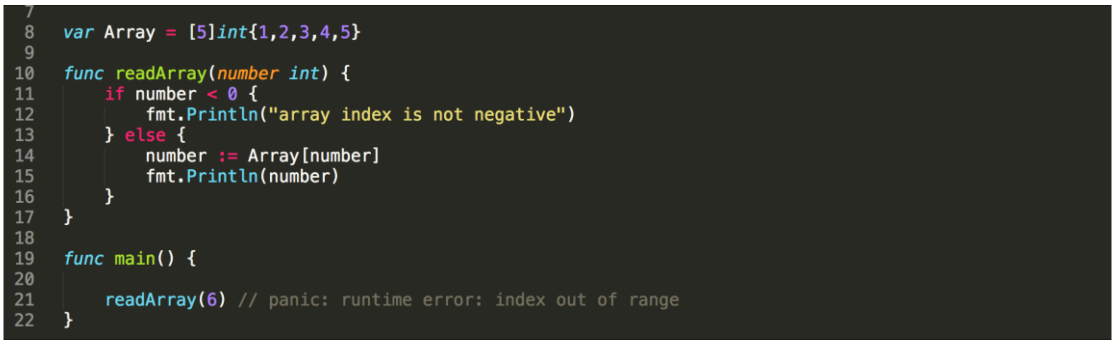
The out-of-bounds access simply means that a piece of memory is pre-applied, but when this memory is used, it exceeds the scope of the application and causes an out-of-bounds. For example, when a program accesses an element in an array, if the index value exceeds the length of the array, it will access the memory outside the array. Go does not check the boundaries of arrays and slices. It does not check whether the subscript is out of range to improve the efficiency of the program, but at the same time, it also delegates the task of checking whether the out of range is over to the developer. Therefore, the developer needs to pay extra attention to avoiding it when writing the program. Cross-border access.

5.2.1. Cross-border access hazards

Out-of-bounds access is a common flaw in the Go language. It does not necessarily cause compilation errors, and the consequences are uncertain. When there is an out-of-bounds, because it is impossible to know the content stored in the accessed space, uncertain behavior may occur, which may be caused by program crashes and unexpected operation results.

5.2.2. Sample code

The sample code used in this chapter is from example/interger_overflow_uint8.go



```
 8  var Array = [5]int{1,2,3,4,5}
 9
10  func readArray(number int) {
11      if number < 0 {
12          fmt.Println("array index is not negative")
13      } else {
14          number := Array[number]
15          fmt.Println(number)
16      }
17  }
18
19  func main() {
20
21      readArray(6) // panic: runtime error: index out of range
22  }
```

In the above code, the number is judged in line 10, but only whether the length is negative, and the upper limit is not limited (boundary check is incomplete). When the value of number is greater than 4, Array[number] The array subscript is out of bounds.

5.2.3. How to avoid out-of-bounds access

(1) Carry out effective boundary inspection to ensure that the operation is within the legal scope. Especially when using external input data as a data source for memory-related operations, special attention should be paid to boundary checking. Contaminated data is one of the important reasons for cross-border access.

(2) Explicitly specifying the array boundary can not only improve the readability of the program, but also, most compilers will give warnings when the length of the array is less than the length of the initialization value list. These warning messages can help developers as soon as possible Find out-of-bounds issues.

(3) When using loops to traverse array elements, pay attention to prevent off-by-one (one byte out of bounds) errors.

5.3. Null pointer references

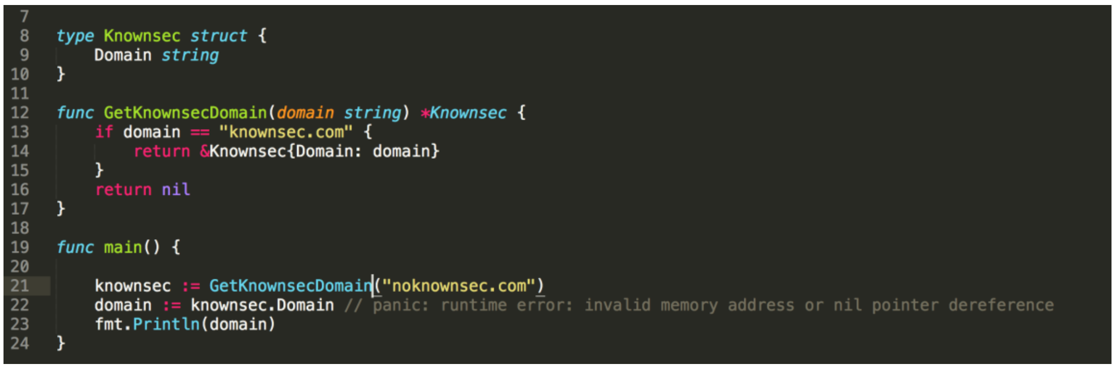
The value of the Go language null pointer is nil. Generally, the nil pointer points to the smallest address of the process, usually this value is 0. Attempting to access data through a null pointer will cause a runtime error. When the program tries to dereference a pointer that is expected to be non-null but is actually null, a null pointer dereference error occurs. Dereferencing a null pointer can cause undefined behavior. On many platforms, dereferencing a null pointer may cause the program to terminate abnormally or denial of service. For example, accessing a null pointer in a Linux system will cause a Segmentation fault error.

5.3.1. Null pointer reference hazards

Null pointer dereference is a common type of memory defect in Go programs. When the pointer points to an invalid memory address and references it, unforeseen errors may occur, causing the software system to crash. Null pointer reference defects may cause system crashes, denial of service and many other serious consequences.

5.3.2. Sample code

The sample code used in this chapter is from example/nil_pointer_dereference.g o



```
 7
 8  type Knownsec struct {
 9      Domain string
10  }
11
12  func GetKnownsecDomain(domain string) *Knownsec {
13      if domain == "knownsec.com" {
14          return &Knownsec{Domain: domain}
15      }
16      return nil
17  }
18
19  func main() {
20
21      knownsec := GetKnownsecDomain("noknownsec.com")
22      domain := knownsec.Domain // panic: runtime error: invalid memory address or nil pointer dereference
23      fmt.Println(domain)
24  }
```

The function GetKnwonsecDomain in the above code returns a pointer variable. If the input parameter doamin is equal to the string "knownsec.com", the Knowsec structure is initialized and returned, otherwise it returns nil. The pointer variable is referenced in line 22, but due to the verification logic Incomplete, the function returns a null pointer when the parameter passed to the function GetKnownsecDomain on line 21 is "noknownsec.com". This leads to the occurrence of null pointer dereference.

5.3.3. How to avoid null pointer references

(1) The pointer needs to be checked for robustness before use, so as to avoid dereferencing the null pointer;

(2) When the return value of the calling function may be null, it is necessary to verify that the return value of the function is not null, so as to avoid null pointer dereference;

(3) Ensure that the exception is handled correctly.

5.4. Ignore the return value

Some functions have return values and the return value is used to judge the behavior of the function execution, such as judging whether the function is executed successfully, so the return value of the function needs to be judged accordingly. Take the strconv.Atoi function as an example, and its prototype is:

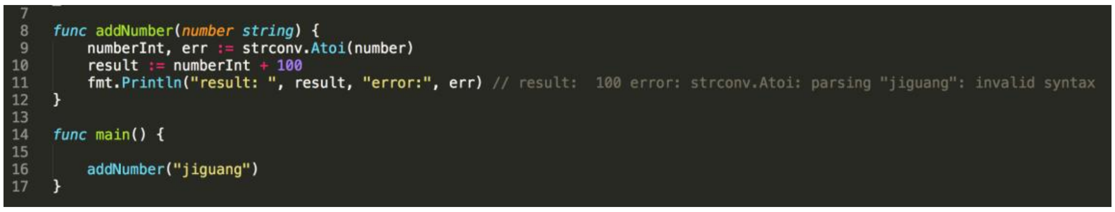
func Atoi(s string) (int, error) If the function executes successfully, it returns the first parameter int; if an error occurs, it returns error. If the function return value is not checked, then when an error occurs in the reading, then It may be that the ignoring of exceptions and error conditions allows the attacker to introduce unexpected behavior.

5.4.1. Ignore the harm of return value

Ignoring the return value of a function can lead to undefined behavior, including information leakage, denial of service, and even program crashes.

5.4.2. Sample code

The sample code used in this chapter is from example/ unchecked_return_ value.go



```
 7
 8  func addNumber(number string) {
 9      numberInt, err := strconv.Atoi(number)
10      result := numberInt + 100
11      fmt.Println("result: ", result, "error:", err) // result:  100 error: strconv.Atoi: parsing "jiguang": invalid syntax
12  }
13
14  func main() {
15
16      addNumber("jiguang")
17  }
```

In the above code, in the 9th line, the incoming string is converted to integer by the strconv.Atoi method, and then the integer is directly operated, ignoring the judgment of the return value err. Therefore, there is the problem of "ignoring the return value".

5.4.3. How to avoid ignoring the return value

There are no related vulnerabilities in the smart contract code.

(1) Properly judge the return value of the function to avoid possible risks when the function is executed abnormally.

(2) Using source code static analysis tools can effectively find such problems.

5.5. Incorrect use of random numbers

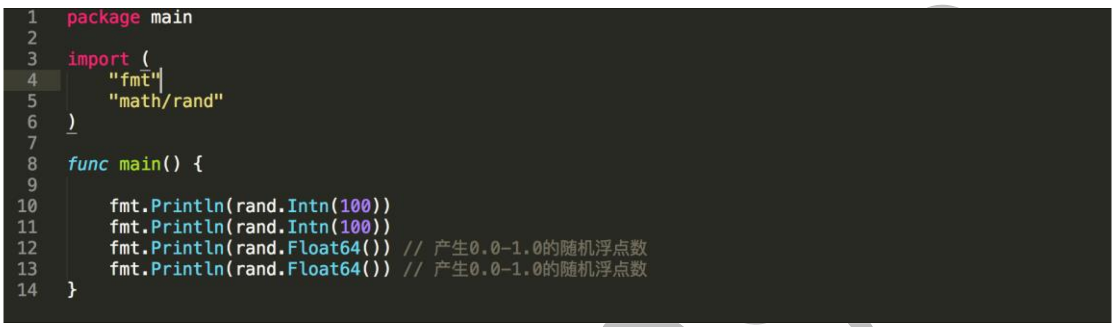
Random numbers are widely used. The most well-known is the application in cryptography. There are many ways to generate random numbers. For example, you can use math/rand to obtain a random number in a Go program. This kind of random number comes from pseudo-random number generation. The output random value can be easily predicted. In environments with high security requirements, such as UUID generation, Token generation, key generation, ciphertext and salt processing. Use a function that can produce a predictable value as a random data source. This predictable value will reduce the security of the system.

5.5.1. Harm of insecure random numbers

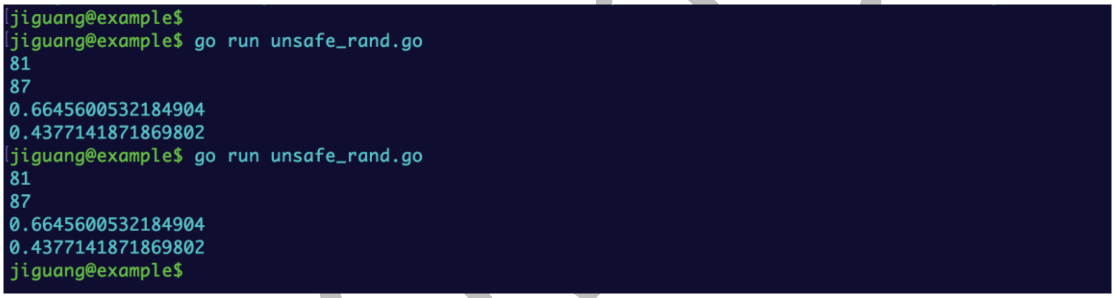
The use of insecure random numbers for encryption operations in encryption functions results in predictable encryption keys. If an attacker can log in to the system, the previous and next encryption keys may be calculated, leading to cracking of the encrypted information.

5.5.2. Sample code

The sample code used in this chapter is from example/ unsafe_rand.go

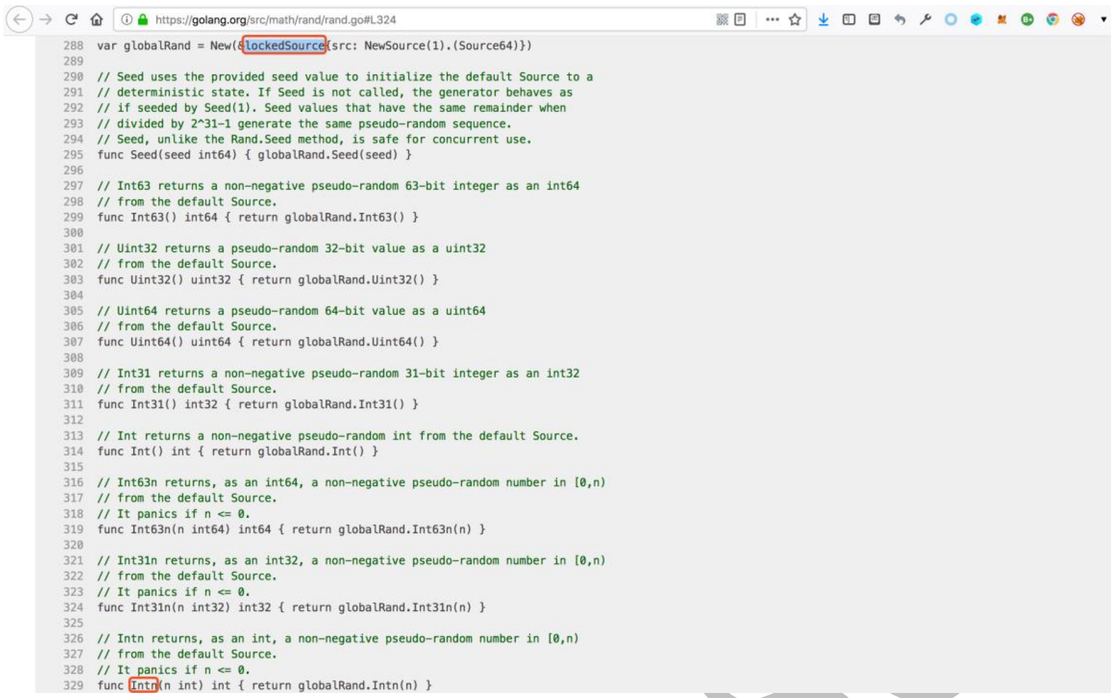


```go
package main

import (
    "fmt"
    "math/rand"
)

func main() {

    fmt.Println(rand.Intn(100))
    fmt.Println(rand.Intn(100))
    fmt.Println(rand.Float64()) // 产生0.0-1.0的随机浮点数
    fmt.Println(rand.Float64()) // 产生0.0-1.0的随机浮点数
}
```



```
jiguang@example$
jiguang@example$ go run unsafe_rand.go
81
87
0.6645600532184904
0.4377141871869802
jiguang@example$ go run unsafe_rand.go
81
87
0.6645600532184904
0.4377141871869802
jiguang@example$
```

Pseudo-random numbers are calculated by a deterministic algorithm from a sequence of uniformly distributed random numbers from [0,1]. It is not truly random, but has statistical characteristics similar to random numbers, such as uniformity and independence. When calculating pseudo-random numbers, if the initial value (seed) used does not change, the “initial value” here is the random seed, and the number sequence of the pseudo-random number is also unchanged. In the above code, the results are the same by comparing the two executions.



```
var globalRand = New(&lockedSource{src: NewSource(1).(Source64)})

// Seed uses the provided seed value to initialize the default Source to a
// deterministic state. If Seed is not called, the generator behaves as
// if seeded by Seed(1). Seed values that have the same remainder when
// divided by 2^31-1 generate the same pseudo-random sequence.
// Seed, unlike the Rand.Seed method, is safe for concurrent use.
func Seed(seed int64) { globalRand.Seed(seed) }

// Int63 returns a non-negative pseudo-random 63-bit integer as an int64
// from the default Source.
func Int63() int64 { return globalRand.Int63() }

// Uint32 returns a pseudo-random 32-bit value as a uint32
// from the default Source.
func Uint32() uint32 { return globalRand.Uint32() }

// Uint64 returns a pseudo-random 64-bit value as a uint64
// from the default Source.
func Uint64() uint64 { return globalRand.Uint64() }

// Int31 returns a non-negative pseudo-random 31-bit integer as an int32
// from the default Source.
func Int31() int32 { return globalRand.Int31() }

// Int returns a non-negative pseudo-random int from the default Source.
func Int() int { return globalRand.Int() }

// Int63n returns, as an int64, a non-negative pseudo-random number in [0,n)
// from the default Source.
// It panics if n <= 0.
func Int63n(n int64) int64 { return globalRand.Int63n(n) }

// Int31n returns, as an int32, a non-negative pseudo-random number in [0,n)
// from the default Source.
// It panics if n <= 0.
func Int31n(n int32) int32 { return globalRand.Int31n(n) }

// Intn returns, as an int, a non-negative pseudo-random number in [0,n)
// from the default Source.
// It panics if n <= 0.
func Intn(n int) int { return globalRand.Intn(n) }
```

By analyzing the source code of rand.Intn(), it can be seen that in the "math/rand" package, if the random seed is not set, the Int() function initializes a locked -Source and generates a pseudo-random number, and the random seed is set to 1. Therefore, no matter how many times the code is executed repeatedly, the random seed is a fixed value each time, and the output pseudo-random number sequence is also fixed. So if the initial value (seed) used by the program can be guessed, then a pseudo-random number of the same number sequence can be generated.

5.5.3. How to avoid insecure random numbers

In applications with high security requirements, a more secure random number generator, such as cryto/rand, should be used.

5.6. Divide by zero

Go has five basic arithmetic operators: addition, subtraction, multiplication, division, and modulo.

The "+" operator adds its operand.

The "-" operator subtracts the second operand from the first operator.

The "*" operator is multiplied by its operand.

The "/" operator divides its first operand by the second.

The "%" operator produces the remainder of dividing the first by the second.

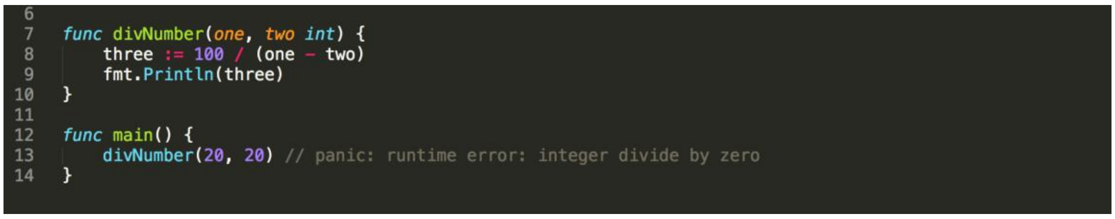
Among them, the second operand of the "/" operation cannot be 0. When the second operand is 0, it will cause a division by zero error.

5.6.1. The hazards of dividing by zero

When a division by zero error occurs, it usually leads to program crashes and denial of service vulnerabilities.

5.6.2. Sample code

The sample code used in this chapter is from example/divide_by_zero.go



```go
func divNumber(one, two int) {
    three := 100 / (one - two)
    fmt.Println(three)
}

func main() {
    divNumber(20, 20) // panic: runtime error: integer divide by zero
}
```

In the 8th line of the above code, the incoming parameters one and two are subtracted and run as a divisor. This input comes from an untrusted source and the value may be 0. When the "/" operation is performed on the 8th line, there is " Divide by zero" problem.

5.6.3. How to avoid dividing by zero

When performing division operations, you need to judge whether the divisor is zero, especially when the divisor comes from untrusted data sources, complex operations, or the return value of a function, you need to pay special attention to whether there is an error in which the divisor is zero.

5.7. Insecure Hash Algorithm

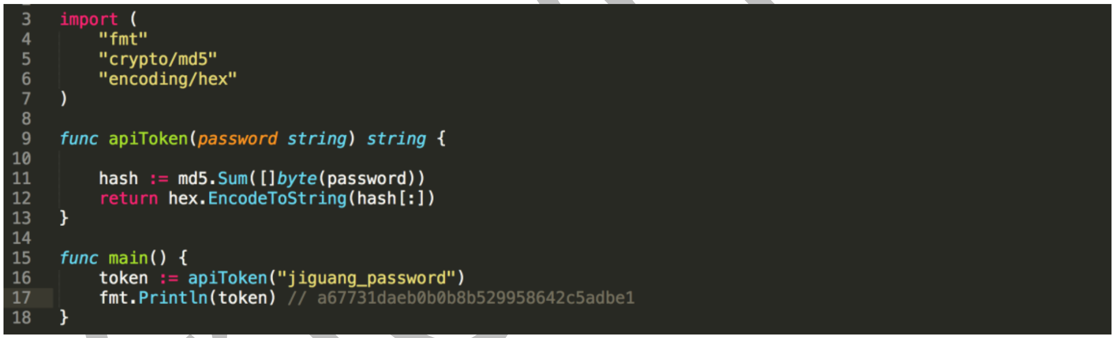
The hash algorithm uses a hash function to map a message of any length into a shorter and fixed-length value, and the mapped value is a hash value. It is a one-way encryption system, that is, an irreversible mapping from plaintext to ciphertext. There is only an encryption process and no decryption process. An insecure hash algorithm can reversely deduce the plaintext. In cryptography, the hash algorithm is mainly used for message digests and signatures to verify the integrity of the entire message, so it is necessary that the hash algorithm cannot derive the original value of the input, which is the basis of the security of the hash algorithm. Currently commonly used hash algorithms include MD4, MD5, SHA, etc. This article uses the Go language source code as an example to analyze the causes of insecure hash algorithm defects and how to fix them. For details, please refer to: CWE ID 327: Use of a Broken or Risky Cryptographic Algorithm (http://cwe.mitre.org/data/definitions/327.html).

5.7.1. Harm of insecure hashing algorithm

Using an insecure hash algorithm to form a digital signature to verify the identity of the data source will affect the integrity and confidentiality of the data and cause the verification method to become invalid.

5.7.2. Sample code

The sample code used in this chapter is from example/unsafe_crypto_md5.go



```
 3  import (
 4      "fmt"
 5      "crypto/md5"
 6      "encoding/hex"
 7  )
 8
 9  func apiToken(password string) string {
10
11      hash := md5.Sum([]byte(password))
12      return hex.EncodeToString(hash[:])
13  }
14
15  func main() {
16      token := apiToken("jiguang_password")
17      fmt.Println(token) // a67731daeb0b0b8b529958642c5adbe1
18  }
```

The above example code operation is the operation of converting the input user password parameter into a hash value and then obtaining the api_token. In line 11, the MD5 converter is used for encoding and hashing. Since MD5 is a recognized hash algorithm that has been cracked, use The hash algorithm to process the data will damage the confidentiality of the data and cause information leakage.

5.7.3. How to avoid insecure hash algorithms

In systems with high security requirements, SHA series algorithms (such as SHA-224, SHA-256, SHA-384, and SHA-512) with a hash value >=224 bits should be used to ensure the integrity of sensitive data.

5.8. Improper use of function addresses

Incorrectly using function addresses as functions, conditional expressions, operation objects, or even participating in logical operations, will cause various unexpected program behaviors to occur. For example, in the following if statement, where func() is a function defined in the program:

```
if (func == nil)
```

Since func is used instead of func(), that is, the address of func is used instead of the return value of the function, and the address of the function is not equal to nil, if the function address is compared with nil, the condition will be judged as false. .

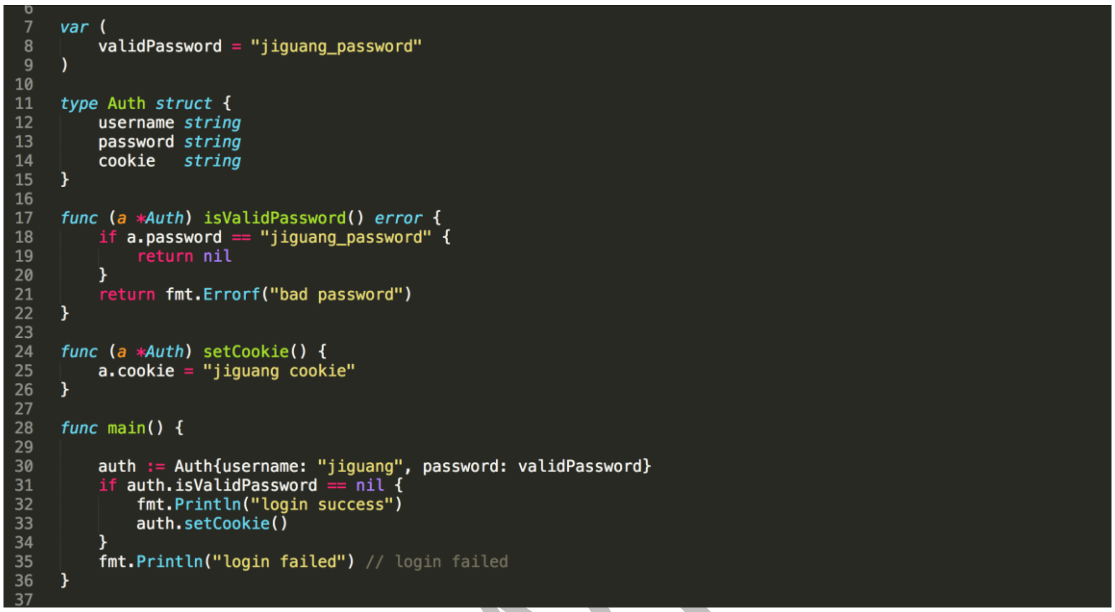
For details, please refer to CWE-480: Use of IncorrectOperator.

5.8.1. The hazards of improper use of function addresses

Improper use of function addresses may cause unexpected program behaviors, such as logic errors that occur when conditions are never triggered, or infinite loops due to conditions that are always true, resulting in resource exhaustion, denial of service, etc. attack.

5.8.2. Sample code

The sample code used in this chapter is from example/use_of_incorrect_operator_basic.go



```
 6
 7  var (
 8      validPassword = "jiguang_password"
 9  )
10
11  type Auth struct {
12      username string
13      password string
14      cookie   string
15  }
16
17  func (a *Auth) isValidPassword() error {
18      if a.password == "jiguang_password" {
19          return nil
20      }
21      return fmt.Errorf("bad password")
22  }
23
24  func (a *Auth) setCookie() {
25      a.cookie = "jiguang cookie"
26  }
27
28  func main() {
29
30      auth := Auth{username: "jiguang", password: validPassword}
31      if auth.isValidPassword == nil {
32          fmt.Println("login success")
33          auth.setCookie()
34      }
35      fmt.Println("login failed") // login failed
36  }
37
```

In the above sample code, the 26th line uses auth.isValidPassword == nil as the judgment condition of the if statement. The isValidPassword() function is defined in the 17th line. The auth.isValidPassword == nil operation causes the if statement to always be false. The 33rd The setCookie() function will never be executed, and there is a problem of "improper use of function address".

5.8.3. How to avoid improper use of function addresses

It is necessary to clarify whether the operation uses the function address or the function return value to avoid the direct use of the function address due to coding errors.

5.9. Double check lock

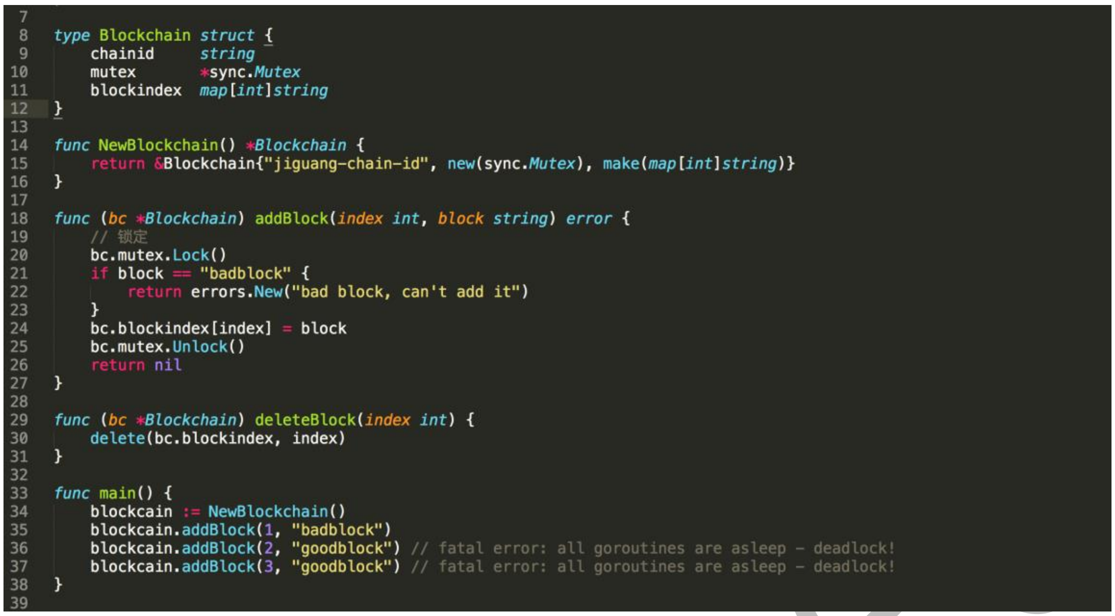
In program development, sometimes it is necessary to postpone some expensive object initialization operations, and initialize them only when these objects are used. At this time, double check locking can be used to delay the object initialization operations. Double check locking is a software design pattern designed to reduce competition and synchronization overhead in concurrent systems. Based on the common singleton pattern, first determine whether the object has been initialized, and then decide whether to lock. Although double-checked locking solves the common singleton model's error-prone and thread-unsafe problems in a multi-threaded environment, there are still some hidden dangers. See CWE ID 609: Double-Checked Locking (http://cwe.mitre.org/data/definitions/609.html) for details.

5.9.1. The hazards of double check lockout

Double check locking has no effect in a single-threaded environment. In a multi-threaded environment, because threads will switch to each other at any time, in the case of instruction rearrangement, the object is not instantiated completely, resulting in program call errors.

5.9.2. Sample code

The sample code used in this chapter is from example/ double_checked_ locking.go



```go
7
8   type Blockchain struct {
9       chainid     string
10      mutex       *sync.Mutex
11      blockindex  map[int]string
12  }
13
14  func NewBlockchain() *Blockchain {
15      return &Blockchain{"jiguang-chain-id", new(sync.Mutex), make(map[int]string)}
16  }
17
18  func (bc *Blockchain) addBlock(index int, block string) error {
19      // 锁定
20      bc.mutex.Lock()
21      if block == "badblock" {
22          return errors.New("bad block, can't add it")
23      }
24      bc.blockindex[index] = block
25      bc.mutex.Unlock()
26      return nil
27  }
28
29  func (bc *Blockchain) deleteBlock(index int) {
30      delete(bc.blockindex, index)
31  }
32
33  func main() {
34      blockcain := NewBlockchain()
35      blockcain.addBlock(1, "badblock")
36      blockcain.addBlock(2, "goodblock") // fatal error: all goroutines are asleep - deadlock!
37      blockcain.addBlock(3, "goodblock") // fatal error: all goroutines are asleep - deadlock!
38  }
39
```

In the above code, add the 21st line of the block function addBlock, first judge the block, if the condition is met, then return to the error prompt operation. At this time, the global lock bc.mutex.Unlock() cannot be executed, and it will be called next time In the addBlock function, the global lock is locked again on line 20, but at this time the global lock is already a locking operation, so the function cannot be executed correctly, so an error occurs.

5.9.3. How to avoid double-checked locking

To avoid double check lock, you need to use defer after using the lock operation to ensure that the lock can be released and prevent double check lock errors from occurring.

5.10. Concurrency security risks

The most important feature of the go language is its native support for concurrency-goroutine. When concurrency is used, data security has to be considered.

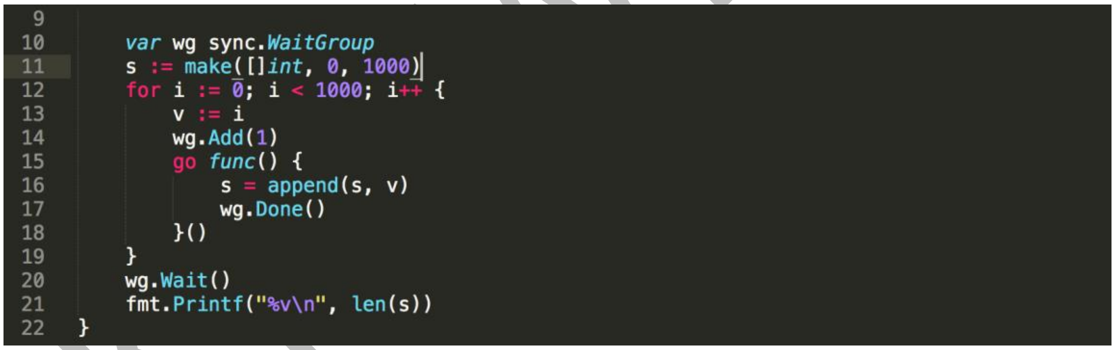
In many cases, the compiler will make some magical optimizations, resulting in unexpected data conflicts. Therefore, as long as it meets the condition that "there are multiple threads accessing the same memory at the same time, and at least one of the threads is a write operation", Need to deal with concurrency security.

5.10.1.The harm of concurrent security risks

When multiple threads share data, results that do not match expectations may occur, which may cause problems such as data loss and value overwriting.

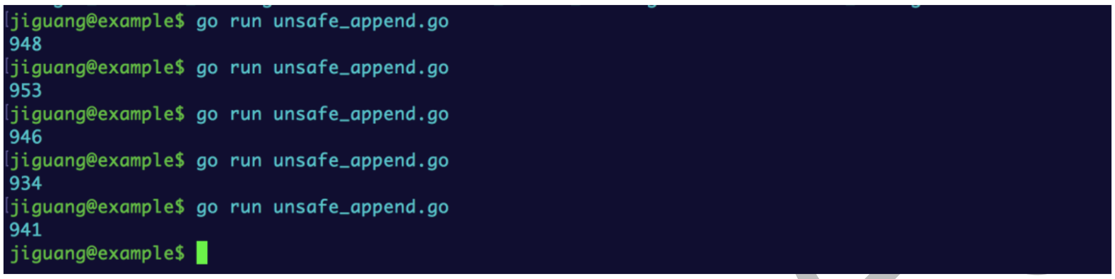
5.10.2.Sample code

The sample code used in this chapter is from example/unsafe_append.go



```go

    var wg sync.WaitGroup
    s := make([]int, 0, 1000)
    for i := 0; i < 1000; i++ {
        v := i
        wg.Add(1)
        go func() {
            s = append(s, v)
            wg.Done()
        }()
    }
    wg.Wait()
    fmt.Printf("%v\n", len(s))
}
```

The above code operates on slices through append. Since slice is a reference type, even if the function is called by value, the parameter copy still points to the mapping slice s, so n goroutines write the same mapping slice s concurrently. There will be competition for shared variables, resources, and concurrent reads and writes. Therefore, shared resources suffer To destroy, so you can lock the operation, or use the channel to queue up serialization. For example, when the two coroutines A and B run append and find that the position of s[1] is empty, they will put their own value in this position, so that their two values will be overwritten, resulting in data Lost. After running the test a few times, you will find that the slice length is not 1000, but is constantly changing.



```
jiguang@example$ go run unsafe_append.go
948
jiguang@example$ go run unsafe_append.go
953
jiguang@example$ go run unsafe_append.go
946
jiguang@example$ go run unsafe_append.go
934
jiguang@example$ go run unsafe_append.go
941
jiguang@example$
```

5.10.3.How to avoid concurrency security risks

When there may be the following situation: multiple threads access the same memory at the same time, and at least one of the threads is a write operation.

If the above conditions are met, it should be locked decisively. The lock operation is in the order of tens of nanoseconds, and the overhead is basically negligible.

5.11. Contaminated memory allocation

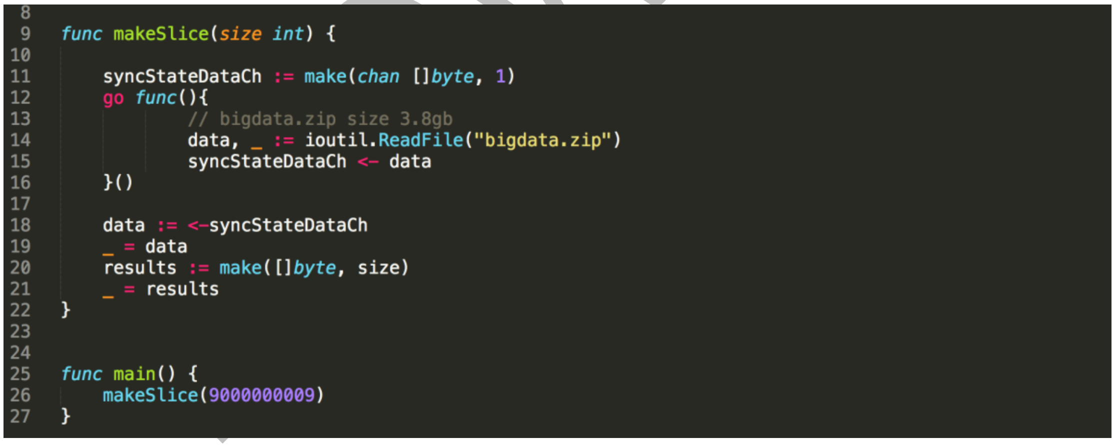
For every mobile developer, memory is a resource that needs to be used carefully. OOM (OutOfMemoryError) is very easy to occur due to careless memory management. Memory leaks will eventually lead to memory overflow. Because the memory in the system is limited, if you overuse resources If it is not released in time, it will eventually lead to insufficient memory, which will not provide enough memory for the data that needs to be stored, resulting in memory overflow. The memory overflow may also be due to the fact that the size of the data is not allocated according to the actual requirements, and finally the allocated memory cannot meet the needs of the data, resulting in memory overflow.

5.11.1.The hazards of contaminated memory allocation

Directly use the polluted data as the length parameter of the memory allocation function. If a very large integer value is passed in, the program will allocate a huge amount of memory accordingly, resulting in huge memory overhead of the system and even denial of service attacks.

5.11.2.Sample code

The sample code used in this chapter is from example/out_of_memory.go



```go

func makeSlice(size int) {

    syncStateDataCh := make(chan []byte, 1)
    go func(){
            // bigdata.zip size 3.8gb
            data, _ := ioutil.ReadFile("bigdata.zip")
            syncStateDataCh <- data
    }()

    data := <-syncStateDataCh
    _ = data
    results := make([]byte, size)
    _ = results
}


func main() {
    makeSlice(9000000009)
}
```

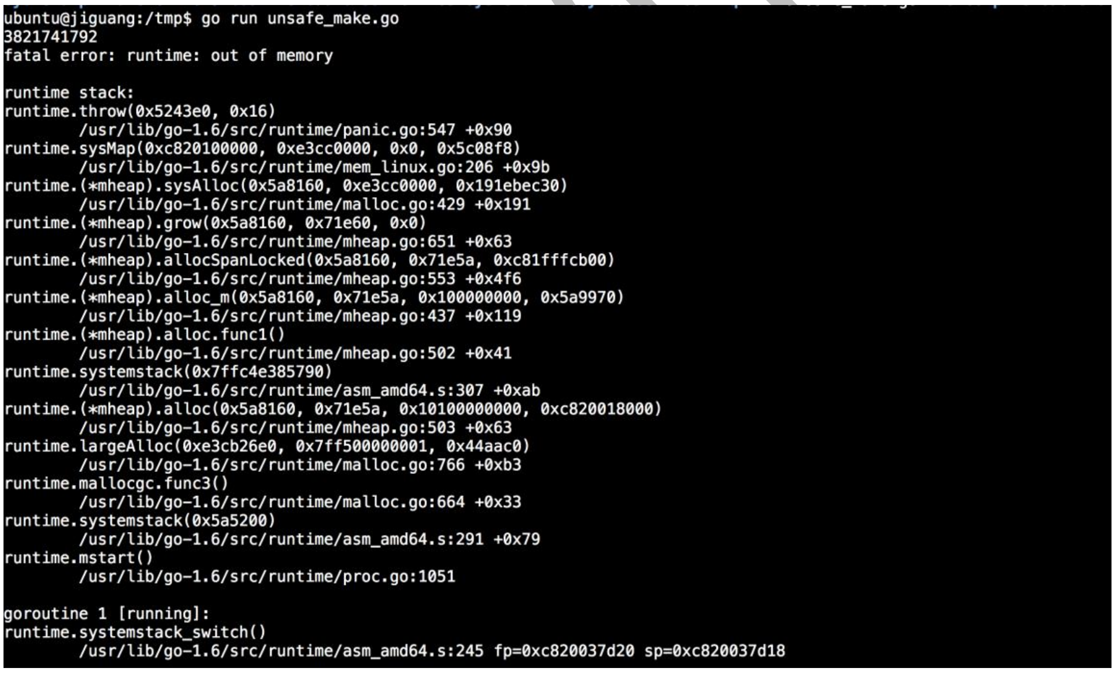
In the above code, the makeSlice function uses the make function to initialize the slice results for the incoming parameter size. The built-in function make is used to allocate memory and initialize an object for slice, map or chan types.

Slice: The second parameter size specifies its length, and its capacity and length are the same. You can pass in the third parameter to specify a different capacity value, but it must not be smaller than the length value. For example, make([]int, 0, 10)

Map: Initialize the allocation of memory according to the size, but the length of the map after allocation is 0, if the size is omitted, then a small size of memory will be allocated during the initial allocation of memory

Channel: The pipeline buffer is initialized according to the buffer capacity. If the capacity is 0 or the capacity is ignored, the pipeline has no buffer.

If you can control the size of the size, the built-in function make is likely to cause memory overflow during memory allocation, leading to out-of-memory.



```
ubuntu@jiguang:/tmp$ go run unsafe_make.go
3821741792
fatal error: runtime: out of memory

runtime stack:
runtime.throw(0x5243e0, 0x16)
        /usr/lib/go-1.6/src/runtime/panic.go:547 +0x90
runtime.sysMap(0xc820100000, 0xe3cc0000, 0x0, 0x5c08f8)
        /usr/lib/go-1.6/src/runtime/mem_linux.go:206 +0x9b
runtime.(*mheap).sysAlloc(0x5a8160, 0xe3cc0000, 0x191ebec30)
        /usr/lib/go-1.6/src/runtime/malloc.go:429 +0x191
runtime.(*mheap).grow(0x5a8160, 0x71e60, 0x0)
        /usr/lib/go-1.6/src/runtime/mheap.go:651 +0x63
runtime.(*mheap).allocSpanLocked(0x5a8160, 0x71e5a, 0xc81fffcb00)
        /usr/lib/go-1.6/src/runtime/mheap.go:553 +0x4f6
runtime.(*mheap).alloc_m(0x5a8160, 0x71e5a, 0x100000000, 0x5a9970)
        /usr/lib/go-1.6/src/runtime/mheap.go:437 +0x119
runtime.(*mheap).alloc.func1()
        /usr/lib/go-1.6/src/runtime/mheap.go:502 +0x41
runtime.systemstack(0x7ffc4e385790)
        /usr/lib/go-1.6/src/runtime/asm_amd64.s:307 +0xab
runtime.(*mheap).alloc(0x5a8160, 0x71e5a, 0x10100000000, 0xc820018000)
        /usr/lib/go-1.6/src/runtime/mheap.go:503 +0x63
runtime.largeAlloc(0xe3cb26e0, 0x7ff500000001, 0x44aac0)
        /usr/lib/go-1.6/src/runtime/malloc.go:766 +0xb3
runtime.mallocgc.func3()
        /usr/lib/go-1.6/src/runtime/malloc.go:664 +0x33
runtime.systemstack(0x5a5200)
        /usr/lib/go-1.6/src/runtime/asm_amd64.s:291 +0x79
runtime.mstart()
        /usr/lib/go-1.6/src/runtime/proc.go:1051

goroutine 1 [running]:
runtime.systemstack_switch()
        /usr/lib/go-1.6/src/runtime/asm_amd64.s:245 fp=0xc820037d20 sp=0xc820037d18
```

5.11.3. How to avoid tainted memory allocation

(1) Avoid using contaminated data directly as the length parameter of the memory allocation function. If it cannot be avoided, the contaminated data should be effectively restricted.

(2) Using source code static analysis tools can effectively find such problems.

5.12. Unsigned integer wraparound

There are problems caused by improper use of unsigned integers in Go. The following table lists the unsigned integer types and ranges defined by the ANSI standard

Type	Number of digits	Value range
uint8	8	0 to 255
uint16	16	0 to 65535
uint32	32	0 to 4294967295
uint64	64	0 to 18446744073709552000

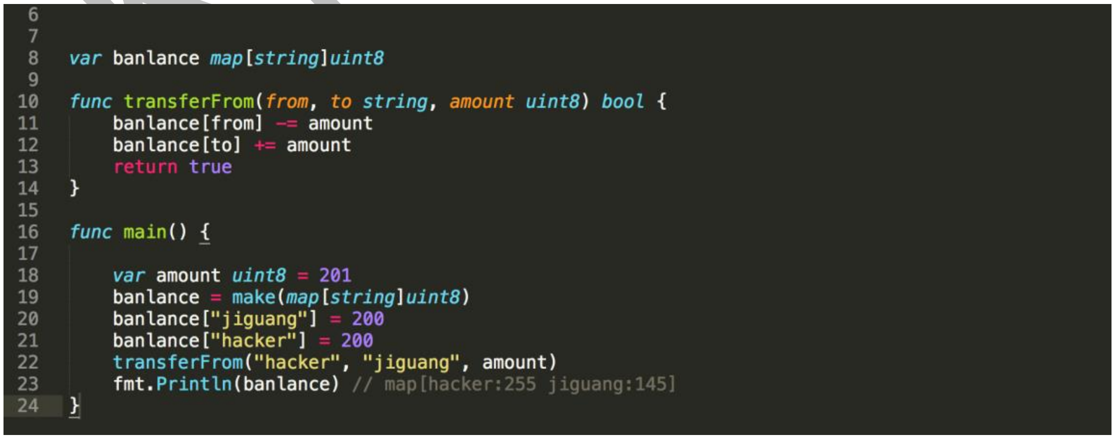
Calculations involving unsigned integers will not overflow, but wrap around when the value exceeds the value range of unsigned integers. Such as: the maximum value of an unsigned integer plus 1 will return 0, and the minimum value of an unsigned integer minus 1 will return the maximum value of the type. Operators that cause unsigned integer operations to wrap around are "+", "-", "*", "++", "–", "+=", "-=", "*=", "<<= ", "<<", etc.

5.12.1.The harm of unsigned integer wraparound

Through the analysis of the unsigned integer wraparound principle, the most direct result of unsigned integer wraparound is a numerical error, and the calculated value does not meet the expectations of the program. When the unsigned integer wraps around to produce a maximum value, if the data is used in memory copy functions such as []byte(string), string([]byte), a huge amount of data will be copied, which may cause errors or destroy the stack. In addition, one of the most likely uses of unsigned integer wraparound is for memory allocation. For example, when using the make() function for memory allocation, when the make() function wraps around, it may be 0 or It is a maximum value, which leads to 0-length memory allocation or memory allocation failure.

5.12.2.Sample code

The sample code used in this chapter is from example/ integer_overflow_uint8.go



```go


var banlance map[string]uint8

func transferFrom(from, to string, amount uint8) bool {
    banlance[from] -= amount
    banlance[to] += amount
    return true
}

func main() {

    var amount uint8 = 201
    banlance = make(map[string]uint8)
    banlance["jiguang"] = 200
    banlance["hacker"] = 200
    transferFrom("hacker", "jiguang", amount)
    fmt.Println(banlance) // map[hacker:255 jiguang:145]
}
```

The transferFrom function in the above code is a transfer operation, and the transfer fund type is uint8. The program does not limit the value of amount. Performing operations on lines 11 and 12 will cause unsigned integer wraparound problems. After the final transfer, the hacker's account balance becomes the maximum.

5.12.3. How to avoid unsigned integer wrapping

(1) When the parameter type of the function is an unsigned integer, the value of the passed parameter needs to be effectively judged to avoid wrapping directly or after calculation;

(2) Data from untrusted sources still requires special attention, and data from untrusted sources should be filtered and restricted;

(3) Using source code static analysis tools for automated detection can effectively find unsigned integer wraparound problems in the source code.

5.13. Path traversal

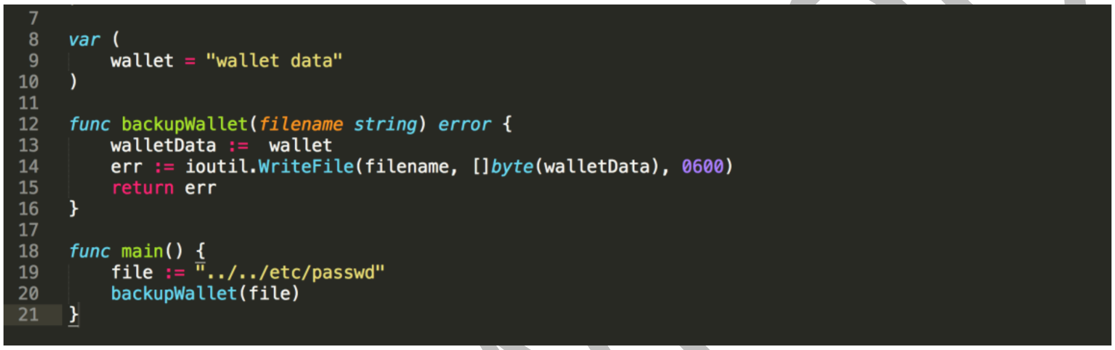
Path traversal means that the application program receives user parameters that have not been properly verified to perform operations related to file reading and viewing. This parameter contains special characters (such as ".." and "/"). Special characters like special characters can get rid of the restrictions of protection, unauthorized access to some protected files, directories, or overwrite sensitive data.

5.13.1.Harm of path traversal

Path traversal uses application special symbols ("~/", "../") to perform directory backtracking, allowing attackers to unauthorized access or overwrite sensitive data, such as website configuration files, system core files, etc.

5.13.2.Sample code

The sample code used in this chapter is from example/ path_traversal.go.



```
 7
 8  var (
 9      wallet = "wallet data"
10  )
11
12  func backupWallet(filename string) error {
13      walletData :=  wallet
14      err := ioutil.WriteFile(filename, []byte(walletData), 0600)
15      return err
16  }
17
18  func main() {
19      file := "../../etc/passwd"
20      backupWallet(file)
21  }
```

The backupWallet function in the above code is a wallet backup operation. The parameter filename is not checked in line 14 and directly brought into the participating function to run. When the filename parameter can be controlled, it can cause any file overwriting risk.

5.13.3.How to avoid path traversal

(1) The program purifies the input data of untrusted users, hard-codes or uniformly codes the file names submitted by website users, and filters illegal characters.

(2) Whitelist control of file suffixes, and reject malicious symbols or empty bytes.

(3) Reasonably configure the directory permissions of the web server.

5.14. Log processing

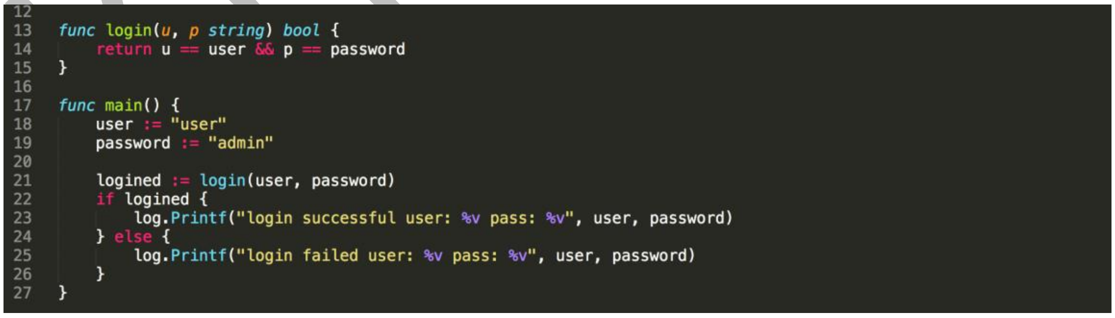
Logging should always be handled by the application and should not rely on server configuration. All logging should be implemented by the main routine on the trusted system. Developers should also ensure that the logs do not contain sensitive data (such as passwords, session information, system details, etc.), and there is no debugging and stack trace information. In addition, the log records should include successful and failed security events. The focus is important Log event data.

5.14.1. The hazards of log error handling

Disclosure of sensitive data such as user passwords and session information.

5.14.2. Sample code

The sample code used in this chapter is from example/ unsafe_log.go



```
12
13  func login(u, p string) bool {
14      return u == user && p == password
15  }
16
17  func main() {
18      user := "user"
19      password := "admin"
20
21      logined := login(user, password)
22      if logined {
23          log.Printf("login successful user: %v pass: %v", user, password)
24      } else {
25          log.Printf("login failed user: %v pass: %v", user, password)
26      }
27  }
```

The above code uses the login function to determine whether the current user password is correct, and the current user account password information is incorrectly printed on lines 23 and 25. It is a log error handling problem.

5.14.3.How to avoid log error handling

(1) Developers should also ensure that the logs do not contain sensitive data (such as passwords, session information, system details, etc.), nor any debugging and stack trace information.

6. Appendix B：Vulnerability rating standard

Blockchain system vulnerability risk rating standard	
Vulnerability rating	**Vulnerability rating description**
Serious vulnerabilities	Forgery of computing power, centralization of computing power, severe cases can lead to 51% attacks; Vulnerabilities can lead to the disclosure of private keys and affect asset security; The public chain can be bifurcated by certain means; Occupy local storage resources DoS nodes. In severe cases, it can cause node program crash, forge blocks, and generate unexpected benefits. In severe cases, it can cause chain bifurcation, data tampering and other problems; Same-chain or cross-chain transaction replay, forgery of legal transactions, affecting asset security, double spending of assets in the same chain, transaction replay, smart bypass of sandbox restrictions, severe cases can affect the execution logic of other contracts, or cause DoS and remote Order execution Unexpected behaviors in contract execution, severe cases can lead to data tampering or node program crash, forged identities to bypass permission checks, severe cases can threaten asset security; At the current stage, there is a large-scale outbreak that should attract enough attention to security vulnerabilities and so on.
High-risk vulnerabilities	The business operation can be seriously affected by certain means, which may cause huge losses to customers, such as key business RPC login interface can be bumped into the database, business system can be easily smashed into the wool, etc.;

	Vulnerabilities that require user interaction to obtain user identity information, read, modify, overwrite, delete, and download arbitrary files; Bypass restrictions to modify user personal data, personal information, and force users to perform certain operations; Serious information leakage that can cause harm to the target system, sensitive information file backup or source code leakage, etc. (such as storage key leakage, database connection password leakage, SVN/Git account leakage, VPN account leakage, etc.). Vulnerabilities that can directly cause the loss of token contracts or user funds, such as: a value overflow vulnerability that can cause the value of tokens to zero, a false recharge vulnerability that can cause exchanges to lose tokens, and can cause contract accounts to lose ETH or tokens. Access loopholes, etc.; Vulnerabilities that can cause the loss of ownership of token contracts, such as: access control defects of key functions, call injection leading to key function access control bypassing, etc.; Vulnerabilities that can cause the token contract to not work properly, such as: denial of service vulnerability caused by sending ETH to malicious addresses, and denial of service vulnerability caused by exhaustion of gas.
Medium-Dangerous Vulnerability	Can affect the security of the target system but cannot directly prove that it can be exploited; Able to obtain the permission of the target network device but it is determined that it cannot be further used; Non-important information leakage, CORS vulnerabilities in non-core critical business operations, etc. High-risk vulnerabilities that require specific addresses to trigger, such as value overflow vulnerabilities that can only be triggered by token contract owners;

	access control flaws in non-critical functions, and logic design flaws that cannot cause direct capital losses, etc.
Low-risk vulnerabilities	It is impossible to determine whether it can affect the security of the target system. For example, the management port of the target system is open to the public network but cannot be used directly, the banner information of the target system can be identified, and other penetration testers determine that it is difficult to use but may have potential Vulnerabilities of security threats. Vulnerabilities that are difficult to trigger, vulnerabilities with limited damage after triggering, such as numerical overflow vulnerabilities that require a large amount of ETH or tokens to trigger, vulnerabilities where attackers cannot directly profit after triggering numerical overflow, and the transaction sequence triggered by specifying high gas depends on the risk Wait.

7. Appendix C: Blockchain system risk rating standards

Public chain system risk rating standard (the risk level of alliance chain and private chain is downgraded by one level)	
System risk rating	Vulnerability rating description
Severe risk system	A system with 1 or more serious vulnerabilities, or 2 or more high-risk vulnerabilities
High risk system	A system with 1 or more high-risk vulnerabilities, or more than 3 medium-risk vulnerabilities
Medium risk system	A system with 1 or more medium-risk vulnerabilities, or 5 or more low-risk vulnerabilities
Low risk system	Systems with 3~5 low-risk vulnerabilities
Security system	There are less than 3 low-risk vulnerabilities, or systems with no vulnerabilities

8. Appendix D: Introduction to Vulnerability Testing Tools

8.1. Gosec

Gosec scans the Go AST to check the source code for security issues, scans for hard-coded credentials, does not check for audit errors, uses unescaped data in HTML templates, uses predictable paths to create temporary files, and extracts zip archives. Security issues such as file traversal can be configured to run only a subset of rules, exclude certain file paths, and generate reports in different formats.

8.2. Snyk

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

8.3. Flawfinder

Flawfinder is an open source C/C++ static scanning analysis tool. It performs a static search based on an internal dictionary database to match simple defects and vulnerabilities. The flawfinder tool does not need to compile C/C++ code and can directly scan and analyze. Simple and fast, the biggest advantage is that it is free and does not need to be compiled.

8.4. SonarQube

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

8.5. Yasca

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

8.6. Safesql

Golang's static analysis tool can detect database security issues such as SQL injection.

8.7. CFSSL

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

8.8. Sublime Text-IDE

Sublime Text supports syntax highlighting of multiple programming languages, has excellent code auto-completion function, and also has the function of code snippets, which can save commonly used code snippets and call them at any time when needed. Support VIM mode, you can use most commands in Vim mode. Macros are supported. Simply put, record the operation or write the command yourself, and then play the operation or command just recorded.

8.9. KNOWNSEC special toolkit for penetration testers

KNOWNSEC Penetration Tester's special toolkit is developed, collected and used by KNOWNSEC penetration test engineers. It contains batch automatic test tools dedicated to testers, self-developed tools, scripts or utilization tools, etc.

KNOWNSEC
Blockchain Lab

Official Website
www.knownseclab.com

E-mail
blockchain@knownsec.com

WeChat Official Account