AnySwap 跨链桥白盒安全审计报告

目录

1 前言

慢雾安全团队于 2020 年 08 月 10 日，收到 AnySwap 团队对 AnySwap 跨链桥项目安全审计的申请，根据项目特点慢雾安全团队制定如下审计方案。

慢雾安全团队将采用“白盒为主，黑灰为辅”的策略，以最贴近真实攻击的方式，对项目进行安全审计。

慢雾科技测试方法：

黑盒测试	站在外部从攻击者角度进行安全测试。
灰盒测试	通过脚本工具对代码模块进行安全测试，观察内部运行状态，挖掘弱点。
白盒测试	基于项目的源代码，进行脆弱性分析和漏洞挖掘。

慢雾科技漏洞风险等级：

严重漏洞	严重漏洞会对项目的安全造成重大影响，强烈建议修复严重漏洞。
高危漏洞	高危漏洞会影响项目的正常运行，强烈建议修复高危漏洞。
中危漏洞	中危漏洞会影响项目的运行，建议修复中危漏洞。
低危漏洞	低危漏洞可能在特定场景中会影响项目的业务操作，建议项目方自行评估和考虑这些问题是否需要修复。
弱点	理论上存在安全隐患，但工程上极难复现。
增强建议	编码或架构存在更好的实践方法。

慢雾安全团队白盒安全审计流程包含两个步骤:

- 使用开源或内部自动化分析的工具对代码中常见的安全漏洞进行扫描和测试。
- 人工审计代码的安全问题，通过人工分析代码，发现代码中潜在的安全问题。

如下是白盒审计过程中慢雾安全团队会重点审查的审计项列表:

（其他未知安全漏洞不包含在本次审计责任范围）

- 通讯安全
- 组件安全
- 多线程安全
- 密码学安全
- RPC 接口安全
- nil 指针安全
- 业务逻辑漏洞
- 拒绝服务漏洞
- 敏感信息泄漏
- 日志处理安全
- 缓存信息安全
- 访问控制策略
- 配置文件安全
- 异常未捕获处理
- 递归和循环安全
- 数据库操作安全

2 项目背景

2.1 项目简介

AnySwap CrossChain-Bridge 通过在目的链上创建影射资产，并维护换进换出操作时源链上已置换资产和目的链上对应影射资产的数额一致性。通过 dcrm 安全多签方法实现多方共同管理资产和影射资产的换进换出置换操作。

本次审计针对 AnySwap CrossChain-Bridge 代码进行安全审计，不包含 dcrm 模块。

如下是相关的文件信息:

1. 审计版本文件信息

https://github.com/fsn-dev/crossChain-Bridge

commit: 590ca12ed7fe5e767f6b7fa04e46df9e870de34d

2. 修复版本文件信息

https://github.com/fsn-dev/crossChain-Bridge
commit: 69b78a5f6f7ac0c36a3704dfcd7a52d089ac2e31

3. 需求设计文档

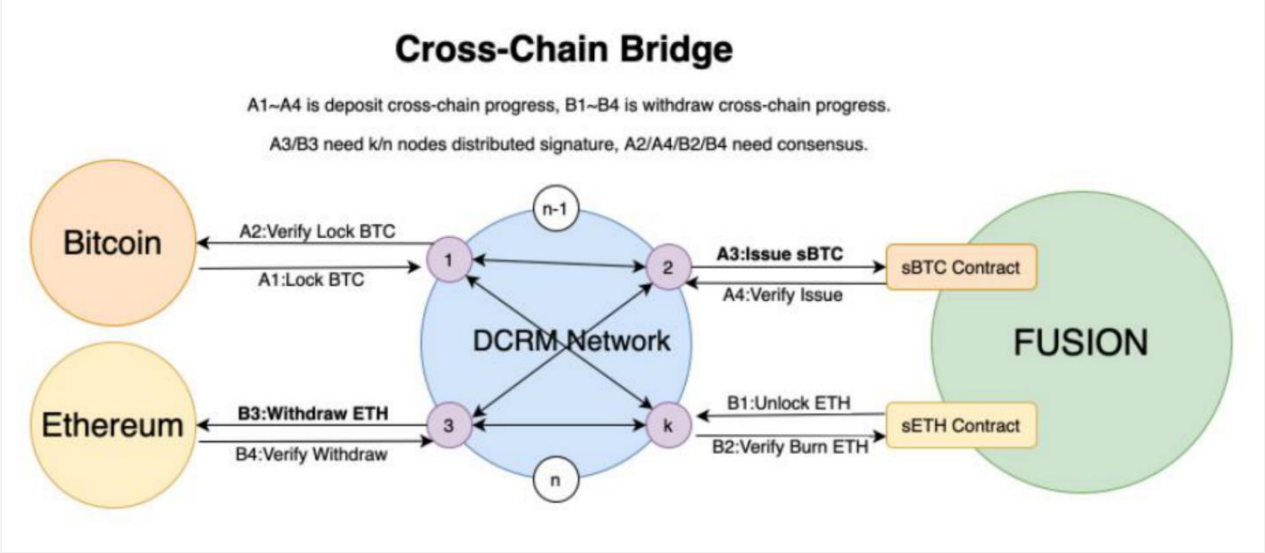
https://github.com/fsn-dev/crossChain-Bridge/wiki/CrossChain-Bridge-Design#%E6%96%B9%E6%A1%88%E6%A6%82%E8%BF%B0

4. 源码目录说明

crosschain-bridge 源码目录说明.txt
MD5: 31e2cfafdd28f6fc7bf9cba1b88a06e9

2.2 项目架构

项目架构图



3 审计结果

3.1 通讯安全

RPC 和 API 等服务的通讯需要在 build/bin/config.toml 进行配置，这里要注意与 RPC 之间通信的安全，建

议采用 https 进行通信。

- build/bin/config.toml

```
# risk control config
InitialDiffValue = 81000.0
MaxAuditDiffValue = 100.0
MinWithdrawReserve = 10000.0
 [Email]
Server = "smtp.gmail.com"
Port = 25
From = "from@gmail.com"
FromName = "Risk Control"
Password = "***"
To = ["to1@gmail.com", "to2@gmail.com"]
Cc = ["cc1@gmail.com", "cc2@gmail.com"]

# source token config[SrcToken]
BlockChain = "Ethereum"
NetID = "Rinkeby"
ID = "ERC20"
Name = "USDTERC20"
Symbol = "USDT"
Decimals = 6
Description = "ERC20 USDT"
ContractAddress = "0xb09bad01684f6d47fc7dc9591889cc77eaed8d22"
DepositAddress = "0x1249E24A077A1d95254DB00ba406100fBca8003F"
DcrmAddress = "0xb09bad01684f6d47fc7dc9591889cc77eaed8d22"

# source blockchain gateway config[SrcGateway]
APIAddress = ["http://5.189.139.168:8018"]

# dest token config[DestToken]
BlockChain = "Fusion"
NetID = "Testnet"
ID = "mUSDT"
Name = "SMPC ERC20 USDT"
Symbol = "mUSDT"
Decimals = 6
```

```
Description = "cross chain bridge USDT with mUSDT"
ContractAddress = "0x13edb3a93a985d5db4b4965d38c446ed7a931c53"
DcrmAddress = "0x826197183eB1a7D71DB073216ce157886680B094"

# dest blockchain gateway config[DestGateway]
APIAddress = ["https://testnet.fsn.dev/api"]
```

RPC 服务没有使用 https 进行通讯，建议可以使用 Nginx 进行代理转发，并使用 https 进行通讯。

- rpc/server/server.go

```
func StartAPIServer() {
	router := initRouter()

	apiPort := params.GetAPIPort()
	apiServer := params.GetConfig().APIServer
	allowedOrigins := apiServer.AllowedOrigins

	corsOptions := []handlers.CORSOption{
		handlers.AllowedMethods([]string{"GET", "POST"}),
	}
	if len(allowedOrigins) != 0 {
		corsOptions = append(corsOptions,
			handlers.AllowedHeaders([]string{"X-Requested-With", "Content-Type"}),
			handlers.AllowedOrigins(allowedOrigins),
		)
	}

	log.Info("JSON RPC service listen and serving", "port", apiPort, "allowedOrigins", allowedOrigins)
	svr := http.Server{
		Addr:         fmt.Sprintf(":%v", apiPort),
		ReadTimeout:  60 * time.Second,
		WriteTimeout: 60 * time.Second,
		Handler:      handlers.CORS(corsOptions...)(router),
	}
	go func() {
		if err := svr.ListenAndServe(); err != nil {
			log.Error("ListenAndServe error", "err", err)
		}
```

```
        }0}
```

3.2 组件安全

对项目的依赖模块进行审计，未发现使用存在已知漏洞的组件。

- go.mod

```
module github.com/anyswap/CrossChain-Bridge

go 1.14

require (
        github.com/BurntSushi/toml v0.3.1
        github.com/btcsuite/btcd v0.20.1-beta
        github.com/btcsuite/btcutil v1.0.2
        github.com/btcsuite/btcwallet/wallet/txauthor v1.0.0
        github.com/btcsuite/btcwallet/wallet/txrules v1.0.0
        github.com/btcsuite/btcwallet/wallet/txsizes v1.0.0
        github.com/dvyukov/go-fuzz v0.0.0-20200805095026-2dc2d88eb660 // indirect
        github.com/ethereum/go-ethereum v1.9.18
        github.com/fastly/go-utils v0.0.0-20180712184237-d95a45783239 // indirect
        github.com/gorilla/handlers v1.4.2
        github.com/gorilla/mux v1.7.4
        github.com/gorilla/rpc v1.2.0
        github.com/jehiah/go-strftime v0.0.0-20171201141054-1d33003b3869 // indirect
        github.com/jonboulle/clockwork v0.2.0 // indirect
        github.com/lestrrat-go/file-rotatelogs v2.3.0+incompatible
        github.com/lestrrat-go/strftime v1.0.1 // indirect
        github.com/pborman/uuid v1.2.0
        github.com/pkg/errors v0.9.1 // indirect
        github.com/sirupsen/logrus v1.5.0
        github.com/stretchr/testify v1.5.1
        github.com/tebeka/strftime v0.1.5 // indirect
        github.com/urfave/cli/v2 v2.2.0
        golang.org/x/crypto v0.0.0-20200622213623-75b288015ac9
        gopkg.in/mgo.v2 v2.0.0-20190816093944-a6b53ec6cb22)
```

3.3 多线程安全

审计期间未发现并发过程中的 slice 安全问题和 map 的读写安全问题。

3.4 密码学安全

本次审计不包含 dcrm 功能模块。

3.5 RPC 接口安全

对暴露在外部的 API 进行测试，审计期间未发现安全问题。

- rpc/rpcapi/api.go

```
GetVersionInfo(r *http.Request, args *RPCNullArgs, result *string) error
GetServerInfo(r *http.Request, args *RPCNullArgs, result *swapapi.ServerInfo) error
GetSwapStatistics(r *http.Request, args *RPCNullArgs, result *swapapi.SwapStatistics) error
GetRawSwapin(r *http.Request, txid *string, result *swapapi.Swap) error
GetRawSwapinResult(r *http.Request, txid *string, result *swapapi.SwapResult) error
GetSwapin(r *http.Request, txid *string, result *swapapi.SwapInfo) error
GetRawSwapout(r *http.Request, txid *string, result *swapapi.Swap) error
GetRawSwapoutResult(r *http.Request, txid *string, result *swapapi.SwapResult) error
GetSwapout(r *http.Request, txid *string, result *swapapi.SwapInfo) error
GetSwapinHistory(r *http.Request, args *RPCQueryHistoryArgs, result *[]*swapapi.SwapInfo) error
GetSwapoutHistory(r *http.Request, args *RPCQueryHistoryArgs, result *[]*swapapi.SwapInfo) error
Swapin(r *http.Request, txid *string, result *swapapi.PostResult) error
RetrySwapin(r *http.Request, txid *string, result *swapapi.PostResult) error
P2shSwapin(r *http.Request, args *RPCP2shSwapinArgs, result *swapapi.PostResult) error
Swapout(r *http.Request, txid *string, result *swapapi.PostResult) error
IsValidSwapinBindAddress(r *http.Request, address *string, result *bool) error
IsValidSwapoutBindAddress(r *http.Request, address *string, result *bool) error
RegisterP2shAddress(r *http.Request, bindAddress *string, result *tokens.P2shAddressInfo) error
GetP2shAddressInfo(r *http.Request, p2shAddress *string, result *tokens.P2shAddressInfo) error
GetLatestScanInfo(r *http.Request, isSrc *bool, result *swapapi.LatestScanInfo) error
RegisterAddress(r *http.Request, address *string, result *swapapi.PostResult) error
GetRegisteredAddress(r *http.Request, address *string, result *swapapi.RegisteredAddress) error
BuildSwapoutTx(r *http.Request, args *BuildSwapoutTxArgs, result *types.Transaction) error
AdminCall(r *http.Request, rawTx *string, result *string) error
```

调用以太坊 API 接口获取信息

tokens/eth/callapi.go

```
SendSignedTransaction -> eth_sendRawTransaction
ChainID     ->          eth_chainId
NetworkID   ->          net_version
GetCode     ->          eth_getCode
CallContract->          eth_call


tokens/eth/callcontract.go
GetErc20Balance         ->          balanceOf
GetErc20Decimals        ->          decimals
```

调用比特币 API 接口获取信息

- tokens/btc/electrs/callapi.go

```
GetLatestBlockNumber    ->          apiAddress + "/blocks/tip/height"
GetTransactionByHash    ->          apiAddress + "/tx/" + txHash
GetElectTransactionStatus           ->          apiAddress + "/tx/" + txHash + "/status"
FindUtxos   ->          apiAddress + "/address/" + addr + "/utxo"
GetPoolTxidList         ->          apiAddress + "/mempool/txids"
GetPoolTransactions     ->          apiAddress + "/address/" + addr + "/txs/mempool"
GetTransactionHistory   ->          apiAddress + "/address/" + addr + "/txs/chain"
GetOutspend             ->          apiAddress + "/tx/" + txHash + "/outspend/" + fmt.Sprintf("%d", vout)
PostTransaction         ->          apiAddress + "/tx"
GetBlockHash            ->          apiAddress + "/block-height/" + fmt.Sprintf("%d", height)
GetBlockTxids           ->          apiAddress + "/block/" + blockHash + "/txids"
```

在进行测试的时候，代码中设置的响应头是 application/json 但是实际上返回的是 text/plain，需要开发人员排查这个 bug。

- rpc/restapi/api.go

```
func writeResponse(w http.ResponseWriter, resp interface{}, err error) {
        if err == nil {
                jsonData, _ := json.Marshal(resp)
                w.Header().Set("Content-Type", "application/json")
                _, _ = w.Write(jsonData)
        } else {
                fmt.Fprintln(w, err.Error())
```

```
}}
```

Request

Raw Headers Hex JSON/JS Beautifier

```
GET /serverinfo HTTP/1.1
Host: 47.111.179.118:21336
Connection: close
Content-type: application/json
Accept: */*
Content-Length: 0
```

Response

Raw Headers Hex Render LINK-DUMPER

```
HTTP/1.1 200 OK
Date: Sun, 16 Aug 2020 08:09:47 GMT
Content-Length: 1091
Content-Type: text/plain; charset=utf-8
Connection: close

{"Identifier":"USDT2FSN","SrcToken":{"BlockChain":"Ethereum","
NetID":"mainnet","ID":"ERC20","Name":"USDTERC20","Symbol":"US
DT","Decimals":6,"Description":"ERC20
USDT","DepositAddress":"0xB846E6CalcE92c9B3e0175f591BF663a439
855C8","DcrmAddress":"0xffd223d9d07731c0768923bcE3b9B73d6aEcB
619","ContractAddress":"0xdac17f958d2ee523a2206206994597c13d8
31ec7","Confirmations":30,"MaximumSwap":100000,"MinimumSwap":
0.5,"BigValueThreshold":10000,"SwapFeeRate":0,"MaximumSwapFee
":0,"MinimumSwapFee":0,"InitialHeight":10631461,"PlusGasPrice
Percentage":10,"DisableSwap":false},"DestToken":{"BlockChain"
:"Fusion","NetID":"mainnet","ID":"mUSDT","Name":"SMPC ERC20
USDT","Symbol":"mUSDT","Decimals":6,"Description":"cross
chain bridge USDT with
mUSDT","DcrmAddress":"0xffd223d9d07731c0768923bcE3b9B73d6aEcB
619","ContractAddress":"0xbba09567c994faf2769bdb26ec9dd98ef9e
58b01","Confirmations":30,"MaximumSwap":100000,"MinimumSwap":
20,"BigValueThreshold":10000,"SwapFeeRate":0.001,"MaximumSwap
Fee":50,"MinimumSwapFee":10,"InitialHeight":0,"PlusGasPricePe
rcentage":1,"DisableSwap":false},"Version":"0.2.14-alpha"}
```

3.6 nil 指针安全

审计期间未发现 nil 指针错误使用导致的安全问题。

3.7 业务逻辑漏洞

比特币公链的交易采用扫描区块，从区块中获取交易信息，并对交易的确认数进行了配置和检查，合法的交易会记录到 db 当中，但是在检查交易的时候没有校验特殊交易，如：CLTV，CSV 这两种时间锁交易，需要在指定时间才能进行花费，所以攻击者可以设置一个极长的时间锁的恶意交易，但是能够通过交易检查，但是项目方收到后需要等到指定的时间才能正常花费，这里存在利用时间锁进行 DoS 的问题，建议添加对时间锁的检查，避免因为恶意交易导致资金流动受到影响。

- tokens/btc/verifyp2shtx.go

```
func (b *Bridge) VerifyP2shTransaction(txHash, bindAddress string, allowUnstable bool) (*tokens.TxSwapInfo, error) {
        swapInfo := &tokens.TxSwapInfo{}
        swapInfo.Hash = txHash // Hash
        if !b.IsSrc {
                return swapInfo, tokens.ErrBridgeDestinationNotSupported
        }
        p2shAddress, _, err := b.GetP2shAddress(bindAddress)
        if err != nil {
                return swapInfo, fmt.Errorf("verify p2sh tx, wrong bind address %v", bindAddress)
        }
```

```
	if !allowUnstable && !b.checkStable(txHash) {
		return swapInfo, tokens.ErrTxNotStable
	}
	tx, err := b.GetTransactionByHash(txHash)
	if err != nil {
		log.Debug(b.TokenConfig.BlockChain+" Bridge::GetTransaction fail", "tx", txHash, "err", err)
		return swapInfo, tokens.ErrTxNotFound
	}
	txStatus := tx.Status
	if txStatus.BlockHeight != nil {
		swapInfo.Height = *txStatus.BlockHeight // Height
	}
	if txStatus.BlockTime != nil {
		swapInfo.Timestamp = *txStatus.BlockTime // Timestamp
	}
	value, _, rightReceiver := b.getReceivedValue(tx.Vout, p2shAddress, p2shType)
	if !rightReceiver {
		return swapInfo, tokens.ErrTxWithWrongReceiver
	}
	swapInfo.To = p2shAddress                          // To
	swapInfo.Bind = bindAddress                        // Bind
	swapInfo.Value = common.BigFromUint64(value) // Value

	swapInfo.From = getTxFrom(tx.Vin, p2shAddress) // From

	// check sender
	if swapInfo.From == swapInfo.To {
		return swapInfo, tokens.ErrTxWithWrongSender
	}

	if !tokens.CheckSwapValue(swapInfo.Value, b.IsSrc) {
		return swapInfo, tokens.ErrTxWithWrongValue
	}

	if !allowUnstable {
		log.Debug("verify p2sh swapin pass", "from", swapInfo.From, "to", swapInfo.To, "bind", swapInfo.Bind,
"value", swapInfo.Value, "txid", swapInfo.Hash, "height", swapInfo.Height, "timestamp", swapInfo.Timestamp)
```

```
        }
        return swapInfo, nil}
```

- tokens/btc/verifytx.go

```
func (b *Bridge) verifySwapinTx(txHash string, allowUnstable bool) (*tokens.TxSwapInfo, error) {
        swapInfo := &tokens.TxSwapInfo{}
        swapInfo.Hash = txHash // Hash
        if !allowUnstable && !b.checkStable(txHash) {
                return swapInfo, tokens.ErrTxNotStable
        }
        tx, err := b.GetTransactionByHash(txHash)
        if err != nil {
                log.Debug(b.TokenConfig.BlockChain+" Bridge::GetTransaction fail", "tx", txHash, "err", err)
                return swapInfo, tokens.ErrTxNotFound
        }
        txStatus := tx.Status
        if txStatus.BlockHeight != nil {
                swapInfo.Height = *txStatus.BlockHeight // Height
        }
        if txStatus.BlockTime != nil {
                swapInfo.Timestamp = *txStatus.BlockTime // Timestamp
        }
        depositAddress := b.TokenConfig.DepositAddress
        value, memoScript, rightReceiver := b.getReceivedValue(tx.Vout, depositAddress, anyType)
        if !rightReceiver {
                return swapInfo, tokens.ErrTxWithWrongReceiver
        }
        swapInfo.To = depositAddress                          // To
        swapInfo.Value = common.BigFromUint64(value) // Value

        memoStr, bindAddress, bindOk := getBindAddressFromMemoScipt(memoScript)
        if memoStr == aggregateMemo {
                return swapInfo, tokens.ErrTxIsAggregateTx
        }

        swapInfo.Bind = bindAddress // Bind
```

```
        swapInfo.From = getTxFrom(tx.Vin, depositAddress) // From

        // check sender
        if swapInfo.From == swapInfo.To {
                return swapInfo, tokens.ErrTxWithWrongSender
        }

        if !tokens.CheckSwapValue(swapInfo.Value, b.IsSrc) {
                return swapInfo, tokens.ErrTxWithWrongValue
        }

        if !bindOk {
                log.Debug("wrong memo", "memo", memoScript)
                return swapInfo, tokens.ErrTxWithWrongMemo
        } else if !tokens.DstBridge.IsValidAddress(swapInfo.Bind) {
                log.Debug("wrong bind address in memo", "bind", swapInfo.Bind)
                return swapInfo, tokens.ErrTxWithWrongMemo
        }

        if !allowUnstable {
                log.Debug("verify swapin pass", "from", swapInfo.From, "to", swapInfo.To, "bind", swapInfo.Bind, "value",
swapInfo.Value, "txid", swapInfo.Hash, "height", swapInfo.Height, "timestamp", swapInfo.Timestamp)
        }
        return swapInfo, nil}
```

3.8 拒绝服务漏洞

在解析 ERC20 的 input data 的时候没有对 input data 长度进行检查，导致存在 panic 的问题，攻击者可以构造恶意的 input data，当节点扫描到这笔交易并进行解析的时候，就会发生 DoS。建议在处理 input data 之前先对 input data 的长度进行检查。

- tokens/eth/verifyerc20tx.go

```
func parseErc20EncodedData(encData []byte, isTransferFrom bool) (from, to string, value *big.Int, err error) {
        if isTransferFrom {
                from = common.BytesToAddress(common.GetData(encData, 0, 32)).String()
                encData = encData[32:]
        }
```

```
        if len(encData) != 64 {
                return "", "", nil, fmt.Errorf("wrong length of encoded data")
        }
        to = common.BytesToAddress(common.GetData(encData, 0, 32)).String()
        value = common.GetBigInt(encData, 32, 32)
        return from, to, value, nil}
```

3.9 敏感信息泄漏

审计期间未发现敏感信息泄露的安全问题。

3.10 日志处理安全

审计期间未发现日志处理的安全问题

3.11 缓存信息安全

扫描到的数据均存储在 MongoDB 当做，审计期间未发现安全问题，存在增强点，建议 db 存数据的时候对数据加上盐进行哈希，获取数据之前，必须要校验数据与 hash 一致才能通过验证，避免由于数据库中的数据被篡改导致资金的损失。

3.12 访问控制策略

admin 的操作需要先在本地使用私钥签名好交易，然后通过 RPC 发送和验证。

- cmd/swapadmin/utils.go

```
func adminCall(method string, params []string) (result interface{}, err error) {
        rawTx, err := admin.Sign(method, params)
        if err != nil {
                return "", err
        }
        err = client.RPCPost(&result, swapServer, "swap.AdminCall", rawTx)
        return result, err}
```

- admin/sign.go

```
func Sign(method string, params []string) (rawTx string, err error) {
        log.Info("admin Sign", "method", method, "params", params)
```

```
	payload, err := encodeCallArgs(method, params)
	if err != nil {
		return "", err
	}

	tx := types.NewTransaction(
		0,                  // nonce
		adminToAddr,     // to address
		big.NewInt(0), // value
		0,                  // gasLimit
		big.NewInt(0), // gasPrice
		payload,            // data
	)

	signedTx, err := types.SignTx(tx, adminSigner, keyWrapper.PrivateKey)
	if err != nil {
		return "", err
	}

	txdata, err := rlp.EncodeToBytes(signedTx)
	if err != nil {
		return "", err
	}

	return common.ToHex(txdata), nil}
```

- tools/loadkeystore.go

```
// LoadKeyStore load keystore from keyfile and passfile
func LoadKeyStore(keyfile, passfile string) (*keystore.Key, error) {
	keyjson, err := ioutil.ReadFile(keyfile)
	if err != nil {
		return nil, fmt.Errorf("read keystore fail %v", err)
	}
	passdata, err := ioutil.ReadFile(passfile)
	if err != nil {
		return nil, fmt.Errorf("read password fail %v", err)
	}
```

```
        passwd := strings.TrimSpace(string(passdata))
        key, err := keystore.DecryptKey(keyjson, passwd)
        if err != nil {
                return nil, fmt.Errorf("decrypt key fail %v", err)
        }
        return key, nil}
```

3.13 配置文件安全

config.toml 配置文件中会存放 smtp 的密码，建议可以将账号密码放在环境变量中避免配置文件泄露导致账号密码一并泄露。

3.14 异常未捕获处理

审计期间发现了 ERC20 input data 未检测，panic 异常未捕获导致 DoS 的问题。
参见：3.8 拒绝服务漏洞。

3.15 递归和循环安全

审计期间未发现递归错误或循环错误导致的安全问题。

3.16 数据库操作安全

在查询历史数据的时候没有对负数进行检查，当数据量大的时候，通过 limit 为负数查询会将数据库服务的资源耗尽。建议对 limit 的范围进行检查，禁止输入负数。
internal/swapapi/api.go

```
func processHistoryLimit(limit int) int {
        if limit == 0 {
                limit = 20
        } else if limit > 100 {
                limit = 100
        }
        return limit}
```

4 审计结论

4.1 严重漏洞

严重漏洞会对项目的安全造成重大影响，强烈建议修复严重漏洞。

经过审计该项目未发现严重漏洞。

4.2 高危漏洞

高危漏洞会影响项目的正常运行，强烈建议修复高危漏洞。

审计期间发现了两个高危漏洞：

1. 业务逻辑漏洞
2. 拒绝服务漏洞

这两个高危漏洞会对资金产生损失，强烈建议修复高危漏洞。

详细内容参见：3.7 业务逻辑漏洞，3.8 拒绝服务漏洞。

4.3 中危漏洞

中危漏洞会影响项目的运行，建议修复中危漏洞。

经过审计该项目未发现中危漏洞。

4.4 低危漏洞

低危漏洞可能会影响未来版本代码中的操作，建议项目方自行评估和考虑这些问题是否需要修复。

审计期间发现一个低危漏洞：

1. 在进行历史数据查询的时候没有判断 limit 的值为负数的情况，随着数据的积累后续会因为该问题导致节点资源被耗尽。

详细内容参见：3.16 数据库操作安全。

4.5 增强建议

增强建议是对项目的优化建议，项目方自行评估和考虑这些问题是否需要优化。

审计期间发现 4 个增强点：

1. 与 RPC 之间通信的安全，要采用 https 进行通信。
2. RPC 接口的响应头返回的是 text/plain，但是代码中设置的响应头是 application/json 需要开发人员排查这个 bug。
3. 建议 db 存数据的时候对数据加上盐进行哈希，获取数据之前，必须要校验数据与 hash 一致才能通过验证，避免由于数据库中的数据被篡改导致资金的损失。
4. config.toml 配置文件中会存放 smtp 的密码，建议可以将账号密码放在环境变量中避免配置文件泄露导致账号密码一并泄露

详细内容参见：3.1 通讯安全，3.5 RPC 接口安全，3.11 缓存信息安全，3.13 配置文件安全。

4.6 结论

总结: 本次审计中发现了 7 个问题，其中包含 2 个高危漏洞、1 个低危漏洞，4 个增强建议。经与项目方沟通后，高危和低危漏洞均已修复，增强点会在后续版本的代码中进行加强。

5 声明

慢雾科技
slow mist

官方网址

www.slowmist.com

电子邮箱

team@slowmist.com

微信公众号