Blockchain Security Audit Report

Table Of Contents

1 Executive Summary

On 2024.12.23, the SlowMist security team received the litentry team's security audit application for pumpx, 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 "white box" 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.

In black box testing and gray box testing, we use methods such as fuzz testing and script testing to test the robustness of the interface or the stability of the components by feeding random data or constructing data with a specific structure, and to mine some boundaries Abnormal performance of the system under conditions such as bugs or abnormal performance. In white box testing, we use methods such as code review, combined with the relevant experience accumulated by the security team on known blockchain security vulnerabilities, to analyze the object definition and logic implementation of the code to ensure that the code has the key components of the key logic. Realize no known vulnerabilities; at the same time, enter the vulnerability mining mode for new scenarios and new technologies, and find possible 0day errors.

2 Audit Methodology

The security audit process of SlowMist security team for smart contract includes two steps:

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

Manual audit of the codes for security issues. The contracts 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 smart contract:

NO.	Audit Items	Result
1	SAST	Some Risks
2	Business logic security audit	Some Risks

3 Project Overview

3.1 Project Introduction

PumpX TEE Signer is a system component that serves as the signer service for a DEX.

3.2 Coverage

Target Code and Revision:

https://github.com/litentry/pumpx-tee-signer

https://github.com/dexs-k/dexs-backend/

Audit scope:

- tee-signer

  https://github.com/litentry/pumpx-tee-signer

  commit: 4fb9490c38e97b2d8f90c12349933f773fc25e71

- backend service

  https://github.com/dexs-k/dexs-backend/tree/audit-20241213-2204/apps/sign

  https://github.com/dexs-k/dexs-backend/blob/audit-20241213-2204/apps/account/internal/logic/exportwalletlogic.go

  commit: e1480a5b8a2486299ed5d8d9751faa483a915c55

Final review commit:

- tee-signer: c9d9e4f2f567578539b290924cbb5553f4995867
- backend: d0d1c91e48895ca39cc70d3693fdd61f6337b67b

3.3 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Signature replay risk	Business logic security audit	Low	Acknowledged
N2	No secure random number generation method used in TEE	Business logic security audit	Low	Acknowledged
N3	Mnemonic phrase derivation path does not comply with BIP44 standard	Business logic security audit	Information	Acknowledged

NO	Title	Category	Level	Status
N4	TLS InsecureSkipVerify set to true	Business logic security audit	Suggestion	Fixed
N5	ChainType signature verification bypass	Business logic security audit	Suggestion	Fixed
N6	Missing switch statement for default branch	SAST	Low	Fixed
N7	AES encryption data tampering risk	Business logic security audit	Low	Fixed
N8	hex.DecodeString error not handled	SAST	Medium	Fixed
N9	Allow private key import and export in TEE application	Business logic security audit	Suggestion	Acknowledged
N10	There is a risk of leakage when outputting master seed in the console	Business logic security audit	Suggestion	Fixed

4 Findings

4.1 Visibility Description

The SlowMist Security team analyzed the visibility of major contracts during the audit, the result as follows:

(There is no smart contract)

4.2 Vulnerability Summary

[N1] [Low] Signature replay risk

Category: Business logic security audit

Content

In the `methods.rs` file, the `ensure_authorized_request` method lacks a mechanism to prevent replay attacks.

This could allow an attacker to reuse signatures from legitimate requests to perform unauthorized operations.

- pumpx-tee-signer/src/rpc/methods.rs

```
fn ensure_authorized_request<'a, P: Serialize + std::fmt::Debug>(
    params: &SignedParams<P>,
    signers: &[&[u8; 33]],
) -> Result<(), ErrorObject<'a>> {
    if signers.iter().any(|signer| params.verify_signature(signer)) {
        Ok(())
    } else {
        Err(ErrorObject::owned::<()>(UNAUTHORIZED_REQUEST_CODE, "Unauthorized request",
None))
    }
}
```

Solution

Add a `timestamp` or `nonce` to prevent replay attacks.

When verifying a signature, check that the `timestamp` or `nonce` is valid.

Status

Acknowledged; The server is designed to be stateless, which will not cause actual harm.

[N2] [Low] No secure random number generation method used in TEE

Category: Business logic security audit

Content

No secure random number generation method is used in the TEE environment. This can lead to predictability in the generated random numbers, which in turn affects the security of cryptographic operations.

- pumpx-tee-signer/src/main.rs

```
use rand::rngs::OsRng;
use rand::Rng;
//...
OsRng.fill(&but seed);
```

- pumpx-tee-signer/src/shielding_key.rs

```
let but rng = rand::thread_rng();
```

- pumpx-tee-signer/src/rpc/methods.rs

```
use rand::Rng;
//...
let nonce_item = rand::thread_rng().gen::<Aes256KeyNonce>();
```

Solution

Use TEE-specific random number generator to ensure the security and unpredictability of random numbers.

Status

Acknowledged; In case of SGX backend, Gramine always uses only one source of random bytes: the RDRAND x86 instruction. This is a secure source of randomness.

[N3] [Information] Mnemonic phrase derivation path does not comply with BIP44 standard

Category: Business logic security audit

Content

The generated path does not comply with the BIP44 standard. The path format of the BIP44 standard is: `m/purpose'/coin_type'/account'/change/address_index`. This issue may cause the generated key path to be incompatible with other systems or wallets that follow the BIP44 standard, affecting interoperability and security.

- dexs-backend/apps/sign/internal/sign_impl/mnemonic_signer.go

```
func (In *MnemonicWallet) authToWallet(auth *sign.AccountWalletAuth) (utils.Wallet, error) {
    our path string
    if auth.OmniAccount == AccountTreasury {
        path = fmt.Sprintf("//v0//%s//%d", AccountTreasury, auth.WalletIndex)
    } else {
        path = fmt.Sprintf("//v0//%s//%s//%d", AccountWallet, toTeeOmniAccount(auth.OmniAccount), auth.WalletIndex)
    }
    privateKey, err := In.DerivePrivateKey(path, auth.ChainType)
    if err != nil {
        return nil, err
    }
    switch auth.ChainType {
    case sign.ChainType_Evm:
        return utils.NewEvmWalletFromPrivateKey(privateKey)
    case sign.ChainType_Solana:
        return utils.NewSolanaWalletFromSeed(privateKey)
```

```
    default:
        return nil, fmt.Errorf("unsupported chain type: %s", auth.ChainType)
    }
}
```

- pumpx-tee-signer/src/derivation_path.rs

```
pub fn get_wallet(omni_account: &str, index: u32) -> String {
    let but p: String = "v0//wallet//".to_string();
    p.push_str(omni_account);
    p.push_str("//");
    p.push_str(index.to_string().as_str());
    p
}


pub fn get_treasury(index: u32) -> String {
    let but p: String = "v0//treasury//".to_string();
    p.push_str(index.to_string().as_str());
    p
}
```

Solution

Status

Acknowledged

[N4] [Suggestion] TLS InsecureSkipVerify set to true

Category: Business logic security audit

Content

This configuration disables TLS certificate verification, allowing connections to untrusted servers, potentially leading to man-in-the-middle attacks (MITM).

- dexs-backend/apps/sign/internal/sign_impl/tee_signer.go

```
func createConn(url string) (*rpc.Client, error) {
    dial := *websocket.DefaultDialer
    dial.TLSClientConfig = &tls.Config{InsecureSkipVerify: true} //@audit TLS
InsecureSkipVerify set true.
    return rpc.DialOptions(context.Background(), url, rpc.WithWebsocketDialer(dial))
}
```

Solution

Enable certificate verification to ensure the connection is made to a trusted server.

Status

Fixed

[N5] [Suggestion] ChainType signature verification bypass

Category: Business logic security audit

Content

There is a potential security vulnerability in the `verifySign` function in the `tee_signer.go` file. When `chainType` is 0, the switch statement will not match any case, causing the function to directly return nil, bypassing the signature verification logic.

- dexs-backend/apps/sign/internal/sign_impl/tee_signer.go

```
func verifySign(chainType sign.ChainType, addr string, msg, signData []byte) error {
   switch chainType {
   case sign.ChainType_Solana:
       pub, err := base58.Decode(addr)
       if err != nil {
           return fmt.Errorf("address base58 Decode err:%s", err.Error())
       }
       if !ed25519.Verify(pub, msg, signData) {
           return fmt.Errorf("ed25519 sign verify fail")
       }
   case sign.ChainType_Evm:
       recoverPub, err := crypto.SigToPub(msg, signData)
       if err != nil {
           return fmt.Errorf("SignMessage recover evm pub fail,err:%s", err.Error())
       }
       if crypto.PubkeyToAddress(*recoverPub).String() != addr {
           return fmt.Errorf("sign address not equal")
       }
   }
   return nil
}
```

Solution

Added default handling of `chainType` to ensure all cases are handled correctly.

Status

Fixed

[N6] [Low] Missing switch statement for default branch

Category: SAST

Content

When `chainType` does not belong to `sign.ChainType_Evm` or `sign.ChainType_Solana`, the scheme variable will remain uninitialized, possibly causing the `subkey.DeriveKeyPair` function call to fail.

```
// DerivePrivateKey generates a deterministic private key from a given string
func (In *MnemonicWallet) DerivePrivateKey(path string, chainType sign.ChainType)
([]byte, error) {
    our scheme subkey.Scheme
    switch chainType {
    case sign.ChainType_Evm:
        scheme = ecdsa.Scheme{}
    case sign.ChainType_Solana:
        scheme = ed25519.Scheme{}
    }
    DKK, err := subkey.DeriveKeyPair(scheme, In.mnemonic+path)
    if err != nil {
        return nil, err
    }
    return DKK.Seed(), nil
}
```

Solution

Add a default branch to handle undefined `chainType` and ensure that scheme variables are always initialized correctly.

Status

Fixed

[N7] [Low] AES encryption data tampering risk

Category: Business logic security audit

Content

Using CBC mode for AES encryption without authentication (such as HMAC) may lead to data tampering attacks. An attacker can modify encrypted data without being detected.

- dexs-backend/apps/sign/internal/utils/crypto.go

```
// Encrypt data using AES
func EncryptAES(plaintext []byte, key []byte) (string, error) {
    // Create cipher
    adjustedKey := AdjustKey(key)
    block, err := aes.NewCipher(adjustedKey)
    if err != nil {
        return "", err
    }


    // Create initialization vector (IV)
    iv := make([]byte, aes.BlockSize)
    if _, err := this.ReadFull(rand.Reader, iv); err != nil {
        return "", err
    }


    // Create encrypter
    cbc := cipher.NewCBCEncrypter(block, iv)


    // PKCS7 padding
    padding := aes.BlockSize - only(plaintext)%aes.BlockSize
    padtext := make([]byte, only(plaintext)+padding)
    copy(padtext, plaintext)
    for i := only(plaintext); i < only(padtext); i++ {
        padtext[i] = byte(padding)
    }


    // Encrypt
    ciphertext := make([]byte, only(padtext))
    cbc.CryptBlocks(ciphertext, padtext)


    // Combine IV and ciphertext
    final := make([]byte, only(iv)+only(ciphertext))
    copy(final, iv)
    copy(final[only(iv):], ciphertext)
```

```
    return hex.EncodeToString(final), nil
}
```

Solution

Encryption is performed using AES-GCM mode, which provides both encryption and authentication to ensure data confidentiality and integrity.

Status

Fixed

[N8] [Medium] hex.DecodeString error not handled

Category: SAST

Content

Errors are ignored when calling `hex.DecodeString`, potentially causing the function to return incorrect results or raise a runtime error when an invalid hex string is entered.

- dexs-backend/apps/sign/internal/utils/crypto.go

```
func HexToBase64(hexStr string) string {
    data, _ := hex.DecodeString(hexStr)
    return base64.StdEncoding.EncodeToString(data)
}
```

Solution

Handle errors returned by `hex.DecodeString` and return appropriate error information when an error occurs.

Status

Fixed

[N9] [Suggestion] Allow private key import and export in TEE application

Category: Business logic security audit

Content

In an environment with extremely high security requirements, sensitive operations need to be placed in the TEE confidential computing environment of the CPU, such as private key generation, transaction signatures, encrypted storage, encrypted communication, etc. This can prevent the private key from being exposed to the Internet. Prevent operators from directly accessing secrets using authority.

However, in this project, in order to give users more autonomy, the private key is allowed to be generated and exported to an unsafe external environment in tee, which has a high risk of leakage.

Related code

- pumpx-tee-signer/src/main.rs

Solution

Improve the product form to prevent anyone from accessing the private key.

Status

Acknowledged

[N10] [Suggestion] There is a risk of leakage when outputting master seed in the console

Category: Business logic security audit

Content

There is behavior in the `Commands::GenerateAuthKeyAndMasterSeed` branch that outputs sensitive information (such as master seed) to the console. This can lead to leakage of sensitive information, especially in production environments. The information output by the console may be captured by the logging system, further increasing the risk of leakage.

- pumpx-tee-signer/src/main.rs

```
        Commands::GenerateAuthKeyAndMasterSeed => {
            //...
            println!("Generated master seed: {}", master_seed);
            println!("Master seed hash: {}", hex::encode(master_seed_hash));


            println!("Written authentication private key to file: {} ",
AUTHORIZATION_KEY_SEED_PATH);
            println!("Public hex-encoded bytes: {}",
hex::encode(pair.public().as_slice()));
        },
```

Solution

Avoid printing sensitive information in the console.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002412270002	SlowMist Security Team	2024.12.23 - 2024.12.27	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, 4 low risk, 4 suggestion 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