Check Point Research detects vulnerability in the Everscale blockchain wallet, preventing cryptocurrency theft

April 25, 2022

Research By: Alexey Bukhteyev

Highlights

- Check Point Research (CPR) discovered a vulnerability in the web version of Ever Surf wallet, part of the Everscale blockchain ecosystem
- By exploiting the vulnerability, an attacker could have gained full control over victim's wallet
- After responsible disclosure, CPR collaborated with the Everscale teams, which acknowledged the vulnerability and released a new desktop version to replace the vulnerable web version

Background

Blockchain technology and decentralized applications (dAPPs) provide users with a number of advantages. For example, users can utilize the service without creating an account and it can be implemented as a single-page application written in JavaScript. This type of application does not require communication with a centralized infrastructure, such as a web server, and it can interact with the blockchain directly or by using a browser extension like Metamask.

In this case, the user is identified using keys that are stored only on a local machine inside a browser extension or a web wallet. If a decentralized application or a wallet stores sensitive data locally, it must ensure this data is reliably protected. In most cases, dAPPs run inside the browser and therefore may be vulnerable to attacks such as XSS.

This research describes the vulnerability found in the web version of Ever Surf, a wallet for the Everscale blockchain (formerly Free TON). By exploiting the vulnerability, it's possible to decrypt the private keys and seed phrases that are stored in the browser's local storage. In other words, attackers could gain full control over victim's wallets.

Responsible disclosure and collaboration with Everscale

CPR disclosed the vulnerability to Ever Surf developers who then released a desktop version that mitigates this vulnerability. The web version is now declared deprecated and should only be used for development purposes. Seed phrases from accounts that store real value in crypto should not be used in the web version of Ever Surf.

Vulnerability description

Ever Surf is a cross-platform messenger, a blockchain browser, and a crypto wallet for Everscale blockchain. It is written in React Native and is available in Google Play Market and Apple Store. There is also a web version of Ever Surf that runs on any platform.

Web version of Ever Surf

Figure 1 — Web version of Ever Surf

Surf doesn't require a backend to work with the blockchain because it implements a non-custodial wallet. This means the keys required to sign transactions are only stored on the user's device. Operations with the blockchain are performed entirely on the client's side. Therefore, like other non-custodial wallets, it doesn't have a registration using login names and passwords.

When users run the application for the first time, it suggests creating a new wallet. Surf generates a seed phrase and a pair of the public and the private keys. In addition, the user is prompted to create a 6-digit PIN code:

PIN code setup in Ever Surf

Figure 2 — PIN code setup in Ever Surf

This PIN is then used to log into the application and also to confirm transactions. Surf has a protection against brute-forcing the PIN code. When users enter an incorrect PIN code five times in a row, the application is temporary blocked and users are prevented from entering additional PIN codes:

Ever Surf prevents brute-forcing PIN codes

Figure 3 — Ever Surf prevents brute-forcing PIN codes

Even without such protection, the performance of manually brute-forcing PIN codes cannot be efficient enough to bother with in most cases. Therefore, to implement a real attack, attackers would need to extract the keystore from the application.

The created keys and the seed phrase are stored in the web browser's local storage. The local storage is a key-value storage accessed from JavaScript using the "localStorage" property. For example:

// Store value on the browser beyond the duration of the sessionlocalStorage.setItem('key', 'value');// Retrieve value (persists even after closing and re-opening the browser)alert(localStorage.getItem('key'));// Store value on the browser beyond the duration of the session localStorage.setItem('key', 'value'); // Retrieve value (persists even after closing and re-opening the browser) alert(localStorage.getItem('key'));// Store value on the browser beyond the duration of the session localStorage.setItem('key', 'value'); // Retrieve value (persists even after closing and re-opening the browser) alert(localStorage.getItem('key'));

Surf stores the keys and the seed phrase in the "surf.ton.wallet" key of the local storage in the following format:

Encrypted wallet data is stored in the browser's localStorage

Figure 4 — Encrypted wallet data is stored in the browser's localStorage

Local storage is not protected in web browsers. For example, in Firefox it is stored in a non-encrypted SQLite database in the file:

```
Mozilla\Firefox\Profiles\{profile_name}\webappstore.sqlite
```

Chrome stores the local storage in the non-encrypted form in a LevelDB database in the folder:

```
Google\Chrome\User Data\Default\Local Storage\leveldb\
```

This means that a human with physical access to the computer or any application or malware such as an infostealer can obtain this data.

In addition, localStorage can be accessed by a browser extension, which can then leak the stored data. To demonstrate this, we created a simple Chrome web extension with the following code:

alert(localStorage.getItem("surf.ton.wallet"));alert(localStorage.getItem("surf.ton.wallet"));alert(localStorage.getItem("surf.ton.wallet"));

If we open the Surf website after installing the extension, we see the message containing the encrypted keystore:

Extracting encrypted wallet data from localStorage using a browser extension

Figure 5 — Extracting encrypted wallet data from localStorage using a browser extension

Therefore, the browser's local storage cannot be considered secure enough.

The responsibility for the reliability of the stored data encryption rests entirely with the web application.

Websites that require registration, such as well-known social networks, may not rely only on the data stored in cookies or local storage. They may also check the web browser and the user's IP address, and require additional user verification in case of suspicious activities. This is impossible with Surf because the data required to control a user's wallet is stored entirely on their computer. The only protection for their funds is strong encryption.

Keystore encryption

Unfortunately, Surf is not open-source. Therefore, Check Point researchers had to analyze the minified JavaScript application bundle to understand the logic.

Although an attacker would be primarily interested in the keys decryption, it is enough to get the seed phrase which can be used to restore the keys. The application decrypts the seed phrase and shows it to the user after entering the PIN code:

Surf asks for the PIN code before showing the seed phrase

Figure 6 — Surf asks for the PIN code before showing the seed phrase

Therefore, to find where the decryption occurs, CPR first anchored to the function that fetches the seed phrase. In this function, the askForPassword function is called and receives the encrypted keystore.

The function responsible for fetching the seed phrase asks for the PIN code

Figure 7 — The function responsible for fetching the seed phrase asks for the PIN code

The askForPassword function performs the PIN code validation by calling the validatePassword function.

The askForPassword function reads and validates the PIN code

Figure 8 — The askForPassword function reads and validates the PIN code

The most important actions are performed throughout the PIN code validation. By calling the deriveKeyFromPasswordAndSalt function, Surf generates the derived key, which is used to decrypt the keystore. Then, in the isDerivedKeyCorrect function, Surf performs the decryption using nacl_secret_box_open:

Surf decrypts the keys using nacl_secret_box_open

Figure 9 — Surf decrypts the keys using nacl_secret_box_open.

These are the arguments for nacl_secret_box_open:

- encrypted — Encrypted string in base64-encoded form
- nonce — Hex-encoded nonce
- key — 64 symbols hex-encoded key derived from the PIN-code

The values “encrypted” and “nonce” are taken from the keystore (encStr and nonce):

{"encSeed": { "encStr":"6rjkbh88WXkrJWl4os3cjRzvZkQGlnV39U3YvLYUrM7yzf6h79XxXX/
VziNde2bkPeSnJZ11Yi4T8CFgPJMyI1W+EN0Yzw+FDdtTq5gaKHjk5nF60CSHZ4qKXObIQ7zBbped", "nonce":"2Tdr9FFP0RcpjecwcS8XtcC5M2/
FRY+I"}}{"encSeed": { "encStr":"6rjkbh88WXkrJWl4os3cjRzvZkQGlnV39U3YvLYUrM7yzf6h79XxXX/
VziNde2bkPeSnJZ11Yi4T8CFgPJMyI1W+EN0Yzw+FDdtTq5gaKHjk5nF60CSHZ4qKXObIQ7zBbped", "nonce":"2Tdr9FFP0RcpjecwcS8XtcC5M2/
FRY+I" }}{"encSeed": { "encStr":"6rjkbh88WXkrJWl4os3cjRzvZkQGlnV39U3YvLYUrM7yzf6h79XxXX/
VziNde2bkPeSnJZ11Yi4T8CFgPJMyI1W+EN0Yzw+FDdtTq5gaKHjk5nF60CSHZ4qKXObIQ7zBbped", "nonce":"2Tdr9FFP0RcpjecwcS8XtcC5M2/
FRY+I" }}

The next step is to understand how the derived key appears. Inside the deriveKeyFromPasswordAndSalt, Surf derives the decryption key from the PIN code using the Scrypt key derivation function with the following arguments:

Surf derives the keystore decryption key from the entered PIN code and salt using Scrypt

Figure 10 — Surf derives the keystore decryption key from the entered PIN code and salt using Scrypt

These are the arguments for scrypt:

- password — Base64-encoded PIN-code
- salt — Salt value in the base64-encoded form

As the password is entered by the user, the only missing piece is the salt value. The salt value is calculated on the client’s device and has two roles:

- It is not stored in the keystore. This should mean decryption is impossible if the encrypted keys were leaked and the attacker doesn’t know how to calculate the salt.
- Protects the resulting value against rainbow table attacks.

Salt is calculated as SHA256 hash of one of the values returned by the DeviceInfo.getUniqueId function of the react-native-device-info package:

Surf generates the salt by hashing the value returned by DeviceInfo.getUniqueId()

Figure 11 — Surf generates the salt by hashing the value returned by DeviceInfo.getUniqueId()

Nevertheless, there is another option. The salt can be calculated as SHA256 of the window.process.argv.slice(-1) value. In a web browser, the window.process is undefined. Therefore, DeviceInfo.getUniqueId is always used.

After checking react-native-device-info documentation, researchers see that it is not supported in a web browser:

getUniqueId() is not supported in a web browser

Figure 12 — getUniqueId() is not supported in a web browser

In a web browser, this function always returns the value “unknown”:

getUniqueId always returns the value “unknown”

Therefore, the salt is always the same and can be easily calculated as `sha256(“unknown”)` and is equal to “`b23a6a8439c0dde5515893e7c90c1e3233b8616e634470f20dc4928bcf3609bc`”:

The salt is always equal to sha256(“unknown”)

Figure 13 — The salt is always equal to sha256(“unknown”)

Attack

CPR roughly re-implemented the key derivation and keystore decryption in NodeJS and performed a brute-force attack on the PIN code.

This resulted in a performance of 95 passwords per second on 4-core Intel Core i7 CPU. Although this is not a very high speed, it is sufficient for the attack on a 6-digit PIN code. In the worst case scenario, checking 10^6 possible variants means the entire attack takes approximately 175 minutes.

For our experiment, we created a new key in Surf and dumped the keystore from the browser’s local storage:

{ "encSeed": { "encStr":"6rjkbh88WXkrJWl4os3cjRzvZkQGlnV39U3YvLYUrM7yzf6h79XxXX/
VziNde2bkPeSnJZ11Yi4T8CFgPJMyI1W+EN0Yzw+FDdtTq5gaKHjk5nF60CSHZ4qKXObIQ7zBbped", "nonce":"2Tdr9FFP0RcpjecwcS8XtcC5M2/
FRY+I" }}{ "encSeed": { "encStr":"6rjkbh88WXkrJWl4os3cjRzvZkQGlnV39U3YvLYUrM7yzf6h79XxXX/
VziNde2bkPeSnJZ11Yi4T8CFgPJMyI1W+EN0Yzw+FDdtTq5gaKHjk5nF60CSHZ4qKXObIQ7zBbped", "nonce":"2Tdr9FFP0RcpjecwcS8XtcC5M2/
FRY+I" } }{ "encSeed": { "encStr":"6rjkbh88WXkrJWl4os3cjRzvZkQGlnV39U3YvLYUrM7yzf6h79XxXX/
VziNde2bkPeSnJZ11Yi4T8CFgPJMyI1W+EN0Yzw+FDdtTq5gaKHjk5nF60CSHZ4qKXObIQ7zBbped", "nonce":"2Tdr9FFP0RcpjecwcS8XtcC5M2/
FRY+I" } }

In our case, the attack took 38 minutes. At the end, we got the derived key and decrypted the seed phrase that can be used to restore the keys on another device:

PIN code brute force results

Figure 14 — PIN code brute force results

If cybercriminals perform scalable attacks using infostealers, they may collect huge numbers of encrypted keys and seed phrases. If they try to decrypt them one by one, it may take a significantly long time.

However, since the salt for all encrypted data from different users is the same, the key derivation function Scrypt can be called only once per checked PIN code. While the Scrypt function is computationally intensive, the nacl_secret_box_open function that is used for the decryption is very fast. This allows brute-forcing the PIN codes for multiple wallets without a significant performance impact.

Compared to brute-forcing PIN codes for one encrypted seed phrase when the performance was 95 PIN codes per second, when we tried to force the PIN code for 100 encrypted seed phrases simultaneously, we got a performance of 79 passwords per second.

Tips to stay safe

We would like to remind you that blockchain transactions are irreversible. In blockchain, unlike a bank, you cannot block a stolen card or dispute a transaction. If the keys for your wallet are stolen, your crypto funds can become easy prey for cybercriminals, and no one can help to return your money back. To prevent theft of the keys, we recommend:

- Do not follow suspicious links especially if they received from strangers.

- Keep your OS and anti-virus software updated.
- Do not download software and browser extensions from unverified sources.

Conclusion

As the browser's local storage is unprotected, the data stored there must be securely encrypted. Despite the fact that Surf uses reliable cryptographic libraries for the key derivation and the encryption, the sensitive data in the web version of Surf doesn't appear to have adequate protection.

CPR's PoC shows that the combination of the issues found presents several attack vectors that can lead to an attacker obtaining private keys and seed phrases in clear text, which can then be used to gain full control over the victim's wallet.

Check Point Researchers collaborated with the Ever Surf teams which decided that the vulnerable web version will be deprecated and replaced with a desktop version. Ever Surf published an article about the new version in which they stated:

"Check Point Research conducted their own independent research about the security status of the Surf web version and found out its weakness. We followed this report, checked everything and ensured that the vulnerability exists. Our web version cannot provide a secure use of password-based KDF because of an inability to provide a unique salt such as device ID for that platform. In simple terms, that means there is a theoretical way to get access to your wallet and assets on it.

Therefore, we needed to take quick steps to eliminate it so as not to put our users at risk. It became clear that we cannot longer postpone the release of the Surf desktop app. Our basic idea was to wrap the existing web application into an Electron packager and pass a unique machineID inside in order to use its hash as a salt for the key derivation.

Thus we could save the familiar interface and everything you are used to in the web version. We didn't want to ruin your user experience so you can still unlock the app with a short PIN instead of memorizing strong and heavy passwords. In the long run, desktop version gives options compared to the web version."