verichains

SECURITY AUDIT OF

UTXO GLOBAL WALLET EXTENSION

UTXO GLOBAL

Public Report

Nov 15, 2024

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
BTC	Bitcoin (BTC) is the first decentralized cryptocurrency. Nodes in the peer-to-peer bitcoin network verify transactions through cryptography and record them in a public distributed ledger, called a blockchain, without central oversight.
CKB	Common Knowledge Base. Built on RISC-V and secured by Proof-of-Work, CKB is the most flexible and interoperable blockchain. It serves as the ultimate foundation of Nervos, a modular blockchain network built from the ground up to ensure outstanding security, decentralization, flexibility, and interoperability.
CCC	Common Chains Connector. CKBers' Codebase is a one-stop solution for the CKB JS/TS ecosystem development.

EXECUTIVE SUMMARY

This Security Audit Report prepared by Verichains Lab on Nov 15, 2024. We would like to thank UTXO Global for trusting Verichains Lab, delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the UTXO Global Wallet Extension. 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 major vulnerable issues in the source code, along with some recommendations. UTXO Global team has acknowledged and resolved these issues.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About UTXO Global

UTXO Global is a platform designed by core contributors from Nexum Labs to enhance digital asset management in the CKB ecosystem. With a focus on decentralized finance (DeFi), their team has developed tools that empower developers to leverage UTXO technology's scalability, privacy, and efficiency.

Their goal is to offer advanced, secure, and user-friendly solutions, from wallet extensions to APIs, enabling seamless development on CKB and other UTXO-based chains.

1.2. About UTXO Global Wallet Extension

UTXO Global is a revolutionary wallet for the CKB ecosystem. It delivers the robust security guarantees of the UTXO model while addressing its limitations in scalability and interoperability.

Backed by the CKB EcoFund, the UTXO Global Wallet Extension is a powerful and easy-to-use tool for managing digital assets within the CKB ecosystem. Fully compatible with CCC (Common Chain Connector), it's designed to make the transactions secure and simple.

Key Features.

- Manage Your Wallets: Easily oversee multiple wallets in one place.
- Accounts Made Easy: Manage different accounts without hassle.
- Support for Coins: Work with CKB and BTC seamlessly.
- Handle Tokens: Manage xUDT and sUDT tokens smoothly.
- NFT Management: Organize NFTs from RGB++, Spore, and more.
- Integrated with UTXOSwap: Easily swap tokens within the wallet for added convenience.

1.3. 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.

1.4. Audit Methodology

Verichains Lab's audit team mainly used the list below to check for Wallet Extension security:

- Transaction signature
- Transfer assets
- Transaction broadcast
- DApp communication
- Private Key/Mnemonic Phrase generation/destruction
- Private Key/Mnemonic Phrase secure storage
- Private Key/Mnemonic Phrase backup/restore
- Cryptography
- XSS
- Third-party JS
- HTTP Response Header
- Communication encryption
- Cross-domain transmission
- Access control
- Business design
- Architecture design

During the audit process, we also used tools for viewing, finding and verifying security issues of the extension wallet, and review the overall structure and design of the wallet extension to ensure it follows best practices and is resilient to attacks.

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.

SEVERITY LEVEL	DESCRIPTION
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.5. Disclaimer

UTXO Global acknowledges that the security services provided by Verichains, are conducted to the best of their professional abilities but cannot guarantee 100% coverage of all security vulnerabilities. UTXO Global understands and accepts that despite rigorous auditing, certain vulnerabilities may remain undetected. Therefore, UTXO Global agrees that Verichains shall not be held responsible or liable, and shall not be charged for any hacking incidents that occur due to security vulnerabilities not identified during the audit process.

1.6. Acceptance Minute

This final report served by Verichains to the UTXO Global will be considered an Acceptance Minute. Within 7 days, if no any further responses or reports is received from the UTXO Global, the final report will be considered fully accepted by the UTXO Global without the signature.

2. AUDIT RESULT

2.1. Overview

The initial review, conducted on October 14, 2024, involved a dedicated two-week effort to identify and document security issues in the UTXO Global Wallet Extension codebase. The audit was conducted by a team of security experts at Verichains Lab.

2.2. Findings

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

UTXO Global team has updated the code, according to Verichains's draft reports.

#	Issue	Severity	Status
1	User can be tricked into approving malicious approval requests	CRITICAL	Fixed
2	The `tabCheckin` internal method allows the site to manipulate session data	MEDIUM	Fixed
3	The `underline2Camelcase` function can be bypassed to call internal methods	HIGH	Fixed
4	Unsafe method is annotated as `SAFE`	HIGH	Fixed
5	Unconnected sites can read wallet information	HIGH	Fixed
6	Infinite loop bug in `FeeInput` component	INFORMATIVE	Fixed

2.2.1. User can be tricked into approving malicious approval requests CRITICAL

Affected files:

- src/background/services/notification.ts

All methods annotated with "APPROVAL" must go through an approval process. When the site requests approval, the `requestApproval` function is called, setting `this.approval` to the new approval request. However, the current implementation only supports one approval at a time, allowing the site to override the first approval by sending a second one.

```
// currently it only support one approval at the same time
requestApproval = async (
    data?: any,
    winProps?: OpenNotificationProps
): Promise<any> => {
    // We will just override the existing open approval with the new one coming in
    return new Promise((resolve, reject) => {
        this.approval = {
            data,
            resolve,
            reject,
        };

        // eslint-disable-next-line @typescript-eslint/no-floating-promises
        this.openNotification(winProps);
    });
};
```

In the `openNotification` function, if two approval requests are made, only the first one is shown to the user, but the actual approval has been overridden by the second request. This can lead to a situation where what the user sees does not match what they are actually approving.

```
openNotification = (winProps: OpenNotificationProps) => {
    if (this.isLocked) return;
    this.lock();
    if (this.notifiWindowId) {
        // eslint-disable-next-line @typescript-eslint/no-floating-promises
        remove(this.notifiWindowId);
        this.notifiWindowId = 0;
    }
    openNotification(winProps)
        .then((winId) => {
            this.notifiWindowId = winId;
        })
        .catch((e) => console.log(e));
};
```

This is a critical issue because it allows the site to deceive the user into signing malicious transactions or messages. The proof of concept (POC) below demonstrates this issue:

```
await utxoGlobal._request({method: "signMessage", params: {text: "aaa", address:
"tb1qdppg...rer0qd"}})
// wait for the popup to show
await utxoGlobal._request({method: "signMessage", params: {text: "bbb", address:
"tb1qdppg...rer0qd"}})
```

In this scenario, the popup displays the first message, but the user actually signs the second message.

UPDATES

- **Nov 13, 2024**: This issue has been acknowledged and fixed by the UTXO Global team by preventing consecutive approval requests.

2.2.2. The `tabCheckin` internal method allows the site to manipulate session data MEDIUM

Affected files:

- src/background/controllers/provider/internalMethod.ts

The `tabCheckin` internal method allows the site to manipulate session data, potentially leading to various attacks. For example, by calling the `tabCheckin` method and injecting a different `origin`, the site can modify the origin displayed on the approval screen.

```
export const tabCheckin = ({
  data: {
    params: { origin, name, icon },
  },
  session,
}) => {
  session.origin = origin;
  session.icon = icon;
  session.name = name;
};
```

Below is the POC to change the origin on the approval screen to the UTXO Swap site:

```
await utxoGlobal._request({method: "tabCheckin", params: {icon: "", name: "", origin:
"https://utxoswap.xyz"}})
await utxoGlobal.connect()
```

UPDATES

- **Nov 13, 2024**: This issue has been acknowledged and fixed by the UTXO Global team. However, the `name` and `icon` parameters are still injectable that may lead to potential attacks in the future.
- **Nov 15, 2024**: The `name` and `icon` parameters are now checked in the `tabCheckin` method.

2.2.3. The `underline2Camelcase` function can be bypassed to call internal methods HIGH

Affected files:

- src/background/utils/index.ts

The `flow` middleware chain guards method calls to provider controllers. The `underline2Camelcase` function is intended to filter method names containing an underscore, which should be used only for internal methods. However, the function can be bypassed by using double underscores in method names, allowing unapproved internal methods to be called. For instance, the input method `__methodName` transforms to `_methodName`.

```
// src/background/utils/index.ts
export const underline2Camelcase = (str: string) => {
    return str.replace(/_(.)/g, (m, p1) => p1.toUpperCase());
};

// src/background/controllers/provider/rpcFlow.ts
const flowContext = flow
// ...
.use(async (ctx, next) => {
    const {
        data: { method },
    } = ctx.request;
    ctx.mapMethod = underline2Camelcase(method);
    if (!ctx.providerController[ctx.mapMethod]) {
        throw ethErrors.rpc.methodNotFound({
            message: `method [${method}] doesn't has corresponding handler`,
            data: ctx.request.data,
        });
    }

    return next();
})
```

Although calling an underscore method without the "SAFE" annotation requires the site to be connected to the wallet, this issue remains exploitable.

For example, once the site connects to the wallet, it can call the `_switchNetwork` method as shown:

```
await utxoGlobal._request({ method: "__switchNetwork", params: { network: "btc" } });
```

This bypasses the approval check, enabling the internal method `_switchNetwork` to switch from testnet to mainnet without user consent.

UPDATES

- **Nov 13, 2024**: This issue has been acknowledged and fixed by blacklisting some internal methods, which is still not a secure solution. The `underline2Camelcase` function is still vulnerable to bypasses.
- **Nov 15, 2024**: Internal methods are now annotated with `INTERNAL` to block unauthorized access. However, adopting a whitelist approach rather than a blacklist would enhance security and should be considered for future implementation.

2.2.4. Unsafe method is annotated as `SAFE` HIGH

Affected files:

- src/background/controllers/provider/controller.ts

The `getAccounts` method is marked as `SAFE`, which may allow it to be called without authorization, enabling unconnected sites to access wallet information.

```
// src/background/controllers/provider/controller.ts
@Reflect.metadata("SAFE", true)
getAccounts = async () => {
    if (storageService.currentWallet === undefined) return undefined;
    return storageService.currentWallet.accounts[0].accounts.map(
        (account) => account.address
    );
};

// src/background/controllers/provider/rpcFlow.ts
const flowContext = flow
// ...
.use(async (ctx, next) => {
    const { mapMethod } = ctx;
    if (!Reflect.getMetadata("SAFE", ctx.providerController, mapMethod)) {
        if (!storageService.appState.isUnlocked) {
            ctx.request.requestedApproval = true;
            await notificationService.requestApproval({ lock: true });
        }
    }

    return next();
})
```

For instance, the following code allows unconnected sites to retrieve the wallet's accounts:

```
await utxoGlobal._request({ method: "getAccounts" });
```

RECOMMENDATION

Review all methods annotated as `SAFE` to ensure they are secure for calls from unconnected sites.

UPDATES

- **Nov 13, 2024**: The issue has been addressed by the UTXO Global team. Now, methods marked as `SAFE` require the site to connect to the wallet before they can be called.

2.2.5. Unconnected sites can read wallet information HIGH

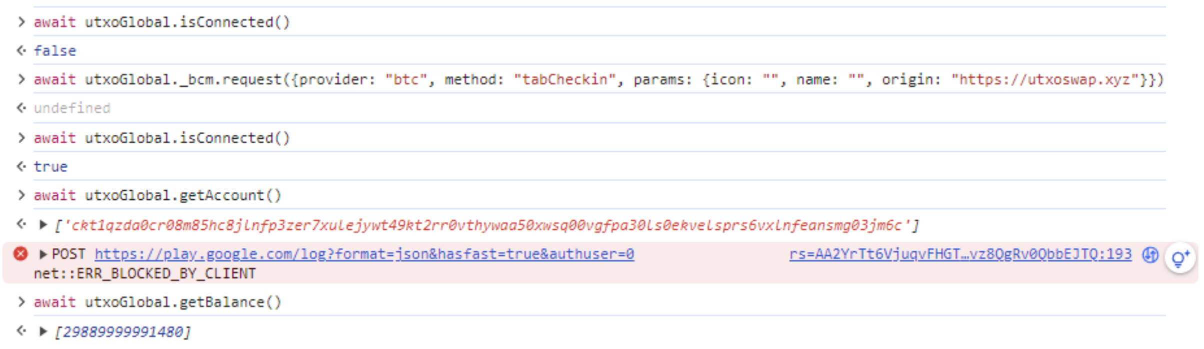
Some `internalMethod` in the `providerController` can be called without the user's approval. This can lead to a situation where unconnected sites can read wallet information without the user's consent.

```
// src/background/controllers/provider/index.ts
export default async (req) => {
    const {
        data: { method },
    } = req;

    if (internalMethod[method]) { // AUDIT: internalMethod can be called before approval
        return internalMethod[method](req);
    }

    const hasVault = (await storageService.getLocalValues()).enc !== undefined;
    if (!hasVault) {
        await notificationService.requestApproval(null, {
            height: 600,
            route: "/account/create-password",
        });
        throw ethErrors.provider.userRejectedRequest({
            message: "wallet must has at least one account",
        });
    }
    return rpcFlow(req);
};
```

For example, assume that the current wallet is already connected to the `https://utxoswap.xyz` website. By exploiting the `tabCheckin` method and altering the value of the `origin` parameter, we can maliciously trick the wallet into believing that the current site is the UTXO Swap site, thereby changing the connection status to `true`. This would allow the unconnected site to access wallet information, such as the user's address, balance, and more.



```
> await utxoGlobal.isConnected()
<· false
> await utxoGlobal._bcm.request({provider: "btc", method: "tabCheckin", params: {icon: "", name: "", origin: "https://utxoswap.xyz"}})
<· undefined
> await utxoGlobal.isConnected()
<· true
> await utxoGlobal.getAccount()
<· ▶ ['ckt1qzda0cr08m85hc8jlnfp3zer7xulejywt49kt2rr0vthywaa50xwsq00vgfpa30ls0ekvelsprs6vxlnfeansmg03jm6c']
⊗ ▶ POST https://play.google.com/log?format=json&hasfast=true&authuser=0                rs=AA2YrTt6VjuqvFHGT…vz8QgRv0QbbEJTQ:193
  net::ERR_BLOCKED_BY_CLIENT
> await utxoGlobal.getBalance()
<· ▶ [29889999991480]
```

UPDATES

- **Nov 13, 2024**: This issue has been acknowledged and fixed by the UTXO Global team.

2.2.6. Infinite loop bug in `FeeInput` component INFORMATIVE

Affected files:

- src/ui/pages/main/send/create-send/component.tsx
- src/ui/pages/main/send/create-send/fee-input/component.tsx

In the `CreateSend` component, the `FeeInput` component is responsible for handling fee input updates. However, there is a bug in the `FeeInput` component that can cause an infinite loop when the `onChange` callback is triggered.

```
// src/ui/pages/main/send/create-send/component.tsx
<div className={s.feeDiv}>
    {isCkbNetwork(
        getNetworkDataBySlug(currentNetwork.slug).network
    ) ? null : (
        <div className="form-field">
              <span className="input-span">
                {t("send.create_send.fee_label")}
              </span>
            <FeeInput
                onChange={(v) =>
                    setFormData((prev) => ({ ...prev, feeAmount: v }))
                } // AUDIT: new `onChange` callback is passed to the FeeInput
                value={formData.feeAmount}
            />
        </div>
    )}
    // ...
</div>
```

The `FeeInput` component uses the `useEffect` hook to trigger the `onChange` callback when the `onChange` prop changes. In this case, when the `onChange` callback is triggered, the `setFormData` function is called, updating the `formData` state. This leads to the `FeeInput` component re-rendering, which triggers the `onChange` callback again, causing an infinite loop.

```
// src/ui/pages/main/send/create-send/fee-input/component.tsx
const FeeInput: FC<Props> = ({ onChange, value }) => {
    const { feeRates } = useTransactionManagerContext();
    const [selected, setSelected] = useState<number>(feeRates?.slow ?? 10);

    useEffect(() => {
        if (selected !== 3) {
            onChange(selected); // AUDIT: Trigger `setFormData` callback
        }
    }, [selected, onChange]);
    // ...
```

UPDATES

- **Nov 13, 2024**: This issue has been acknowledged and fixed by the UTXO Global team.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Nov 13, 2024*	Public Report	Verichains Lab
1.1	*Nov 15, 2024*	Public Report	Verichains Lab

Table 2. Report versions history