Blockchain Security Audit Report

Table Of Contents

1 Executive Summary

On 2024.07.10, the SlowMist security team received the HashKey team's security audit application for HashKey Layer2, 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.

The test method information:

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules 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 whether there are vulnerabilities in programs such as nodes, SDK, etc.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project party should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.
Suggestion	There are better practices for coding or architecture.

2 Audit Methodology

The security audit process of SlowMist security team for the chain includes two steps:

Chain codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.

Manual audit of the codes for security issues. The codes are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the chain:

NO.	Audit Items	Result
1	Others	Some Risks
2	Non-privacy/Non-dark Coin Audit	Passed
3	Cryptographic Security Audit	Passed
4	Account and Transaction Security Audit	Passed
5	RPC Security Audit	Passed
6	P2P Security Audit	Some Risks
7	Consensus Security Audit	Passed

3 Project Overview

3.1 Project Introduction

HashKey L2 network built with OP Stack. The OP Stack is the set of software that powers Optimism — currently in the form of the software behind OP Mainnet and eventually in the form of the Optimism Superchain and its governance.

3.2 Coverage

Target Code and Revision:

optimism:

https://github.com/HashKeyChain/optimism

commit:

915f5c0b10d1c08ce59a71d790b391f1e3fd76bc

op-geth:

https://github.com/HashKeyChain/op-geth

commit:

7c2819836018bfe0ca07c4e4955754834ffad4e0

Hashkay Chain Testnet

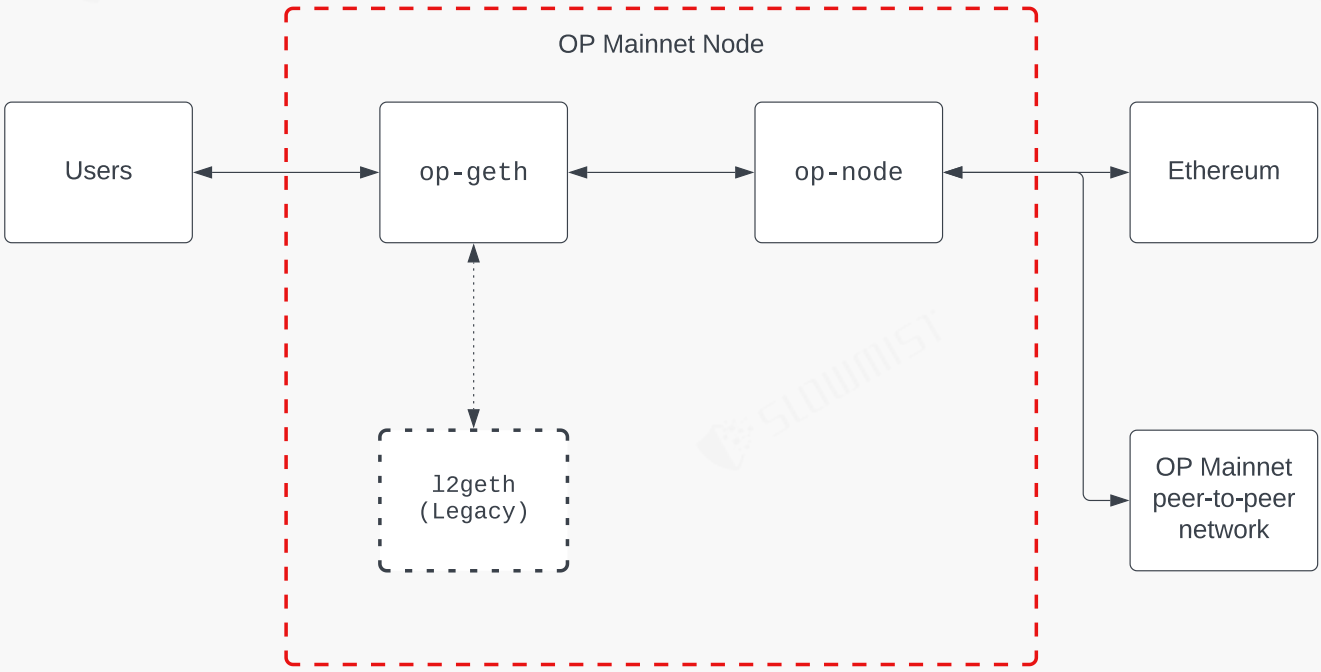
RPC endpoint: https://hashkeychain-testnet.alt.technology

ChainID: 133

Symbol: HSK

Explorer: https://hashkeychain-testnet-explorer.alt.technology

Node Architecture:



Others

Contract Authorization Access Audit

Malicious contracts or poor quality contracts deployed on the main network can cause users' assets to suffer, in order to ensure the security of contracts deployed on the main network, all deployed contracts need to be officially certified. The development team modified the code of op-geth to achieve this goal, the relevant code:

- core/state_processor_test.go
- core/error.go
- core/state_transition.go

```
if msg.To == nil && !st.checkContractCreationPassedDeploymentContract(msg.From,
msg.Data){
    return nil, ErrContractNotPassDeploymentControllerCheck
}
```

The primary method implemented checks that the transaction's from address belongs to an authenticated user and that the code is from a verified source.

Non-privacy/Non-dark Coin Audit

The main check is whether a blockchain project has transparency and compliance, that is, whether it meets regulatory requirements such as Anti-Money Laundering (AML) and Know Your Customer (KYC).
In Op Stack, privacy coin technologies such as zero-knowledge proofs are not used. All transactions are open and transparent, ensuring that illegal funds can be traced and sourced.

Cryptographic Security Audit

(1) Insufficient entropy of private key random numbers audit

The main point is to ensure that there is sufficient randomness and unpredictability when generating the private key of the cryptocurrency wallet.

In the OP Stack, roles such as the Batcher, Proposer, and other privileged roles need to generate key pairs (public and private keys) for various use cases. The current code does not provide tools for key generation and management; instead, it relies on external tools. For example, Clef is recommended when using geth, and it is also suggested to use HSMs (Hardware Security Modules) or hardware wallets.

(2) Precision loss in private key seed conversion

The main task is to check whether there is any precision loss during the conversion process from the seed to the address of the private key.

In the OP Stack, the current code does not provide tools for key generation and management; instead, it relies on external tools.

(3) Theoretical reliability assessment of symmetric encryption algorithms

The main check is to evaluate the theoretical reliability of symmetric encryption algorithms.

The following cryptographic algorithms are used in OP Stack:

```
"crypto/aes"
```

(4) Supply chain security of symmetric crypto algorithm reference libraries

The main task is to check the supply chain security of the reference libraries of the symmetric encryption algorithms used.

The used cryptographic algorithm libraries are

- golang.org/x/crypto

and the crypto library that comes with the official Go.

(5) Keystore encryption strength detection

The main purpose is to check the encryption strength of the Keystore.

In the OP Stack, the current code does not provide tools for key generation and management; instead, it relies on external tools.

(6) Hash algorithm length extension attack

The main purpose is to check the length extension attack of the hash algorithm.

In the transactions of OP Stack, Keccak-256 is used for hashing. Keccak-256 is a variant of the algorithm based on SHA-3, and this attack scenario does not exist.

(7) Merkle-tree Malleability attack

The main purpose is to check the modifiability of the Merkle Tree in the blockchain system.

In some merkle-tree algorithms, if there are an odd number of nodes, the last node is automatically copied to spell an even number, and an attacker may try to double-flush by including two identical transactions at the end of the block.

The Merkle-tree scheme used by OP Stack is consistent with that of Ethereum and there are no security issues.

(8) secp256k1 k-value randomness security

The main purpose is to examine the randomness and security of the k value in the secp256k1 elliptic curve encryption algorithm.

The k value of Secp256k1 in Optimism is generated based on the RFC-6979 standard. It is not random but kept confidential.

- crypto/secp256k1/secp256.go

```
func Sign(msg []byte, seckey []byte) ([]byte, error) {
   //...
   if C.secp256k1_ecdsa_sign_recoverable(context, &sigstruct, msgdata, seckeydata,
noncefunc, nil) == 0 {
      return nil, ErrSignFailed
   }
   //...
   return sig, nil
}
```

(9) secp256k1 r-value reuse private key extraction attack

The main purpose is to examine the influence of r-value reuse in the secp256k1 elliptic curve encryption algorithm on private key extraction attacks.

The security of the r value stems from the random security of the k value. Since the k value is secure, then the r value is also secure here.

- crypto/secp256k1/libsecp256k1/src/secp256k1.c

```
int secp256k1_ecdsa_sign(const secp256k1_context* ctx, secp256k1_ecdsa_signature
*signature, const unsigned char *msg32, const unsigned char *seckey,
secp256k1_nonce_function noncefp, const void* noncedata) {
   secp256k1_scalar r, s;
```

```
    secp256k1_scalar sec, non, msg;
    int ret = 0;
    int overflow = 0;
    VERIFY_CHECK(ctx != NULL);
    ARG_CHECK(secp256k1_ecmult_gen_context_is_built(&ctx->ecmult_gen_ctx));
    ARG_CHECK(msg32 != NULL);
    ARG_CHECK(signature != NULL);
    ARG_CHECK(seckey != NULL);
    if (noncefp == NULL) {
        noncefp = secp256k1_nonce_function_default;
    }

    secp256k1_scalar_set_b32(&sec, seckey, &overflow);
    /* Fail if the secret key is invalid. */
    if (!overflow && !secp256k1_scalar_is_zero(&sec)) {
        unsigned char nonce32[32];
        unsigned int count = 0;
        secp256k1_scalar_set_b32(&msg, msg32, NULL);
        while (1) {
            ret = noncefp(nonce32, msg32, seckey, NULL, (void*)noncedata, count);
            if (!ret) {
                break;
            }
            secp256k1_scalar_set_b32(&non, nonce32, &overflow);
            if (!overflow && !secp256k1_scalar_is_zero(&non)) {
                if (secp256k1_ecdsa_sig_sign(&ctx->ecmult_gen_ctx, &r, &s, &sec, &msg,
&non, NULL)) {
                    break;
                }
            }
            count++;
        }
        memset(nonce32, 0, 32);
        secp256k1_scalar_clear(&msg);
        secp256k1_scalar_clear(&non);
        secp256k1_scalar_clear(&sec);
    }
    if (ret) {
        secp256k1_ecdsa_signature_save(signature, &r, &s);
    } else {
        memset(signature, 0, sizeof(*signature));
    }
    return ret;
}
```

(10) ECC signature malleability attack

The main task is to examine the malleability of signatures in Elliptic Curve Digital Signature (ECC signature), as well as the related security risks and potential attacks.

There is an extensibility problem in secp256k1. By changing the s value in the signature to s - N or N - s, the signature result can be changed. This led to the attack on the mt.gox in the early version. However, this only affects the security of off-chain applications and does not affect the security of OP Stack itself.

(11) ed25519 private key extraction attack

The main purpose is to check the risks and impacts of the private key extraction attack in the Ed25519 elliptic curve digital signature algorithm.

Ed25519 is a digital signature algorithm based on elliptic curves, which is widely used in the fields of blockchain and cryptocurrencies. The private key extraction attack refers to an attacker's attempt to recover the signer's private key from known signatures by exploiting vulnerabilities or weaknesses in the system. If successful, the attacker can forge valid signatures, thereby performing malicious operations such as tampering with transactions and impersonating identities.

OP Stack does not use the library of ed25519 with security issues.

(12) Schnorr private key extraction attack

The main task is to check the risks and impacts of the private key extraction attack existing in the Schnorr signature algorithm.

OP Stack has not used the Schnorr signature algorithm.

(13) ECC twist attack

The main task is to check the risks and impacts of the elliptic curve twist attack existing in Elliptic Curve Cryptography (ECC).

OP Stack does not use the elliptic curve cryptography library with security issues.

(14) Supply chain security of hash algorithm reference libraries

The main task is to check the supply chain security of the reference libraries of the encryption hash algorithms used.

The used cryptographic algorithm libraries are

golang.org/x/crypto

and the Crypto library that comes with the official Go.

(15) Theoretical reliability assessment of hash algorithms

The main check is to evaluate the theoretical reliability of encryption hash algorithms.

The following cryptographic hash algorithms are used in Optimism:

```
"crypto/sha256"
"crypto/sha3"
```

Account and Transaction Security Audit

(1) Native characteristic false recharge

The main objective is to examine the risk and impact of possible native feature false top-up vulnerabilities in blockchain systems.

Native feature false recharge refers to attackers exploiting the characteristics or vulnerabilities of the blockchain system to forge recharge records on the blockchain. This kind of attack may cause losses to users, exchanges or blockchain platforms. It should be noted that OP Stack has a notable feature: failed transactions may still be submitted to the chain, and the status of each transaction needs to be confirmed.

(2) Contract call-based false recharge

The main check is the risks and impacts of potential false recharge vulnerabilities based on contract calls that may exist in blockchain smart contracts.

OP Stack adopts an EVM-compatible smart contract architecture, which brings powerful functions while also inheriting certain complex characteristics. One of the particularly notable ones is the behavior of cross-contract calls. In a complex transaction, even if an internal cross-contract call fails, the entire transaction may still be marked as successful. This feature may lead to potential security risks, such as vulnerabilities like 'false top-up'. Therefore, it is necessary to deeply verify the execution results of all internal calls in the transaction. Only when all related internal transactions are successfully executed can the validity of the entire transaction be truly confirmed.

(3) Native chain transaction replay attack

The main purpose is to assess the risks and impacts of possible local chain transaction replay attacks in the blockchain network.

Transaction replay attacks refer to a type of attack where the attacker resubmits previously valid transaction data to the blockchain network, deceiving network nodes and participants, causing the transaction to be executed repeatedly. This may result in unnecessary financial losses, transaction delays, or other negative impacts.

For each address in OP Stack, a Nonce is added as a parameter for transactions. After a successful transaction, the Nonce is incremented by one. Therefore, there is no problem of transaction replay.

(4) Heterogeneous chain transaction replay attack

The main task is to examine the risks and impacts of potential transaction replay attacks that may exist between heterogeneous chains.

Transaction replay attacks between heterogeneous chains refer to a type of attack where attackers exploit the interoperability between different blockchain networks to repeat a valid transaction that was successfully executed on one chain on another chain, in order to gain improper benefits or cause system damage. In this type of attack, the attacker copies previously valid transaction data across chains and submits it to another chain, deceiving nodes and participants on the chain, causing the transaction to be executed repeatedly. This may result in financial losses, transaction delays, or other negative impacts.

Although OP Stack is designed to be fully compatible with the Ethereum Virtual Machine (EVM), a specific chainID is embedded in the signature data of each transaction to prevent the transaction from being directly re-executed on another chain.

(5) Transaction lock attack

The main task is to check the risks and impacts of possible transaction locking attacks in the blockchain system.

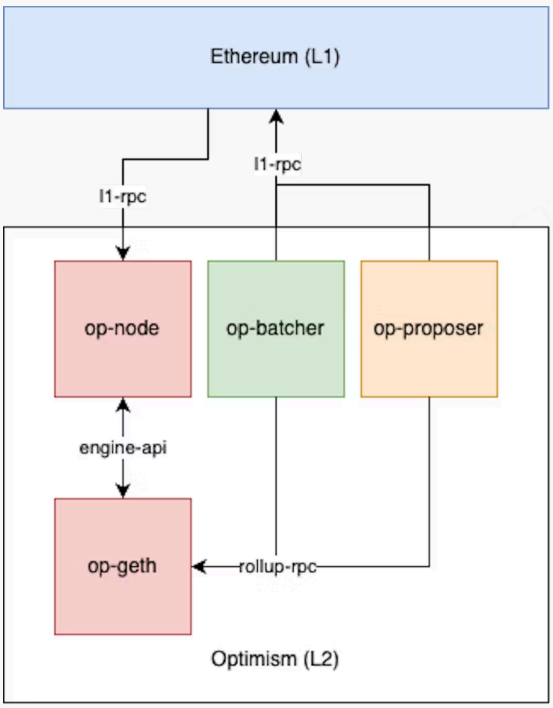
Transaction locking attack refers to the manipulation of blockchain transactions by malicious users or attackers to make certain funds or resources remain locked for a long period of time, in order to achieve the purpose of causing damage to the system, interfering with it, or obtaining undue benefits. In this type of attack, the attacker may take advantage of incompletely understood smart contract logic, blockchain protocol vulnerabilities, or transaction

sequencing rules to make specific transactions unable to be confirmed or executed, resulting in funds or resources being locked for a long period of time, affecting the normal operation of the system and user experience.

Transactions in OP Stack do not have a locking function.

RPC Security Audit

In the OP Stack, communication between different nodes within the system is conducted via RPC calls. As long as the configuration is correct, these internal communications will not be vulnerable to external attacks. However, RPC calls that are exposed to external users may be subject to various types of attacks. During our audit, we primarily focus on the security of the sequencer externally exposed ports.



(1) RPC remote key theft attack

The main task is to examine the risks and impacts of potential RPC remote key theft attacks that may exist in the blockchain system.

In OP Stack Sequencer, there are no RPC interfaces for managing wallets, etc.

(2) RPC port identifiability

The main check is to assess the identifiability of RPC ports in the blockchain system to determine whether the system is vulnerable to attacks targeting RPC ports.

When deploying and running the OP Stack node service, special attention needs to be paid to the access control of certain highly sensitive RPC interfaces. These interfaces, such as TransferLeader and RemoveServer, have potential system-level impacts and may directly affect the stability and security of the network.

Therefore, after the node service is started, it becomes crucial to implement a strict access restriction policy. This is not just a suggestion but a necessary measure to maintain the integrity of the network. Administrators should configure detailed Access Control Lists (ACLs) to only allow access to these sensitive interfaces from authenticated and authorized addresses or IP ranges.

(3) RPC open cross-domain vulnerability to local phishing attacks

The main check is to assess the cross-domain vulnerabilities of RPC interfaces in the blockchain system to determine whether the system is vulnerable to local phishing attacks.

The Sequencer RPC services of Optimism do not enable Cross-Origin Resource Sharing (CORS) support by default configuration.

(4) JsonRPC malformed packet denial-of-service attack

The main check is the security of the JsonRPC interface in the blockchain system to determine whether the system is vulnerable to Denial of Service (DoS) attacks caused by maliciously constructed abnormal JSON data packets.

When conducting robustness tests on the Sequencer RPC interface of OP Stack, we constructed a special boundary case data packet:

```
Data = "'{slowmist}' * 0x101000 + '":' + '{"x":' * 0x10000 + '"}'"
```

The purpose was to test the behavior of the RPC server when facing requests of abnormal size and format. The returned result "413 Request Entity Too Large" did not cause the node to crash.

(5) RPC database injection

The main check is whether there is a database injection problem.

OP Stack uses LevelDB by default as the database for its blockchain data. LevelDB is a key-value pair repository and does not operate using SQL statements, so there is no database injection problem.

(6) RPC communication encryption

The main purpose is to check whether the RPC (Remote Procedure Call) communication in the blockchain system has appropriate encryption protection.

In the Op-Stack L2 architecture:

op-conductor is responsible for managing and coordinating the consensus process of multiple nodes. It provides a set of RPC APIs such as AddServerAsVoter, AddServerAsNonvoter, RemoveServer, etc., used to control and monitor the status of nodes and cluster membership. HTTPS encrypted communication is not enabled.

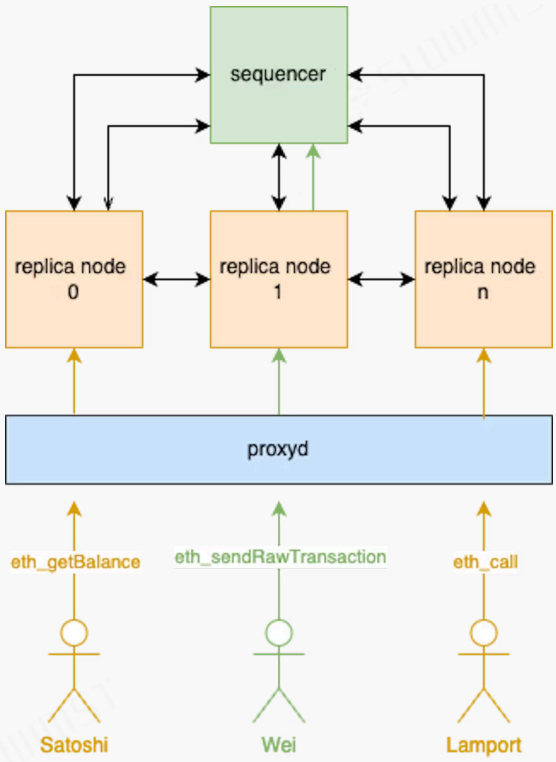
op-batcher: responsible for reading transaction content from the sequencer at regular intervals and rolling it up to on-chain DA. It provides RPC API interfaces such as StartBatcher, StopBatcher. HTTPS encrypted communication is not enabled.

op-proposer: responsible for rolling up transaction states to the contract. It provides RPC API interfaces such as StartProposer. HTTPS encrypted communication is not enabled.

However, these RPC interfaces are generally not available to the public and do not have transaction data, account information, etc.

P2P Security Audit

OP Stack P2P communication occurs between sequecer and replica nodes:



(1) P2P communication encryption

The main check is whether the P2P (peer-to-peer) communication between nodes in the blockchain network uses an appropriate encryption mechanism to protect the privacy and security of the communication content.

```
In the Op-Stack L2 architecture, the Sequencer composed of op-node and op-geth needs
to synchronize block information with external Replica nodes through the P2P protocol.

Devp2p:
Op-Stack adopts the widely used devp2p protocol of Ethereum as the basis of its P2P
network, supporting functions such as node discovery, connection, and data exchange.
The RLPx transport protocol is used in devp2p, and this protocol uses the ECIES
encryption method based on secp256k1/hmac/AES in communication.

Gossip:
Op-Stack adopts a customized Gossip protocol to deliver newly generated blocks and
transactions.

Since the identities of the nodes have been completed in the devp2p interaction
stage, the p2p of gossip does not use encrypted communication.

Relevant codes:
- op-node/p2p/discovery.go
- op-node/p2p/gossip.go
- op-node/p2p/node.go


func (n *NodeP2P) init(resourcesCtx context.Context, rollupCfg *rollup.Config, log
```

```
log.Logger, setup SetupP2P, gossipIn GossipIn, l2Chain L2Chain, runCfg
GossipRuntimeConfig, metrics metrics.Metricer, elSyncEnabled bool) error {
    bwc := p2pmetrics.NewBandwidthCounter()

    n.log = log

    var err error
    // nil if disabled.
    n.host, err = setup.Host(log, bwc, metrics)
    if err != nil {
       if n.dv5Udp != nil {
          n.dv5Udp.Close()
       }
       return fmt.Errorf("failed to start p2p host: %w", err)
    }

    // TODO(CLI-4016): host is not optional, NodeP2P as a whole is. This if statement
is wrong
    if n.host != nil {
       // Enable extra features, if any. During testing we don't setup the most
advanced host all the time.
       if extra, ok := n.host.(ExtraHostFeatures); ok {
          n.gater = extra.ConnectionGater()
          n.connMgr = extra.ConnectionManager()
       }
       eps, ok := n.host.Peerstore().(store.ExtendedPeerstore)
       if !ok {
          return fmt.Errorf("cannot init without extended peerstore: %w", err)
       }
       n.store = eps
       scoreParams := setup.PeerScoringParams()

       if scoreParams != nil {
          n.appScorer = newPeerApplicationScorer(resourcesCtx, log, clock.SystemClock,
&scoreParams.ApplicationScoring, eps, n.host.Network().Peers)
       } else {
          n.appScorer = &NoopApplicationScorer{}
       }
       // Activate the P2P req-resp sync if enabled by feature-flag.
       if setup.ReqRespSyncEnabled() && !elSyncEnabled {
          n.syncCl = NewSyncClient(log, rollupCfg, n.host.NewStream,
gossipIn.OnUnsafeL2Payload, metrics, n.appScorer)
          n.host.Network().Notify(&network.NotifyBundle{
             ConnectedF: func(nw network.Network, conn network.Conn) {
                n.syncCl.AddPeer(conn.RemotePeer())
             },
             DisconnectedF: func(nw network.Network, conn network.Conn) {
                // only when no connection is available, we can remove the peer
                if nw.Connectedness(conn.RemotePeer()) == network.NotConnected {
```

```
                    n.syncCl.RemovePeer(conn.RemotePeer())
                }
            },
        })
        n.syncCl.Start()
        // the host may already be connected to peers, add them all to the sync
client
        for _, peerID := range n.host.Network().Peers() {
            n.syncCl.AddPeer(peerID)
        }
        if l2Chain != nil { // Only enable serving side of req-resp sync if we have
a data-source, to make minimal P2P testing easy
            n.syncSrv = NewReqRespServer(rollupCfg, l2Chain, metrics)
            // register the sync protocol with libp2p host
            payloadByNumber := MakeStreamHandler(resourcesCtx, log.New("serve",
"payloads_by_number"), n.syncSrv.HandleSyncRequest)
            n.host.SetStreamHandler(PayloadByNumberProtocolID(rollupCfg.L2ChainID),
payloadByNumber)
        }
    }
    n.scorer = NewScorer(rollupCfg, eps, metrics, n.appScorer, log)
    // notify of any new connections/streams/etc.
    n.host.Network().Notify(NewNetworkNotifier(log, metrics))
    // note: the IDDelta functionality was removed from libP2P, and no longer needs
to be explicitly disabled.
    n.gs, err = NewGossipSub(resourcesCtx, n.host, rollupCfg, setup, n.scorer,
metrics, log)
    if err != nil {
        return fmt.Errorf("failed to start gossipsub router: %w", err)
    }
    n.gsOut, err = JoinGossip(n.host.ID(), n.gs, log, rollupCfg, runCfg, gossipIn)
    if err != nil {
        return fmt.Errorf("failed to join blocks gossip topic: %w", err)
    }
    log.Info("started p2p host", "addrs", n.host.Addrs(), "peerID",
n.host.ID().String())

    tcpPort, err := FindActiveTCPPort(n.host)
    if err != nil {
        log.Warn("failed to find what TCP port p2p is binded to", "err", err)
    }

    // All nil if disabled.
    n.dv5Local, n.dv5Udp, err = setup.Discovery(log.New("p2p", "discv5"),
rollupCfg, tcpPort)
    if err != nil {
        return fmt.Errorf("failed to start discv5: %w", err)
    }
```

```
        if metrics != nil {
           go metrics.RecordBandwidth(resourcesCtx, bwc)
        }

        if setup.BanPeers() {
           n.peerMonitor = monitor.NewPeerMonitor(resourcesCtx, log, clock.SystemClock,
n, setup.BanThreshold(), setup.BanDuration())
           n.peerMonitor.Start()
        }
        n.appScorer.start()
     }
     return nil
}
```

(2) Insufficient number of core nodes

The main check is whether the number of core nodes in the blockchain network is sufficient to ensure the security and stability of the network.

In the OP Stack L2 architecture, the Sequencer composed of op-node and op-geth needs to synchronize block information with external Replica nodes through the P2P protocol. In the consensus security model of OP Stack, its security is mainly provided by fraud-challenge proof and does not require a large number of block production nodes. The Replica nodes will provide data relay services for external RPC communication. Therefore, there is no problem of insufficient node quantity in OP Stack.

(3) Excessive concentration of core node physical locations

The main check is whether the physical location distribution of the core nodes in the blockchain network is too concentrated.

In the consensus security model of OP Stack, its consensus security is mainly provided by the fraud-challenge proof and does not require a large number of block production nodes. Therefore, there is no problem of overly concentrated core nodes in OP Stack.

(4) P2P node maximum connection limit

The main check is the maximum connection limit of a blockchain node to other nodes.

In the OP Stack L2 architecture, the Sequencer composed of op-node and op-geth needs to synchronize block information with external Replica nodes through the P2P protocol.

- op-node/p2p/config.go

```
const maxMeshParam = 1000
```

(5) P2P node independent IP connection limit

The main check is the limit on the number of independent connections of the blockchain node to the same IP address.

In the OP Stack L2 architecture, the Sequencer composed of op-node and op-geth needs to synchronize block information with the external Replica node through the P2P protocol, but the protocol does not limit the number of node connections in the same IP network. Malicious attackers may use a large number of Sybil nodes at a relatively low cost to occupy the available connection pool of the Sequencer, resulting in other nodes being unable to connect to the Sequencer. However, as a functional supplement, the node provides a method for actively disabling the IP.

- op-node/p2p/node.go

```
func (n *NodeP2P) BanPeer(id peer.ID, expiration time.Time) error {
   //...
}


func (n *NodeP2P) BanIP(ip net.IP, expiration time.Time) error {
   //...
}
```

(6) P2P inbound/outbound connection limit

The main check is the limit on the number of incoming and outgoing connections of the blockchain node.

In the OP Stack L2 architecture, the Sequencer composed of op-node and op-geth needs to synchronize block information with the external Replica node through the P2P protocol. The Sequencer uses the libp2p component to synchronize blocks, and no issue of unrestricted inbound/outbound has been found in this protocol currently.

(7) P2P Alien attack

The Alien attack vulnerability was first discovered by the SlowMist team and is also known as peer pool pollution. It refers to an attack method that induces mutual invasion and pollution among similar chain nodes. The main reason

for this vulnerability is that similar chain systems fail to identify dissimilar nodes in the communication protocol.

In the OP Stack L2 architecture, the Sequencer composed of op-node and op-geth needs to synchronize block information with the external Replica node through the P2P protocol, using the discovery protocol of libp2p. No P2P Alien attack vulnerability has been found in this protocol currently.

Relevant code:

- op-node/p2p/host.go
- op-node/p2p/pings.go

(8) P2P port identifiability

The main purpose is to check whether the ports used for P2P (peer-to-peer) communication between nodes in the blockchain network are easy to be identified and tracked.

OP Stack has the ports 8545/8546/7060 open by default. Network attacks against OP Stack may take advantage of this port feature to scan all OP Stack nodes across the entire network and launch targeted attacks.

- ops-bedrock/docker-compose.yml

```
ports:
  - "8545:8545"
  - "8546:8546"
  - "7060:6060"
```

Consensus Security Audit

(1) Excessive administrator privileges

The main task is to check whether the system has administrator permissions or special beneficiary accounts to ensure the rationality, minimization and decentralization of permission control, thereby guaranteeing that there is no fraudulent behavior in the system.

OP Stack has administrator permissions, but there are no special beneficiary accounts.

Relevant explanations of administrator permissions can be found here:

https://docs.optimism.io/chain/security/privileged-roles.

(2) Transaction fees not dynamically adjusted

The main check is whether the transaction fees in the blockchain system are dynamically adjusted according to the network conditions and demands.

OP Stack has an upper limit on the gas usage for each block, which is 0x7fffffffffffff. During transaction congestion, transactions will be sorted based on the PriorityFee level according to the rules of EIP-1559.

In the OP Stack, through the EIP-1559 mechanism, transaction fees are dynamically adjusted based on the network congestion situation to ensure the efficiency and fairness of transaction processing.

```
func applyLondonChecks(env *stEnv, chainConfig *params.ChainConfig) error {
    if !chainConfig.IsLondon(big.NewInt(int64(env.Number))) {
       return nil
    }
    // Sanity check, to not `panic` in state_transition
    if env.BaseFee != nil {
       // Already set, base fee has precedent over parent base fee.
       return nil
    }
    if env.ParentBaseFee == nil || env.Number == 0 {
       return NewError(ErrorConfig, errors.New("EIP-1559 config but missing
'currentBaseFee' in env section"))
    }
    env.BaseFee = eip1559.CalcBaseFee(chainConfig, &types.Header{
       Number:   new(big.Int).SetUint64(env.Number - 1),
       BaseFee:  env.ParentBaseFee,
       GasUsed:  env.ParentGasUsed,
       GasLimit: env.ParentGasLimit,
    }, env.Timestamp)
    return nil
}
```

```
const (
    GasLimitBoundDivisor uint64 = 1024               // The bound divisor of the gas
limit, used in update calculations.
    MinGasLimit          uint64 = 5000               // Minimum the gas limit may
ever be.
    MaxGasLimit          uint64 = 0x7fffffffffffffff // Maximum the gas limit (2^63-
1).
    GenesisGasLimit      uint64 = 4712388            // Gas limit of the Genesis
block.
```

```
    MaximumExtraDataSize  uint64 = 32    // Maximum size extra data may be after
Genesis.
    ExpByteGas            uint64 = 10    // Times ceil(log256(exponent)) for the EXP
instruction.
    SloadGas              uint64 = 50    // Multiplied by the number of 32-byte words
that are copied (round up) for any *COPY operation and added.
    CallValueTransferGas  uint64 = 9000  // Paid for CALL when the value transfer is
non-zero.
    CallNewAccountGas     uint64 = 25000 // Paid for CALL when the destination
address didn't exist prior.
    TxGas                 uint64 = 21000 // Per transaction not creating a contract.
NOTE: Not payable on data of calls between transactions.
    TxGasContractCreation uint64 = 53000 // Per transaction that creates a contract.
NOTE: Not payable on data of calls between transactions.
    TxDataZeroGas         uint64 = 4     // Per byte of data attached to a
transaction that equals zero. NOTE: Not payable on data of calls between transactions.
    QuadCoeffDiv          uint64 = 512   // Divisor for the quadratic particle of the
memory cost equation.
    LogDataGas            uint64 = 8     // Per byte in a LOG* operation's data.
    CallStipend           uint64 = 2300  // Free gas given at beginning of call.
...
}
```

(3) Miner grinding attack

The main purpose is to assess the potential risk of grinding attacks by miners in the blockchain system.

Consensus Mechanism:

OP Stack does not use the traditional Proof of Work (PoW) mechanism. It relies on the security of the Ethereum mainnet and uses a sequencer to sequence transactions. This means that traditional "mining" does not exist in Optimism.

Block Production:

In OP Stack, there is no competitive block production process. The Sequencer is responsible for packaging transactions and submitting them to the Ethereum mainnet. This eliminates the competitive element in traditional mining and thus reduces the possibility of grinding attacks.

(4) PoS/BFT final confirmation conditions

The security of OP Stack mainly depends on the Ethereum mainnet and its fraud-proof mechanism. Any malicious behavior can be detected and punished through fraud-proof.

(5) PoS/BFT double-signing penalty

The security of OP Stack mainly depends on the Ethereum mainnet and its fraud-proof mechanism. Any malicious behavior can be detected and punished through fraud-proof.

(6) PoS/BFT block refusal penalty

The security of OP Stack mainly depends on the Ethereum mainnet and its fraud-proof mechanism. Any malicious behavior can be detected and punished through fraud-proof.

(7) Block time offset attack

A block time manipulation attack occurs when a malicious miner deliberately timestamps a block with an incorrect time during its production. Blockchain systems typically tolerate a certain degree of time deviation, but if there are no limits on the permissible range, the production of subsequent blocks may not occur within the expected time frame.

- op-geth/core/blockchain.go

```
maxTimeFutureBlocks = 30
//...
func (bc *BlockChain) addFutureBlock(block *types.Block) error {
        max := uint64(time.Now().Unix() + maxTimeFutureBlocks)
        if block.Time() > max {
                return fmt.Errorf("future block timestamp %v > allowed %v",
block.Time(), max)
        }
    //...
}
```

OP Stack set future block timestamp to 30s, it's in a reasonable range.

(8) Consensus algorithm potential risk assessment

OP Stack is the L2 layer, consensus is still mainly dependent on Ethereum mainnet, Any malicious behaviour can be detected and punished through fraud-proof.

3.3 Vulnerability Information

The following is the status of the vulnerabilities found in this audit:

NO	Title	Category	Level	Status
N1	Privileged Role Management Issue	Rug Pull Attack	Medium	Confirming
N2	Potential false recharge vulnerability in contract call-based transactions	[Off-Chain]False Top-up Audit	Information	Confirming
N3	Native characteristic false recharge vulnerability in blockchain systems, with focus on Optimism	[Off-Chain]False Top-up Audit	Information	Confirming
N4	RPC identifiability and access control in Optimism nodes	Others	Low	Confirming
N5	P2P node independent IP connection limit in Op-Stack L2 architecture	P2P Security Audit	Low	Confirming
N6	P2P port identifiability in Optimism nodes	P2P Security Audit	Information	Confirming

4 Findings

4.1 Vulnerability Summary

[N1] [Medium] Privileged Role Management Issue

Category: Rug Pull Attack

Content

In the entire system operation of Optimism, there are several key roles, namely L1 Admin, L2 Admin, System Config Owner, and Mint Manager Owner.

L1 Admin: Can upgrade most of the OP mainnet system contracts. Risks include upgrading malicious version contracts, deleting or locking assets in the standard bridge, etc.

L2 Admin: Can upgrade most of the system contracts on L2. The risks are similar to those of the L1 proxy administrator and are controlled by the same account of the L1 proxy administrator through address aliases.

System Config Owner: Can change the values in the Ethereum contract. Risks include causing network disruption and users paying excessive transaction fees.

Mint Manager Owner: Controls the contract that can mint new OP tokens on the OP mainnet. Risks include minting tokens arbitrarily or preventing token minting.

If the private keys of these privileged roles are leaked, it will have a serious impact on the entire ecosystem.

Solution

For the management of these privileged roles, the official has given some suggestions:

L1 Admin: Manage using a 2-of-2 multi-signature address.

L2 Admin: Manage using a 2-of-2 multi-signature address.

System Config Owner: Manage using a 5-of-7 multi-signature address.

Mint Manager Owner: Manage using a 3-of-5 multi-signature address.

Status

Confirming

[N2] [Information] Potential false recharge vulnerability in contract call-based transactions

Category: [Off-Chain]False Top-up Audit

Content

Optimism's EVM-compatible smart contract architecture inherits complex characteristics, particularly in cross-contract calls. A notable feature is that even if an internal cross-contract call fails, the entire transaction may still be marked as successful. This behavior is not a vulnerability per se, but a characteristic that developers and auditors should be aware of.

This feature could potentially lead to security risks if not properly handled, such as "false recharge" vulnerabilities. To mitigate this risk, it is crucial to thoroughly verify the execution results of all internal calls within a transaction. The validity of the entire transaction should only be confirmed when all related internal transactions have been successfully executed.

This information serves as a reminder for developers to implement proper checks and balances in their smart contracts, especially when dealing with complex, multi-contract interactions. It emphasizes the importance of comprehensive testing and auditing to ensure the intended behavior of smart contracts in all scenarios.

Solution

Status

Confirming

[N3] [Information] Native characteristic false recharge vulnerability in blockchain systems, with focus on Optimism

Category: [Off-Chain]False Top-up Audit

Content

Native characteristic false recharge is a potential security concern in blockchain systems. It refers to a scenario where attackers exploit inherent characteristics or vulnerabilities of a blockchain to forge recharge records. This type of attack could result in financial losses for users, exchanges, or blockchain platforms.

Optimism, in particular, has a notable feature that warrants attention: failed transactions may still be submitted to the chain. This characteristic necessitates careful confirmation of the status of each transaction.

Solution

Status

Confirming

[N4] [Low] RPC identifiability and access control in Optimism nodes

Category: Others

Content

When deploying and running Optimism node services, special attention must be paid to the access control of highly sensitive RPC interfaces. Interfaces such as TransferLeader and RemoveServer have potential system-level impacts and may directly affect the network's stability and security.

The identifiability of RPC ports could make the system vulnerable to targeted attacks. To mitigate this risk, a strict access restriction policy must be implemented after the node service is started.

Solution

Therefore, after the node service is started, it becomes crucial to implement a strict access restriction policy. This is not just a suggestion but a necessary measure to maintain the integrity of the network. Administrators should configure detailed Access Control Lists (ACLs) to only allow access to these sensitive interfaces from authenticated and authorized addresses or IP ranges.

Status

Confirming

[N5] [Low] P2P node independent IP connection limit in Op-Stack L2 architecture

Category: P2P Security Audit

Content

In the Op-Stack L2 architecture, the Sequencer composed of op-node and op-geth needs to synchronize block information with the external Replica node through the P2P protocol, but the protocol does not limit the number of node connections in the same IP network. Malicious attackers may use a large number of Sybil nodes at a relatively low cost to occupy the available connection pool of the Sequencer, resulting in other nodes being unable to connect to the Sequencer. However, as a functional supplement, the node provides a method for actively disabling the IP.

Solution

It is advisable to limit the number of connections to a single IP.

Status

Confirming

[N6] [Information] P2P port identifiability in Optimism nodes

Category: P2P Security Audit

Content

In the case of Optimism, the default configuration opens ports 8545, 8546, and 7060 for P2P communication. This is evident in the ops-bedrock/docker-compose.yml file:

yamlCopyports:

- "8545:8545"
- "8546:8546"
- "7060:6060"

The use of well-known, default ports makes Optimism nodes easily identifiable on the network.

Attackers could potentially use these port characteristics to scan the entire network for Optimism nodes.

This identifiability could facilitate targeted attacks against Optimism nodes.

Solution

Status

Confirming

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002407260003	SlowMist Security Team	2024.07.10 - 2024.07.26	Low Risk

Summary conclusion: The SlowMist security team use a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 1 medium risk, 2 low risk vulnerabilities.

6 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 based 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 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