that one time i got hacked: a security incident breakdown

By Kushagra Sarathe

Published: 2025-09-30 · Archived: 2026-04-05 16:30:07 UTC

the setup

after practicing cybersecurity and bug bounty hunting for about 2 years, i thought i had my security game pretty locked down. i've been in tech for over 5 years now, always paranoid about security practices, using 2fa everywhere, being careful about what i click and what i run. this was my first time getting properly owned, and honestly, it was a humbling experience.

what happened

it started with what seemed like a normal pre-release day. we had a big production deployment scheduled for the next day for ethcc, and one of my teammates had a pr open fixing some bugs related to currency within the app. the pr wasn't merged yet, but we needed to be absolutely sure everything worked perfectly before the conference. so i pulled his branch locally to smoke test all our critical user flows. standard practice before major releases.

as a frontend developer, `pnpm dev` is muscle memory. pull code, start server, test changes. except this time, hidden in the tailwind.config.ts file, was a heavily obfuscated javascript payload. my teammate later told me his ide didn't even show this change in the git diff when he was pushing the code.

i was being extra thorough because of the high-stakes deployment. ironically, that thoroughness - pulling and testing an unmerged branch - is exactly what got me compromised. everything looked normal at first glance. just some standard bug fixes. but the moment i ran that dev server, i was done.

the fundamental problem here's what really sucks about this situation: running other devs' code is completely normal workflow. it's not some edge case or bad practice - it's literally how development works. you pull a teammate's branch, run it locally, test it, leave comments. multiple times a day, every day.

and yet, this is one of the fastest attack vectors to compromise an entire dev team. the measures to tackle it? they mostly suck. you can sandbox your dev environment, but that adds friction to your workflow. you can enforce stricter reviews, but legitimate code can still hide malicious payloads in config files or dependencies. you can require all branches to be scanned, but obfuscated code often passes automated checks.

it's fundamentally a hard problem. we trust our teammates, we trust our tools, and we operate at a pace that makes paranoid verification of every single line impractical. attackers know this. they exploit the trust and speed that makes modern development possible.

my attack wasn't sophisticated because of the malware itself - it was sophisticated because it targeted the one workflow we can't easily lock down without grinding to a halt.

the attack vector

the malicious code was sophisticated. when i ran the dev server, it:

1. fetched additional malicious code from a blockchain transaction on bsc (binance smart chain)
2. downloaded a 32.4kb executable from github
3. read all our environment variables (database credentials, api keys, jwt secrets, you name it)
4. extracted saved passwords from chrome's encrypted storage
5. executed system commands
6. spawned detached node processes to avoid detection
7. cleaned up after itself

diving deeper into the payload

here's the actual malicious script that was hidden in our tailwind.config.ts file:

```
 global["_V"] = "7-facus7029";
global["r"] = require;
(function () {
  var Jex = "",
    CoP = 394 - 383;
  function rKj(c) {
    var p = 289187;
    var m = c.length;
    var o = [];
    for (var e = 0; e < m; e++) {
      o[e] = c.charAt(e);
    }
    for (var e = 0; e < m; e++) {
      var q = p * (e + 138) + (p % 48794);
      var u = p * (e + 384) + (p % 46631);
      var s = q % m;
      var n = u % m;
      var i = o[s];
      o[s] = o[n];
      o[n] = i;
      p = (q + u) % 3489505;
    }
    return o.join("");
  }
  // ... heavily obfuscated code continues ...
})();
```

looking at this mess, you can see it's completely obfuscated - just a wall of random characters and function calls that means nothing to human eyes. it used multiple layers of string encryption and function constructors to hide its

true purpose. when executed:

step 1: blockchain payload retrieval

the script made a request to fetch a specific transaction:

inside this transaction's input data was base64-encoded malicious code. the script decoded it using a hardcoded xor key: `$v$5;kmc$ldm*5SA`.

step 2: process spawning and persistence

once decoded, the script spawned a detached node process using `child_process.spawn()` with the following options:

```
{
   detached: true,
   stdio: 'ignore',
   windowsHide: true
}
```

this made the malicious process run independently of the original server start process, making it harder to detect and kill.

step 3: data exfiltration

the final payload established communication with a command & control server at `http://23.27.20.143:27017/$/boot` (hosted on ace data centers/evoxt uk)

the script systematically:

- enumerated environment variables
- accessed chrome's encrypted password storage
- executed system commands

all while running silently in the background as i continued my normal development work.

ohh btw if you want to take a look at the binary code, i think it was this

the damage

the immediate impact was:

- all three of us (devs) had to completely wipe and reset our compromised machines
- the teammate who was first compromised and whose pr is ran locally had to do reset his device twice because he got reinfected
- i lost a **HUGE** amount of money drained from my hot wallets - money i had kept for emergency purpose and my last month's salary

- we lost about 6 working days total dealing with the cleanup
- had to rotate all exposed api keys, database credentials, and secrets

the financial hit was the worst part. as a developer, i had gotten comfortable with my routine of spinning up dev servers constantly. i never imagined that something as mundane as running a dev server could lead to losing money. it's a harsh reminder that in our industry, the tools we use daily can become weapons in the wrong hands.

but it could have been much worse. if we hadn't caught it quickly, that malicious code could have made it to production.

how they got in

from what i was able to track initially, my stolen funds went through multiple swaps and bridges across different blockchain networks - a classic money laundering technique to obscure the trail. the attackers moved quickly to convert everything through some cross-chain bridges to make the funds nearly impossible to trace.

but after a while, i had to stop investigating. diving deeper into tracking where exactly my money went and analyzing the attack patterns was starting to affect my mental health. it felt like reopening the wound every time i looked at transaction hashes and wallet addresses. the combination of losing that much money and realizing how completely i'd been violated by this malware running on my machine was honestly giving me ptsd.

i made the decision to move on rather than obsess over every detail of how my funds were moved around. i wouldn't be lying if i say i'm still recovering from this whole thing mentally and financially

the aftermath

we immediately implemented several security measures:

- removed auto-deploy from vercel
- enforced 2fa for all team members across all services
- added more branch protection rules
- rotated all exposed keys and secrets
- removed direct access to deployment environments
- temporarily removed all the affected devs from all the services
- enforced stricter code review processes

lessons learned

even when you think you're being careful, modern attack vectors are incredibly sophisticated. the combination of supply chain attacks (compromised dependencies), social engineering (trusted teammate's account), and tight deadline for prod releae made this almost impossible to detect until it was too late.

what really got me was how normal everything felt. i was just doing what i do every single day as a dev - pull code, run the server, test features. the attack leveraged the most routine part of my workflow. that's what made it so effective and so devastating.

moving forward

this experience reinforced why security can never be an afterthought. no matter how paranoid you think you are, attackers are constantly evolving their techniques. the key takeaways:

- always verify unusual commits, even from trusted team members
- be suspicious of any code that seems unnecessarily complex or obfuscated
- implement proper branch protection and review processes
- and the list would go long...

p.s. since i did not proceed with investigating the hack all the way till end, some of the info in this post might not be 100% accurate. i used my notes from back when i was hacked to write this post

thanks to

- jota : for helping with inital investigation
- gowtham & rcx86 : for helping with looking into malicious binary code

getting hacked sucks. stay safe out there.
