ExVul

Oct 2024

EXVUL WEB3 SECURITY AUDIT FOR OKX

WEB3 SECURITY

https://exvul.com

Table of Contents

1.EXECUTIVE SUMMARY

Exvul Web3 Security was engaged by Js-wallet-sdk to review Wallet SDK implementation. The assessment was conducted in accordance with our systematic approach to evaluate potential security issues based upon customer requirement. The report provides detailed recommendations to resolve the issue and provide additional suggestions or recommendations for improvement.

The outcome of the assessment outlined in chapter 3 provides the system's owners a full description of the vulnerabilities identified, the associated risk rating for each vulnerability, and detailed recommendations that will resolve the underlying technical issue.

1.1 Methodology

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology [10] which is the gold standard in risk assessment using the following risk models:

- Likelihood: represents how likely a particular vulnerability is to be uncovered and exploited in the wild.
- Impact: measures the technical loss and business damage of a successful attack.
- Severity: determine the overall criticality of the risk.

Likelihood can be: High, Medium and Low and impact are categorized into for: High, Medium, Low, Informational. Severity is determined by likelihood and impact and can be classified into five categories accordingly, Critical, High, Medium, Low, Informational shown in table 1.1.

Likelihood					
	High	*Informational*	Medium	High	Critical
	Medium	*Informational*	Low	Medium	High
	Low	*Informational*	Low	Low	Medium
		Informational	*Low*	*Medium*	*High*
			IMPACT		

Table 0.1 Overall Risk Severity

To evaluate the risk, we will be going through a list of items, and each would be labelled with a severity category. The audit was performed with a systematic approach guided by a comprehensive assessment list carefully designed to identify known and impact security issues. If our tool or analysis does not identify any issue, the contract can be considered safe regarding the assessed item. For any discovered issue, we might further deploy code on our private test environment and run tests to confirm the findings. If necessary, we would additionally build a PoC to demonstrate the possibility of exploitation. The concrete list of check items is shown in Table 1.2.

- Basic Coding Bugs: We first statically analyze given code with our proprietary static code analyzer for known coding bugs, and then manually verify (reject or confirm) all the issues found by our tool.
- Code and business security testing: We further review the business logic and examine the system operation to identify possible pitfalls and/or errors.
- Additional Recommendations: We also provide additional advice on coding and development from the perspective of proven programming practices.

Category	Assessment Item
Basic Coding Assessment	Apply Verification Control
	Authorization Access Control
	Forged Transfer Vulnerability
	Forged Transfer Notification
	Numeric Overflow
	Transaction Rollback Attack
	Transaction Block Stuffing Attack
	Soft Fail Attack
	Hard Fail Attack
	Abnormal Memo
	Abnormal Resource Consumption
	Secure Random Number
Advanced Source Code Scrutiny	Asset Security
	Cryptography Security
	Business Logic Review
	Source Code Functional Verification
	Account Authorization Control
	Sensitive Information Disclosure
	Circuit Breaker
	Blacklist Control
	System API Call Analysis
	Contract Deployment Consistency Check
Additional Recommendations	Semantic Consistency Checks
	Following Other Best Practices

Table 0.2: The Full List of Assessment Items

To better describe each issue we identified, we categorize the findings with Common Weakness Enumeration (CWE-699) [14], which is a community-developed list of software weakness types to better delineate and organize weaknesses around concepts frequently encountered in software development.

2. FINDINGS OVERVIEW

2.1 Project Info And Contract Address

Project Name: Js-wallet-sdk

Audit Time: August 15, 2024 - October 18, 2024

Language: TypeScript

File Name	Link
js-wallet-sdk/coin-aptos/*	https://github.com/okx/js-wallet-sdk 79dfe3bacaa797b38e146d2b9b2fdc079d79e2bf
js-wallet-sdk/coin-base/*	https://github.com/okx/js-wallet-sdk 79dfe3bacaa797b38e146d2b9b2fdc079d79e2bf
js-wallet-sdk/coin-bitcoin/*	https://github.com/okx/js-wallet-sdk 79dfe3bacaa797b38e146d2b9b2fdc079d79e2bf
js-wallet-sdk/coin-solana/*	https://github.com/okx/js-wallet-sdk 79dfe3bacaa797b38e146d2b9b2fdc079d79e2bf
js-wallet-sdk/coin-ton/*	https://github.com/okx/js-wallet-sdk 79dfe3bacaa797b38e146d2b9b2fdc079d79e2bf
js-wallet-sdk/crypto-lib/*	https://github.com/okx/js-wallet-sdk 79dfe3bacaa797b38e146d2b9b2fdc079d79e2bf

2.2 Summary

Severity	Found	
Critical	0	
High	0	
Medium	0	
Low	3	
Informational	0	

2.3 Key Findings

ID	Severity	Findings Title	Status	Confirm
NVE- 001	**Low**	Should clear the privateKey immediately	Ignore	Confirmed
NVE- 002	**Low**	Use Buffer.alloc instead of Buffer.allocunsafe to ensure the buffers are zero-filled, which is safer	Ignore	Confirmed
NVE- 003	**Low**	Useless parameter privatekey in function createAndSignVersionedTransaction	Ignore	Confirmed

Table 2.3: Key Audit Findings

3. DETAILED DESCRIPTION OF FINDINGS

3.1 Should clear the privateKey immediately

ID:	NVE-001	Location:	coin-ton/src/TonWallet.ts
Severity:	Low	Category:	Business Issues
Likelihood:	Low	Impact:	Low

Description:

When processing a signature transaction, if the incoming private key (privateKey) is not cleaned up in time after the public key is calculated, the private key may stay in the memory for a long time. This provides potential malicious code or attackers with an opportunity to read or steal the private key, which in turn leads to the leakage of the private key.

```
async simulateMultiTransaction(param: SignTxParams) {
     if (param.privateKey) {
         const {
             publicKey
         } = signUtil.ed25519.fromSeed(base.fromHex(param.privateKey));
         const publicKeyHex = base.toHex(publicKey);
         if (param.data.publicKey && param.data.publicKey != publicKeyHex) {
             throw new Error("public key not pair the private key");
         }
         if (!param.data.publicKey) {
             param.data.publicKey = publicKeyHex;
         }
         param.privateKey = '';
     } else {
         if (!param.data.publicKey) {
             throw new Error("both private key and public key are null");
         }
     }
     return this.signMultiTransaction(param);
 }
```

Recommend:

After using privateKey, you should immediately set its value to an empty string or use other methods to clean it up to avoid exposing sensitive information in memory. For example, after processing the signature-related logic, set the private key field to empty, as shown in the following code:

```
async simulateMultiTransaction(param: SignTxParams) {
     if (param.privateKey) {
         const {
             publicKey
         } = signUtil.ed25519.fromSeed(base.fromHex(param.privateKey));
          param.privateKey = '';
         const publicKeyHex = base.toHex(publicKey);
         if (param.data.publicKey && param.data.publicKey != publicKeyHex) {
             throw new Error("public key not pair the private key");
         }
         if (!param.data.publicKey) {
             param.data.publicKey = publicKeyHex;
         }

     } else {
         if (!param.data.publicKey) {
             throw new Error("both private key and public key are null");
         }
     }
     return this.signMultiTransaction(param);
 }
```

Status: Ignore

3.2 Use Buffer.alloc instead of Buffer.allocunsafe to ensure the buffers are zero-filled, which is safer

ID:	NVE-002	Location:	coin-ton/src/TonWallet.ts
Severity:	Low	Category:	Business Issues
Likelihood:	Low	Impact:	Low

Description:

Using Buffer.allocUnsafe to allocate memory in the code is a potential security risk. Buffer.allocUnsafe allocates an uninitialized memory area, which may contain old data or other sensitive information without being cleaned or initialized. An attacker can exploit this vulnerability to access sensitive data left by other processes or applications in this memory area, such as keys, passwords, or other sensitive information.

```
async signTonProof(param: SignTonProofParams): Promise<any> {
    const {timestamp, domain, payload} = param.proof;

    const timestampBuffer = Buffer.allocUnsafe(8);
    timestampBuffer.writeBigInt64LE(BigInt(timestamp));

    const domainBuffer = Buffer.from(domain);
    const domainLengthBuffer = Buffer.allocUnsafe(4);
    domainLengthBuffer.writeInt32LE(domainBuffer.byteLength);

    const address = Address.parse(param.walletAddress);

    const addressWorkchainBuffer = Buffer.allocUnsafe(4);
    addressWorkchainBuffer.writeInt32BE(address.workChain);

    const addressBuffer = Buffer.concat([
        addressWorkchainBuffer,
        address.hash,
    ]);
```

Recommend:

To ensure that the allocated memory is safe and has zero contents, you can use Buffer.alloc instead of Buffer.allocUnsafe. Buffer.alloc will automatically fill the allocated memory block with zeros to ensure that it does not contain previous sensitive data.

Status: Ignore

customer response:The performance of allocunsafe is higher than that of alloc. If the allocunsafe method is used correctly in the code, dirty data will not be used. Where the method is called, there is a safe assignment, and normal use will not cause problems.

3.3 Useless parameter privatekey in function createAndSignVersionedTransaction

ID:	NVE-003	Location:	coin-solana/src/api.ts
Severity:	Low	Category:	Business Issues
Likelihood:	Low	Impact:	Low

Description:

Function getSerializedVersionedTransaction has an argument privateKey which is not used in the function body, and this function also called by getSerializedTokenTransferVersionedTransaction ,should remove this argument.

```
export async function getSerializedVersionedTransaction(payer: string, blockHash: string, instructions: TransactionInstruction[], privateKey: string[]) {
    const messageV0 = new TransactionMessage({
        payerKey: new PublicKey(payer),
        recentBlockhash: blockHash,
        instructions,
    }).compileToV0Message();

    const transaction = new VersionedTransaction(messageV0);

    // const signers: Signer[] = [];
    // privateKey.forEach(key => {
    //     let keypair = Keypair.fromSecretKey(base.fromBase58(key));
    //     signers.push({
    //         publicKey: keypair.publicKey,
    //         secretKey: keypair.secretKey,
    //     });
    // });
    // transaction.sign(signers);
    //
    // if (!transaction.signature) {
    //     return Promise.reject("sign error");
    // }

    return Promise.resolve(base.toBase58(transaction.serialize()));
}
```

Recommend:

To improve code security and maintainability, it is recommended to remove unused privateKey parameters.

Status: Ignore

4.CONCLUSION

In this audit, we thoroughly analyzed **Js-wallet-sdk** Wallet-Sdk implementation. The problems found are described and explained in detail in Section 3. The problems found in the audit have been communicated to the project leader. We therefore consider the audit result to be PASSED. To improve this report, we greatly appreciate any constructive feedbacks or suggestions, on our methodology, audit findings, or potential gaps in scope/coverage.

5. APPENDIX

5.1 Basic Coding Assessment

5.1.1 Apply Verification Control

- Description: The security of apply verification
- Result: Not found
- Severity: Critical

5.1.2 Authorization Access Control

- Description: Permission checks for external integral functions
- Result: Not found
- Severity: Critical

5.1.3 Forged Transfer Vulnerability

- Description: Assess whether there is a forged transfer notification vulnerability in the code
- Result: Not found
- Severity: Critical

5.1.4 Transaction Rollback Attack

- Description: Assess whether there is transaction rollback attack vulnerability in the code.
- Result: Not found
- Severity: Critical

5.1.5 Transaction Block Stuffing Attack

- Description: Assess whether there is transaction blocking attack vulnerability.
- Result: Not found
- Severity: Critical

5.1.6 Soft Fail Attack Assessment

- Description: Assess whether there is soft fail attack vulnerability.
- Result: Not found
- Severity: Critical

5.1.7 Hard Fail Attack Assessment

- Description: Examine for hard fail attack vulnerability
- Result: Not found
- Severity: Critical

5.1.8 Abnormal Memo Assessment

- Description: Assess whether there is abnormal memo vulnerability in the code.
- Result: Not found
- Severity: Critical

5.1.9 Abnormal Resource Consumption

- Description: Examine whether abnormal resource consumption in code processing.
- Result: Not found
- Severity: Critical

5.1.10 Random Number Security

- Description: Examine whether the code uses insecure random number.
- Result: Not found
- Severity: Critical

5.2 Advanced Code Scrutiny

5.2.1 Cryptography Security

- Description: Examine for weakness in cryptograph implementation.
- Results: Not Found
- Severity: High

5.2.2 Account Permission Control

- Description: Examine permission control issue in the code
- Results: Not Found
- Severity: Medium

5.2.3 Malicious Code Behavior

- Description: Examine whether sensitive behavior present in the code
- Results: Not found
- Severity: Medium

5.2.4 Sensitive Information Disclosure

- Description: Examine whether sensitive information disclosure issue present in the code.
- Result: Not found
- Severity: Medium

5.2.5 System API

- Description: Examine whether system API application issue present in the code
- Results: Not found
- Severity: Low

6. DISCLAIMER

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without ExVul's prior written consent.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts ExVul to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. ExVul's position is that each company and individual are responsible for their own due diligence and continuous security. ExVul's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

7. REFERENCES

[1] MITRE. CWE- 191: Integer Underflow (Wrap or Wraparound).

https://cwe.mitre.org/data/ definitions/191.html.

[2] MITRE. CWE- 197: Numeric Truncation Error.

https://cwe.mitre.org/data/definitions/197. html.

[3] MITRE. CWE-400: Uncontrolled Resource Consumption.

https://cwe.mitre.org/data/ definitions/400.html.

[4] MITRE. CWE-440: Expected Behavior Violation.

https://cwe.mitre.org/data/definitions/440. html.

[5] MITRE. CWE-684: Protection Mechanism Failure.

https://cwe.mitre.org/data/definitions/ 693.html.

[6] MITRE. CWE CATEGORY: 7PK - Security Features.

https://cwe.mitre.org/data/definitions/ 254.html.

[7] MITRE. CWE CATEGORY: Behavioral Problems.

https://cwe.mitre.org/data/definitions/438. html.

[8] MITRE. CWE CATEGORY: Numeric Errors.

https://cwe.mitre.org/data/definitions/189.html.

[9] MITRE. CWE CATEGORY: Resource Management Errors.

https://cwe.mitre.org/data/ definitions/399.html.

[10] OWASP. Risk Rating Methodology.

https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology

ExVul

EXVUL
WEB3 SECURITY

www.exvul.com

contact@exvul.com

www.x.com/EXVULSEC

github.com/EXVUL-Sec