HALO Network Security Audit Report

Contents

11. Executive Summary

On July 13, 2021, the SlowMist security team received the Halo Network team's security audit application for Halo, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "black, grey box lead, white box assists" to conduct a complete security test on the project in the way closest to the real attack.

SlowMist blockchain system test method:

Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code module through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect wether there are vulnerabilities in programs suck as nodes, SDK, etc.

SlowMist blockchain risk level:

Critical vulnerabilities	Critical vulnerabilities will have a significant impact on the security of the blockchain, and it is strongly recommended to fix the critical vulnerabilities.

High-risk vulnerabilities	High-risk vulnerabilities will affect the normal operation of blockchain. It is strongly recommended to fix high-risk vulnerabilities.
Medium-risk vulnerablities	Medium vulnerability will affect the operation of blockchain. It is recommended to fix medium-risk vulnerabilities.
Low-risk vulnerabilities	Low-risk vulnerabilities may affect the operation of blockchain in certain scenarios. It is suggested that the project party should evaluate and consider whether these vulnerabilities need to be fixed.
Weaknesses	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.
Enhancement Suggestions	There are better practices for coding or architecture.

2. Project Background (Context)

2.1 Project Introduction

Project Website: https://www.halo.land

Coin Symbol: HO

Project source code: https://github.com/HaloNetwork/Halo_Source_Code

Audit version: commit:1cbcf502a38def05b5a869d13bdd9fa055318a4a

2.2 Scope of Audit

The main types of security audit include:

(other unknown security vulnerabilities are not included in the scope of responsibility of this audit)

No.	Audit Category	Audit Result
1	Code Compliance Audit	Some Risks
2	Random Number Generation Algorithm Audit	PASSED
3	Keystore Audit	PASSED
4	Cryptographic Component Call Audit	PASSED
5	Encryption Strength Audit	PASSED
6	Length Extension Attack Audit	PASSED
7	Transaction Malleability Attack Audit	PASSED
8	Replay Attack Audit	PASSED
9	Top-up Program Audit	PASSED
10	RPC Permission Audit	PASSED

3. Code Overview

3.1 Infrastructure

Halo is based on the open source heco-chain development.

3.2 Code Compliance Audit

Forked Github repo: https://github.com/HuobiGroup/huobi-eco-chain

Forked version: v1.0.0

Main change feature:

- Change `chainid` to 1280;
- Change block reward logic;
- Remove validator punishment;

Main change files:

```
consensus/congress/abi.go
consensus/congress/congress.go
consensus/congress/interactive.go
eth/config.go
params/config.go
```

Fork open source blockchain source code or using similar protocol will cause problems such as replay attacks and node peer pool pollution. We conduct relevant security compliance assessments for this.

P2P protocol is the same with Ethereum mainnet, it may cause node peer pool pollution.

Reference: https://mp.weixin.qq.com/s/UmricgYGUakAlZTb0ihqdw

3.3 Random Number Generation Algorithm Audit

The generation of the private key seed is based on the `crypto/rand` standard library, and the entropy value is secure.

- crypto/crypto.go

```
func GenerateKey() (*ecdsa.PrivateKey, error) {
return ecdsa.GenerateKey(S256(), rand.Reader)
}
```

3.4 Keystore Audit

Use the keystore to encrypt the storage, and the password strength is not verified. Weak passwords such as `123456` can be used in the test, which can be easily cracked.

3.5 Cryptographic Component Call Audit

Signature algorithm: Secp256k1

Hash algorithm: SHA256

Using Ethereum cryptography-related components widely used in the industry, no security risks have been found.

3.6 Encryption Strength Audit

Weak hash functions such as md5 and sha1 are not used.

3.7 Length Extension Attack Audit

In cryptography and computer security, a length extension attack is a type of attack where an attacker can use Hash(message1) and the length of message1 to calculate Hash(message1 ‖ message2) for an attacker-controlled message2, without needing to know the content of message1. Algorithms like MD5, SHA-1, and SHA-2 that are based on the Merkle–Damgard construction are susceptible to this kind of attack. The SHA-3 algorithm is not susceptible.

No error calls were found.

3.8 Transaction Malleability Attack Audit

Allowing transactions with any s value with 0 < s < secp256k1n, as is currently the case, opens a transaction malleability concern, as one can take any transaction, flip the s value from s to secp256k1n - s, flip the v value (27 -> 28, 28 -> 27), and the resulting signature would still be valid. This is not a serious security flaw, especially since Ethereum uses addresses and not transaction hashes as the input to an ether value transfer or other transaction, but it nevertheless creates a UI inconvenience as an attacker can cause the transaction that gets confirmed in a block to have a different hash from the transaction that any user sends, interfering with user interfaces that use transaction hashes as tracking IDs. Preventing high s values removes this problem.

This problem were fixed in EIP-2.

Vulnerability reference:

https://en.bitcoinwiki.org/wiki/Transaction_Malleability

https://github.com/ethereum/EIPs/blob/master/EIPS/eip-2.md

3.9 Transaction Replay Attack Audit

Each transaction is signed with a unique `nonce` value, and there is no replay attack problem in the same chain.

Use `chainid` to distinguish different chains when signing transactions, and there is no replay attack problem for transactions between different chains.

- core/types/transaction_signing.go

```
// Hash returns the hash to be signed by the sender.
```

```
// It does not uniquely identify the transaction.
func (s EIP155Signer) Hash(tx *Transaction) common.Hash {
return rlpHash([]interface{}{
tx.data.AccountNonce,
tx.data.Price,
tx.data.GasLimit,
tx.data.Recipient,
tx.data.Amount,
tx.data.Payload,
s.chainId, uint(0), uint(0),
})
}
```

3.10 Top-up Program Audit

The structure of Receipt is as follows, the `Status` field is used to mark the status of the transaction.

- core/types/receipt.go

```
// Receipt represents the results of a transaction.
type Receipt struct {
// Consensus fields
PostState []byte `json:"root"`
Status uint `json:"status"`
CumulativeGasUsed uint64 `json:"cumulativeGasUsed" gencodec:"required"`
Bloom Bloom `json:"logsBloom" gencodec:"required"`
Logs []*Log `json:"logs" gencodec:"required"`

// Implementation fields (don't reorder!)
TxHash common.Hash `json:"transactionHash" gencodec:"required"`
ContractAddress common.Address `json:"contractAddress"`
GasUsed uint64 `json:"gasUsed" gencodec:"required"`
}
```

Initiate a transfer transaction on the main network, the test data is as follows:

```
> personal.unlockAccount(eth.accounts[0])
> eth.sendTransaction({from: eth.accounts[0],to: eth.accounts[1], value: 1000})
```

```
"0xaee12d984ad589d3a0611b397f267f4474dc8058faada44bc80f18a5b4ff3c8f"
> eth.getTransaction('0xaee12d984ad589d3a0611b397f267f4474dc8058faada44bc80f18a5b4ff3c8f')
{
  blockHash: "0x6a4088af2fcd49cdf122b3a414168703e21bacc55a77b5832091354bbe8ed5fa",
  blockNumber: 1616857,
  from: "0xd412972904431c8ad834922a83c2d3b30dfb0aab",
  gas: 21000,
  gasPrice: 1000000000,
  hash: "0xaee12d984ad589d3a0611b397f267f4474dc8058faada44bc80f18a5b4ff3c8f",
  input: "0x",
  nonce: 0,
  r: "0xfeb02efbf14f1b5f19733e75af34169e92dffd8a112d2dd4b50e25c8e9e0b42f",
  s: "0x3178a15b87c40356f60b5b58af589cbf9f4ebee5da88fc34c648442c7050d86a",
  to: "0x91e121833050b712e56bf2b006dba8daa5ce8c59",
  transactionIndex: 0,
  v: "0xa24",
  value: 1000
}
> eth.getTransactionReceipt('0xaee12d984ad589d3a0611b397f267f4474dc8058faada44bc80f18a5b4ff3c8f')
{
  blockHash: "0x6a4088af2fcd49cdf122b3a414168703e21bacc55a77b5832091354bbe8ed5fa",
  blockNumber: 1616857,
  contractAddress: null,
  cumulativeGasUsed: 21000,
  from: "0xd412972904431c8ad834922a83c2d3b30dfb0aab",
  gasUsed: 21000,
  logs: [],
  logsBloom:
"0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000",
  status: "0x1",
  to: "0x91e121833050b712e56bf2b006dba8daa5ce8c59",
  transactionHash: "0xaee12d984ad589d3a0611b397f267f4474dc8058faada44bc80f18a5b4ff3c8f",
  transactionIndex: 0
}
```

When the exchange passes the recharge receipt, it needs to strictly verify the values of `to`, `value` and `status`.

3.11 RPC Permission Audit

RPC has a wallet function. By default, the node keeps the RPC port open in the WAN, which is insecure. The exchange should disable the account module and keep the port open in the local host.

Vulnerability reference: https://mp.weixin.qq.com/s/Kk2lsoQ1679Gda56Ec-zJg

4. Audit Result

4.1 Low-risk Vulnerabilitys

- Weak passwords can be used in the keystore, which can be easily cracked.
- P2P protocol is the same with Ethereum mainnet, it may cause node peer pool pollution.

4.2 Enhancement Suggestions

- There are RPC "Black Valentine's Day Vulnerabilities", which can lead to node privacy disclosure or asset theft. It is recommended to prohibit unlocking accounts when opening RPC, or only open RPC ports locally.
- Keep RPC port closed, or do not open in WAN.

4.3 Exchange Suggestions

- The exchange should check all relevant fields in the transaction and receipt structure, and real-time reconciliation with the total balance of the account. If an abnormality occurs, it needs to be manually checked before processing the entry to prevent "false top-up attacks."
- When a withdrawal transaction fails, the original transaction needs to be rebroadcast or repackaged with the same `nonce` to prevent malicious repeated withdrawals.
- It is forbidden to open the RPC interface to the WAN to prevent node privacy leakage or asset theft.

- When top-up a contract address, it is necessary to determine whether there is a revert transaction in the inline transaction. If there is a revert transaction, the account will be rejected.

4.4 Conclusion

Audit result: PASSED

Audit No. : BCA002107200001

Audit date: July 20, 2021

Audit team: SlowMist security team

Summary conclusion: After correction, all problems found have been fixed and the above risks have been eliminated by Halo. Comprehensive assessed, Halo no risks above already.

5. Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility base on these.
For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance this report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

Github
https://github.com/slowmist