Hunting for SSRF vulnerabilities in Next.js targets

BY BLACKBIRD-EU · SEPTEMBER 28, 2025 · LAST UPDATED ON OCTOBER 12, 2025

Next.js is a powerful open-source React framework that enables developers to build fast, interactive, and SEO-friendly web applications. With almost 13 million weekly downloads via NPM, and the framework being complex by nature, it makes it a prime target for unfriendly intruders.

In this article, we'll be diving deeper into the most common server-side request forgery vulnerabilities in targets extensively utilizing Next.js.

Let's dive in!

Next.js: The modern web application framework

Next.js is a powerful React-based framework offering developers several utilities and components to build interactive web applications and APIs. Furthermore, it also supports server-side rendering, API, middleware logic, routing, image optimization, to name a few.

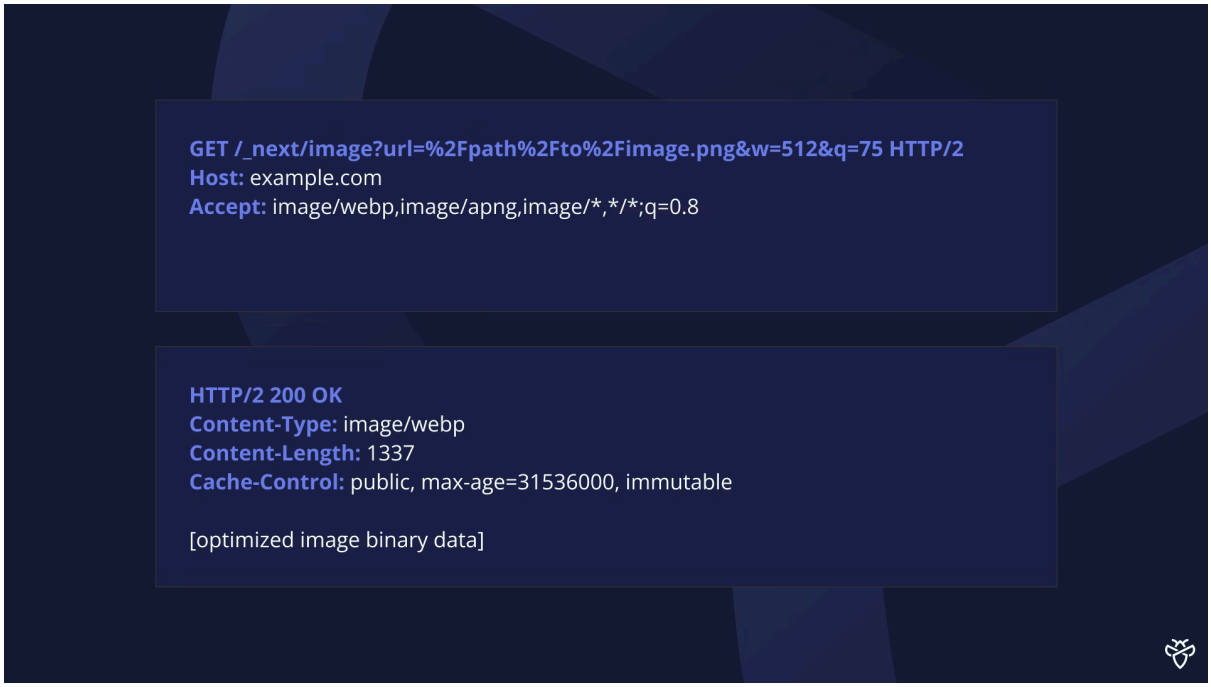
Its vast functionality creates the right environment for security misconfigurations to arise. In the following sections, we will document 3 methods that allow us to induce a vulnerable Next.js application component to make outbound HTTP requests to arbitrary hosts, a vulnerability type that's commonly referred to as a server-side request forgery.

> **Server-side request forgery (SSRF)**
>
> Learn more about identifying and exploiting server-side request forgery (SSRF) vulnerabilities in bug bounty targets.
>
> https://www.intigriti.com/researchers/blog/hacking-tools/ssrf-a-complete-guide-to-exploiting-advanced-ssrf-vulnerabilities

SSRF in Next.js Image component

Image optimization is a must in web application development. Loading large, unoptimized image assets anywhere in your website will often introduce a negative SEO impact. That's why Next.js recommends using the Next.js Image component, a server-side component that does all the optimization and caching work for you.

For this component to work, all the optimization and image processing have to be done on the server-side. Whenever you include the Image component in your Next.js project, which is often done by default, you expose an additional API endpoint located at `/_next/image`. Next time, when you include an image, the location of that image will be forwarded to the API endpoint, perform all the optimization work, and finally return the optimized image:



Next.js Image component API

As you can see in the image above, the `url` parameter looks like the perfect place to test for server-side request forgery vulnerabilities. Especially since Next.js supports the wildcard scope configuration option:

```
/** @type {import('next').NextConfig} */
const nextConfig = {
  images: {
    remotePatterns: [
      {
        protocol: 'https',
        hostname: '**',
      },
      {
        protocol: 'http',
        hostname: '**',
      }
    ],
  },
}

module.exports = nextConfig
```

Developers do this to allow loading images from any host, unknowingly introducing a new attack vector. As this security misconfiguration allows us to make HTTP requests on behalf of the vulnerable server, a typical (blind) server-side request forgery vulnerability.

In the event no wildcard is specified, you should still attempt to probe for potential whitelisted hosts (such as CDNs) as the underlying Image component, which is responsible for pulling the initial version of the image, follows redirects by default. A simple open URL redirect vulnerability could, in practice, open up a way to reach any arbitrary host.

Weaponizing SSRF vulnerabilities

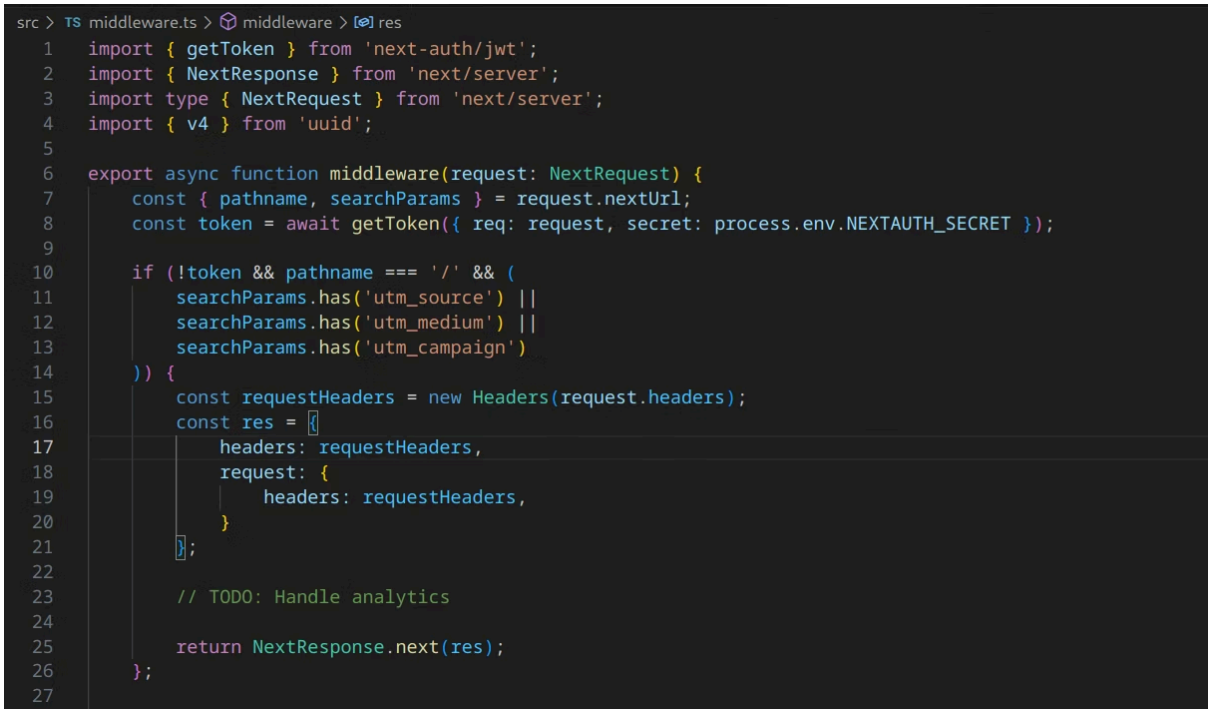
Server-side request forgery vulnerabilities can often be further weaponized to reach protected components or networks, disclose sensitive information, and even achieve remote code execution. Learn more about weaponizing (blind) SSRF vulnerabilities in our comprehensive article:

https://www.intigriti.com/researchers/blog/hacking-tools/ssrf-a-complete-guide-to-exploiting-advanced-ssrf-vulnerabilities

SSRF in Next.js Middleware (CVE-2025-57822)

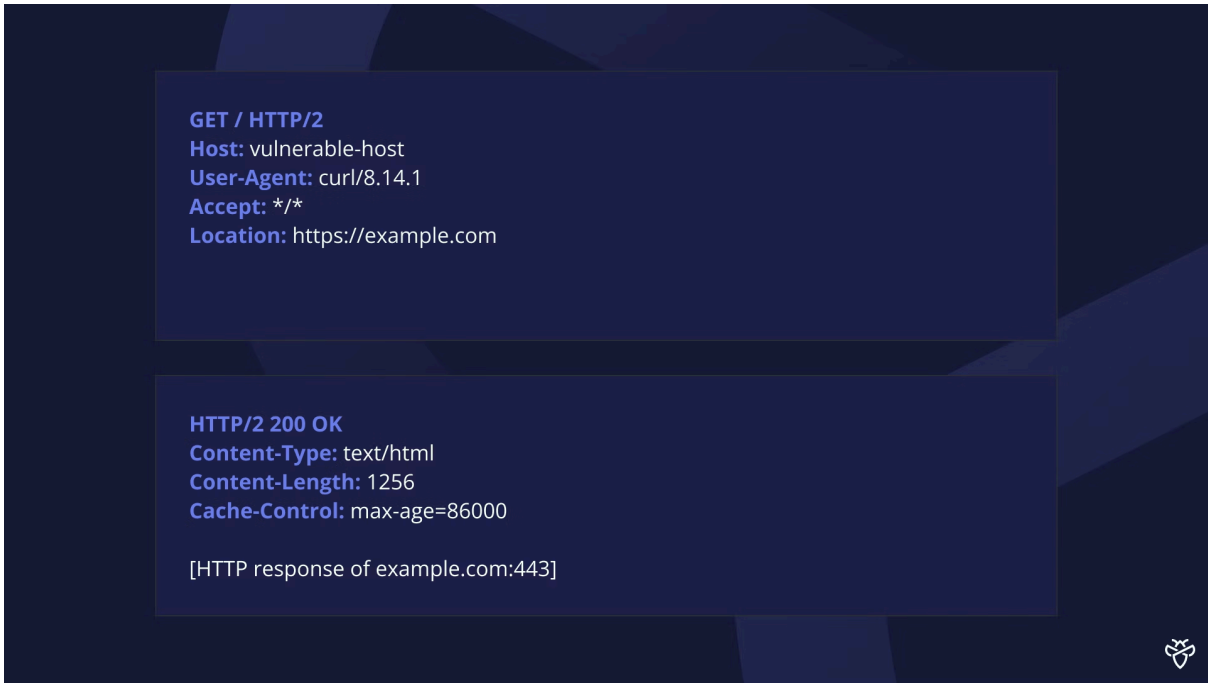
Next.js Middleware allows developers to run code before responses are returned to the client. This can be useful when implementing server-side redirects, authentication, and authorization checks based on application logic. However, in some cases, developers can unknowingly introduce a server-side request forgery vulnerability by accidentally passing unsanitized user input to be evaluated by a Middleware method.

To further understand the underlying issue and explain why it's also a commonly occurring one, we must take a look at a simple example of a vulnerable implementation. Here's an example of a vulnerable middleware implementation in Next.js from one of our previously hosted Intigriti CTF challenges that keeps track of UTM parameters:



Next.js Middleware misconfiguration introduces an SSRF

When you look closely, the developer seems to have passed the entirety of the object, including unsanitized request headers, to the `next()` method. This will make the `next()` method evaluate all headers before returning the response, including checking for possible internal redirects initiated by the `Location` header. For instance, if we were to make a request to the vulnerable application with an arbitrary `Location` header, we would, in practice, make Next.js load our requested resource specified in the `Location` header.



Next.js Middleware SSRF (CVE-2025-57822) proof of concept

This undocumented vulnerability went unnoticed for years before being acknowledged by the vendor and assigned CVE-2025-57822. Thanks to security researcher Dominik Prodinger, we learned through his research that over 5,000 potentially affected hosts have been identified on the internet. When testing Next.js applications, it is advisable to include checks for this specific CVE.

CTF: Exploiting SSRF via Next.JS Middleware

Intigriti 0825 CTF challenge featured CVE-2025-57822. Read our official write-up to learn how further exploitation can even result in remote code execution:

https://www.intigriti.com/researchers/blog/hacking-tools/catflix-ctf-ssrf-nextjs-middleware

SSRF via Next.js Server Actions (CVE-2024-34351)

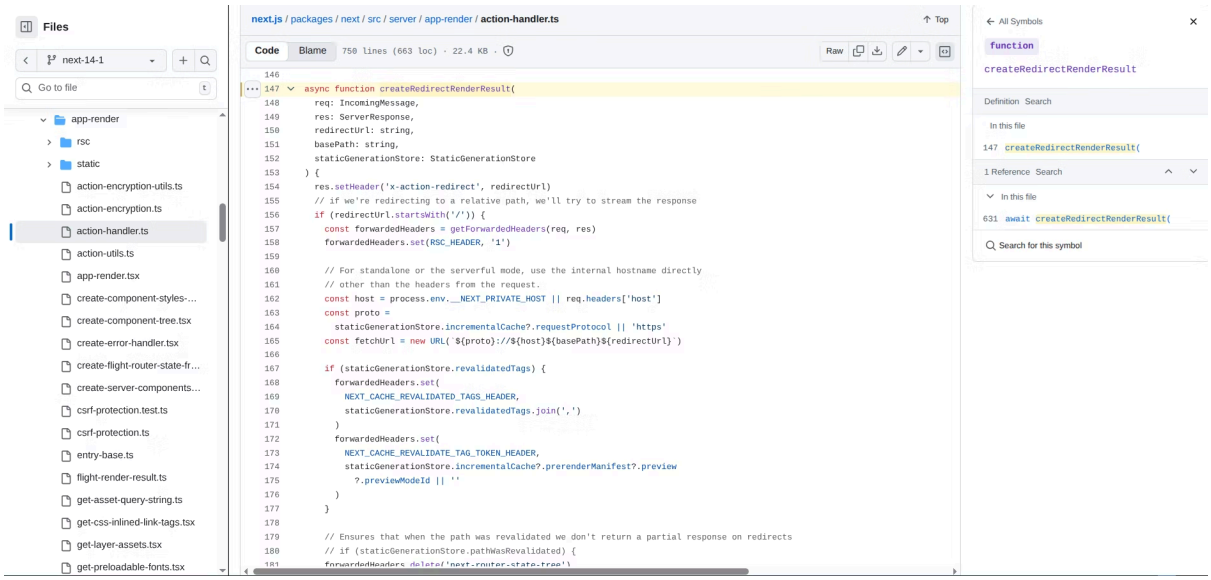
Next.js Server Actions allow developers to define asynchronous functions that run server-side and can be invoked directly from React components, enabling seamless server-side data fetching, mutations, and form handling without the need for custom API routes or extra backend boilerplate.

Originally discovered by researchers at Assetnote, an incorrect implementation of Server Actions can possibly lead to server-side request forgery. For your target to be vulnerable, it must meet the following conditions:

- Next.js (version 14.1.1 or lower) must be running in a self-hosted environment
- Your Next.js target needs to make use of Server Actions
- The Server Action must perform a redirect to a relative path

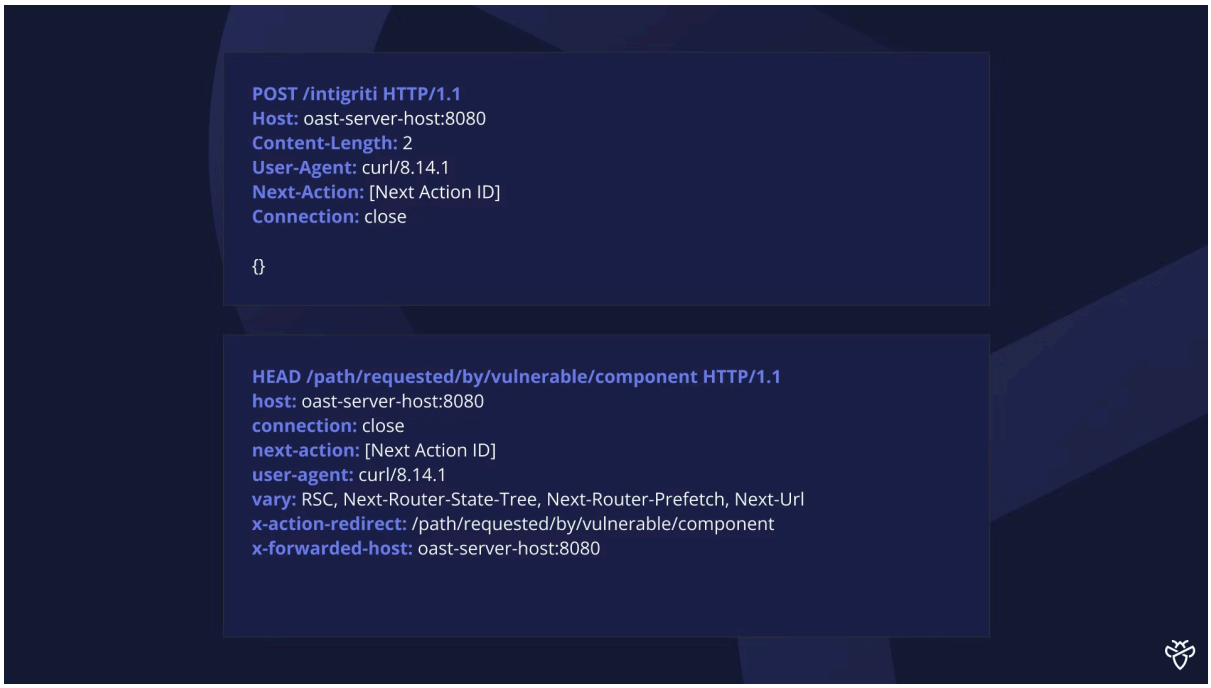
The last requirement is essential because Next.js v14.1.1 and earlier versions stream responses back to clients using a specific validation process. The framework first sends a `HEAD` request to the host

specified in the incoming request's host header, validates the response's content-type, and then proceeds with the `GET` request to retrieve the actual content:



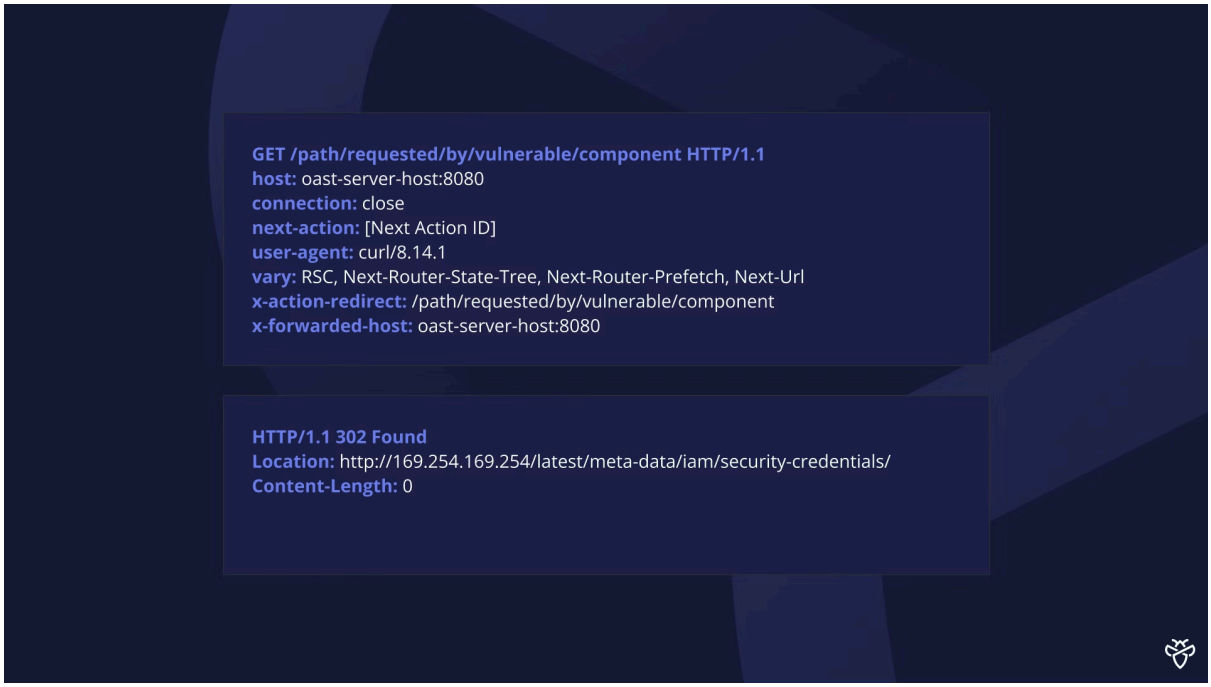
Next.js Server Actions Handler (v14.1.1 and earlier)

The exploitation phase requires configuring a custom OAST server that responds with the appropriate `content-type` header to bypass the initial validation check.



The first request will trigger the vulnerable Next.js Server Action component to initiate a HEAD HTTP request to our OAST server specified in the Host header.

When the server subsequently receives the `GET` request, we respond with a `302` HTTP redirect containing our target resource in the `Location` header, such as the AWS Metadata endpoint URI.



Our OAST server is configured to respond with the supported content-type response header, which in turn initiates the GET request. When we receive the GET request, we respond with a 302 status code.

Conclusion

Next.js is a complex web application development framework where simple mistakes often introduce security flaws, including SSRF vulnerabilities. In this article, we've documented several server-side request forgery (SSRF) vulnerabilities in Next.js targets.

So, you've just learned something new about exploiting SSRF vulnerabilities in Next.js targets... Right now, it's time to put your skills to the test! Browse through our 70+ public bug bounty programs on Intigriti, and who knows, maybe you're next to earn a bounty with us today!

START HACKING ON INTIGRITI TODAY