SECURITY AUDIT OF

OXALUS MOBILE WALLET

Public Report

Aug 12, 2022

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network.

EXECUTIVE SUMMARY

This Security Audit Report prepared by Verichains Lab on Aug 12, 2022. We would like to thank Oxalus for trusting Verichains Lab in auditing the mobile wallet. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the Oxalus Mobile Wallet. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified some vulnerable issues in the application, along with some recommendations.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Oxalus Mobile Wallet

Oxalus Wallet is the first NFT Game Wallet, which is safe and easy to use. With Oxalus Wallet, your account on all devices is synchronized and data is transferred within seconds.

By using, you have access to:

- Display of your in-game NFTs in an intuitive and interactive way
- Security for your assets that will be protected within Oxalus
- Controllability over what you own, where you can take action with full empowerment
- Store and transfer with multi-chain digital assets and tokens, including:

1.2. Audit scope

In this particular project, a timebox approach was used to define the consulting effort. This means that **Verichains Lab** allotted a prearranged amount of time to identify and document vulnerabilities. Because of this, there is no guarantee that the project has discovered all possible vulnerabilities and risks.

Furthermore, the security check is only an immediate evaluation of the situation at the time the check was performed. An evaluation of future security levels or possible future risks or vulnerabilities may not be derived from it.

The security check was conducted on commit `18ef43e6142f0fc1946469371f2750846abc0734` from git repository *https://git.xantus.network/oxalus/oxalus-wallet-mobile*.

1.3. Audit methodology

Verichains Lab's audit team mainly used the **Open Web Application Security Project (OWASP) Mobile Security Testing Guide (MTSG)**. The **MSTG** is a comprehensive manual for mobile app security development, testing and reverse engineering. It describes technical processes for verifying the controls listed in the **OWASP Mobile Application Verification Standard (MASVS)**. During the audit process, the audit team also used several tools for viewing, finding and verifying security issues of the app, such as following:

#	Name	Version
1	Mobile Security Framework (MobSF)	v3.5.0 beta
2	Frida tools	14.2.13
3	Android Studio	Bumblebee 2021.1.1

#	Name	Version
4	Visual Studio Code	1.64.2
5	Android Debug Bridge (adb)	1.0.41

Table 1. Tools used for audit

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the application functioning; creates a critical risk to the application; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the application with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the application with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 2. Severity levels

1.4. Disclaimer

Please note that security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a 100% secure application. However, auditing allows discovering vulnerabilities that were unobserved, overlooked during development and areas where additional security measures are necessary.

2. AUDIT RESULT

2.1. Overview

The Oxalus Mobile Wallet was written in `TypeScript` Programming Language using `React Native` Framework. It keeps the user's mnemonic seed and private key securely in their device's secure storage (`Keystore`/`Keychain`) with password protection.

The main features of the Oxalus Mobile Wallet are:

- Manage multi-wallets under one account.
- Swap tokens.
- One place for your digital assets, NFTs and dApps

2.2. Findings

This section contains a detailed analysis of all the vulnerabilities that were discovered by the audit team during the audit process.

Oxalus fixed the code according to Verichains's draft report in commit `f5832d9104637a5778721d1b842a6689516c86b9`.

#	Issue	Severity
1	Failed to decrypt seed phrase overwrite user's seed phrase with empty one	CRITICAL
2	Malicious sites can inject JS code into trusted sites	CRITICAL
3	Bypass lock mechanism	HIGH
4	Sensitive data stored in `AsyncStorage`	MEDIUM
5	Malicious sites can inject fake site to bypass connect popup	MEDIUM
6	Can't go back to menu after deleting contact	LOW
7	`value` should be sorted by name after `quoteOfToken`	LOW

Audit team also suggested some possible enhancements and notes.

#	Issue	Severity
1	`getWalletFromPwd` should not `saveWallet`	INFORMATIVE

#	Issue	Severity
2	Hard coded `seedphrase_encrypted`	INFORMATIVE
3	Typos	INFORMATIVE
3	Webview JS injection	INFORMATIVE
3	Biometric issue on some android devices	INFORMATIVE

Oxalus fixed the code, according to Verichains's draft report.

2.3. Issues

2.3.1. Failed to decrypt seed phrase overwrite user's seed phrase with empty one CRITICAL

Affected files:

- `src/store/wallet/functions.ts`

In `changePassword` function, if `crypto.AES.decrypt` failed to decrypt user's current seed (either wrong `currentPass` or error with `AsyncStorage.getItem`), `seedPhrase` will be empty string and after re-encoding with `newPass`, user will lose their seed phrase.

```
export const changePassword = async (currentPass: string, newPass: string) => {
    const dec = (await AsyncStorage.getItem(STORE_KEY.SEEDPHRASE)) || '';
    const seedPhrase = crypto.AES.decrypt(dec, currentPass).toString(
        crypto.enc.Utf8,
    );
    const enc = crypto.AES.encrypt(seedPhrase, newPass).toString();
    AsyncStorage.setItem(STORE_KEY.SEEDPHRASE, enc);
};
```

RECOMMENDATION

Checking for `seedPhrase` is not empty (or even recover the wallet with `currentPass`) to make sure we can properly decode current seed phrase before re-encoding with `newPass`.

```
export const changePassword = async (currentPass: string, newPass: string) => {
    const dec = (await AsyncStorage.getItem(STORE_KEY.SEEDPHRASE)) || '';
    const seedPhrase = crypto.AES.decrypt(dec, currentPass).toString(
        crypto.enc.Utf8,
    );

    // handle decryt failed.
    if (!seedPhrase){
        throw Error('');
    }

    const enc = crypto.AES.encrypt(seedPhrase, newPass).toString();
    AsyncStorage.setItem(STORE_KEY.SEEDPHRASE, enc);
};
```

UPDATES

The issue has been fixed.

2.3.2. Malicious sites can inject JS code into trusted sites CRITICAL

Affected files:

- `src/screens/BrowserScreen/index.tsx`

With current implement of `convertToMsg` function, attackers can inject JS code into any trusted sites with `https://url#'+script+'`. For example: `https://pancakeswap.com#'+alert('injected')+'`. So if users access attackers link (maybe pretend to be a shortlink service) and the link redirect website with:

```
window.location = "https://pancakeswap.com#'+alert('injected')+'";
```

The website will be redirected to `pancakeswap.com` (user can see it's a trusted site) with malicious JS code injected. Attackers can use the malicious JS code to steal user's tokens instead of doing swap on `pancakeswap`.

```
function convertToMsg(data: any, url: string) {
    const js = `(function () {
    try {
      window.postMessage(${JSON.stringify(data)}, '${url}');
    } catch (e) {
      //Nothing to do
    }
  })()`;
    return js;
}

const onConnect = useCallback(
    ({ chain }) => {
        ...
        const js = convertToMsg(responseMsg, url);

        webView?.current?.injectJavaScript(js);
    },
    [curWallet, connected, setConnected, site, popup],
);
```

RECOMMENDATION

Converting `url` to JSON with `JSON.stringify`.

```
function convertToMsg(data: any, url: string) {
    const js = `(function () {
    try {
      window.postMessage(${JSON.stringify(data)}, ${JSON.stringify(url)}); // Fix here
    } catch (e) {
      //Nothing to do
    }
  })()`;
    return js;
}

const onConnect = useCallback(
    ({ chain }) => {
        ...
        const js = convertToMsg(responseMsg, url);

        webView?.current?.injectJavaScript(js);
    },
    [curWallet, connected, setConnected, site, popup],
);
```

UPDATES

The issue has been fixed.

2.3.3. Bypass lock mechanism HIGH

Affected files:

- `App.tsx`

The mobile app implements a lock mechanism which locks the app if it in the background for an amount of time.

```
const reloadData = useCallback(async (value: string) => {
    const lastTime = await AsyncStorage.getItem('lastTime');
    const currentTime = new Date().getTime();
    const rangeTime = currentTime - Number(lastTime || currentTime);
    const _1minutes = 1 * 60 * 1000;
    if (appState.current.match(/background/) && value === 'active') {
        if (rangeTime >= _1minutes) {
            Actions.reset('root');
        }
        appState.current = value;
    } else if (value.match(/background/) && appState.current === 'active') {
        AsyncStorage.setItem('lastTime', new Date().getTime() + '');
        appState.current = value;
    }
}, []);
```

The problem is `new Date().getTime()` of React Native is untrusted, attackers can modify system datetime to a past timestamp to bypass the time restriction and access the app without locking.

RECOMMENDATION

Instead of `new Date().getTime()` based time measuring, it is recommended to implement secure date time measuring using native module combination of time-synchronization from trusted source and local real-time clocks APIs like `SystemClock.elapsedRealtime` and `SystemClock.elapsedRealtimeNanos` on Android, `mach_continuous_time` on iOS. These return the elapsed time since the system was booted, including time when the device goes to deep sleep. This clock is guaranteed to be monotonic and continues to tick even when the CPU is in power saving mode, so is the recommended basis for general purpose interval timing.

You can take a look at *https://github.com/planado/react-native-elapsed-realtime*.

UPDATES

The issue has been fixed.

2.3.4. Sensitive data stored in `AsyncStorage` MEDIUM

Affected files:

- `src/store/wallet/functions.ts`

In `saveWallet` function, user's encrypted seed phrase is stored in the `AsyncStorage`. Even the seed phrase is encrypted before storing, `AsyncStorage` is not a safe place for storing sensitive data (in case the device is rooted/jailbroken, encrypted seed phrase could be stolen and bruteforce/dictionary attack to recover seed phrase).

```
export const saveWallet = async (wallet: ISaveWallet) => {
    setSetting('wallet', wallet.wallet);

    if (wallet.password && wallet.seedPhrase) {
        const enc = crypto.AES.encrypt(
            wallet.seedPhrase,
            wallet.password,
        ).toString();
        await AsyncStorage.setItem(STORE_KEY.SEEDPHRASE, enc);
        await AsyncStorage.setItem(
            STORE_KEY.WALLET_ADDRESS,
            wallet.wallet?.address || '',
        );
    }
    if (wallet.wallet) {
        global.wallet = wallet.wallet;
    }
};
```

RECOMMENDATION

- Using more secure storage like keychain/keystore to store sensitive data. You can take a look at *https://github.com/oblador/react-native-keychain*.
- Prevent/warning users from running the wallet on rooted/emulation devices.

UPDATES

The issue has been fixed.

2.3.5. Malicious sites can inject fake site to bypass connect popup MEDIUM

Affected files:

- `src/screens/BrowserScreen/index.tsx`

In `onMessage` function, current site is defined by `payload.payload` which is received from `postMessage GET_WEBVIEW_URL`. Any malicious sites can do a `postMessage` with fake `payload` to pretend to be trusted sites like:

```
window.ReactNativeWebView && window.ReactNativeWebView.postMessage(JSON.stringify(
    {
        type: 'GET_WEBVIEW_URL',
        payload: {
            url: 'https://pancakeswap.finance',
            icon: 'https://pancakeswap.finance/favicon.ico',
            title: 'pancakeswap',
            domain: `pancakeswap.finance`
        }
    }
```

Then they can request `eth_accounts` and bypass connect popup and get user's wallet address or request `eth_requestAccounts` to fake connection request popup from a trusted site.

```
window.ReactNativeWebView && window.ReactNativeWebView.postMessage(JSON.stringify(
    {
        type: 'GET_WEBVIEW_URL',
        payload: {
            url: location.href,
            icon: __getFavicon(),
            title: document.title,
            domain: (new URL(location.href)).hostname
        }
    }
))

  const onMessage = useCallback(
    async ({ nativeEvent }: WebViewMessageEvent) => {
        let payload = nativeEvent.data as any;
        payload = typeof payload === 'string' ? JSON.parse(payload) : payload;
        let js = '';
        if (payload.type) {
            switch (payload.type) {
                case 'GET_WEBVIEW_URL': {
                    setSite(payload.payload);
                    break;
                }
            }
            return;
        }
    ...
    });
```

RECOMMENDATION

Do not trust input from the current site JS. Get the `url` and `title` from `nativeEvent` instead of `postMessage` payload. You can get favicon with 3rd service like *https://www.google.com/s2/favicons?sz=64&domain_url=https://pancakeswap.finance*

UPDATES

The issue has been fixed.

2.3.6. Can't go back to menu after deleting contact LOW

Affected files:

- `src/store/contact/hook.ts`

After deleting contact, the route is replaced by `contact_screen` so there is no way to back to other screens because the navigation bar is hided in this screen.

```
const editContactItem = useCallback(
    (contact: IContactItem) => {
        const findIndex = contacts.findIndex(c => c.address === contact.address);
        if (findIndex === -1) {
            return global.showMessage(
                'This wallet address is not exist in your list contact',
            );
        }
        contacts[findIndex] = { ...contact };
        setListContact([...contacts]);
        global.showMessage('Edit contact success!');
        Actions.replace('contact_screen');
    },
    [contacts, setListContact],
);
```

RECOMMENDATION

Should `pop` instead of `replace`.

UPDATES

The issue has been fixed.

2.3.7. `value` should be sorted by name after `quoteOfToken` LOW

Affected files:

- `src/store/chain/functions.ts`

In `setChain` function, `value` should be sorted by name after `quoteOfToken` to avoid random order (in case `quoteOfToken` is the same for tokens) each time user access token list.

```
export const setChain = async (value: any[] = global.allTokens) => {
    ...
    value = value.sort((a: any, b: any) => b.quoteOfToken - a.quoteOfToken);
    setSetting('all_token', [...value]);
    global.allTokens = [...value];
    ...
};
```

RECOMMENDATION

Fixing the code like below.

```
export const setChain = async (value: any[] = global.allTokens) => {
    ...
    value = value.sort((a: any, b: any) => b.quoteOfToken - a.quoteOfToken ||
a.Token.localeCompare(b.Token)); // Fix here
    setSetting('all_token', [...value]);
    global.allTokens = [...value];
    ...
};
```

UPDATES

The issue has been fixed.

2.4. Possible enhancements

2.4.1. `getWalletFromPwd` should not `saveWallet` INFORMATIVE

Affected files:

- `src/store/wallet/functions.ts`

Function `getWalletFromPwd` should only return wallet, not `saveWallet` to avoid confusing and misuse.

```
export const getWalletFromPwd = async (
    pwd: string,
    getSeedphrase?: (v: string) => void,
) => {
    const enc = (await AsyncStorage.getItem('seedphrase_encrypted')) || '';
    const seedPhrase = crypto.AES.decrypt(enc, pwd).toString(crypto.enc.Utf8);
    try {
        const wallet = await getWalletFromSeedPhrase(seedPhrase);
        getSeedphrase?.(seedPhrase);
        saveWallet({ wallet }); // remove this

        return wallet;
    } catch (error) {
        return null;
    }
};
```

UPDATES

The issue has been acknowledged.

2.4.2. Hard coded `seedphrase_encrypted` INFORMATIVE

Affected files:

- `src/store/wallet/functions.ts`

Should use `STORE_KEY.SEEDPHRASE` instead of hard coded `seedphrase_encrypted`.

```
export const getWalletFromPwd = async (
    pwd: string,
    getSeedphrase?: (v: string) => void,
) => {
    const enc = (await AsyncStorage.getItem('seedphrase_encrypted')) || ''; // Use
STORE_KEY.SEEDPHRASE instead of hardcoded
    const seedPhrase = crypto.AES.decrypt(enc, pwd).toString(crypto.enc.Utf8);
    try {
        const wallet = await getWalletFromSeedPhrase(seedPhrase);
        getSeedphrase?.(seedPhrase);
        saveWallet({ wallet });
        return wallet;
    } catch (error) {
        return null;
    }
};
```

UPDATES

The issue has been fixed.

2.4.3. Typos INFORMATIVE

Affected files:

- `src/screens/Token/SendToken/index.tsx`

There are some typos in the code.

`Transaction submited, please wait for confirmation!` should be `Transaction submitted, please wait for confirmation`

`You dont have enought` should be `You dont have enough`

UPDATES

The issue has been fixed.

2.4.4. Webview JS injection INFORMATIVE

The webview interacts with the React Native app by injecting JS code and `postMessage`. It is acceptable but please note that website/dapp can inject malicious JS code to exploit the React Native app, so you have to carefully when handling payload from `postMessage` to avoid any issues like above.

UPDATES

The issue has been acknowledged.

2.4.5. Biometric issue on some android devices INFORMATIVE

The `react-native-keychain` is currently having problem with biometric on some android devices causing biometric to not work after rebooting the devices. We tested on emulator and can't log in using biometric after rebooting the device.

More information can be found here: *https://github.com/oblador/react-native-keychain/issues/318*

UPDATES

The issue has been acknowledged.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Aug 12, 2022*	Public Report	Verichains Lab

Table 3. Report versions history