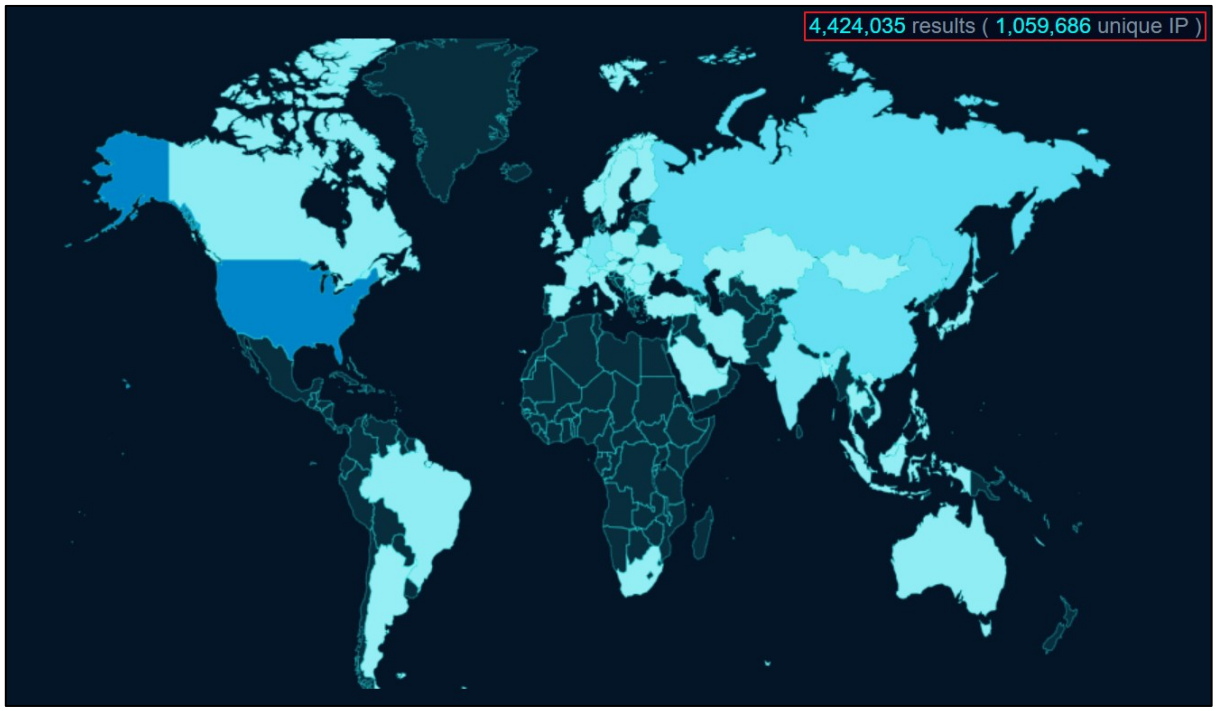
Research & Technique

Bypass Vulnerability in Next.js Middleware (CVE-2025-29927)

■ Introduction

Next.js is an open-source web framework based on Node.js that supports Server-Side Rendering (SSR)[1] and Static Site Generation (SSG)[2]. The globally popular JavaScript library React mentions Next.js as a recommended toolchain in its official documentation. An OSINT-based analysis of publicly available data showed that, as of April 15, 2025, Next.js powers approximately 4.42 million websites worldwide, including sites in the United States, Russia, and Germany.



Source: fofa.info

Figure 1. Usage Statistics of Next.js

[1] SSR (Server-Side Rendering): A web communication method in which the server generates all data and sends it to the client, and the client interprets that data to render the website.

[2] SSG (Static Site Generation): A method in which page HTML is generated at build time and reused for each request.

On March 21, 2025, a vulnerability in Next.js middleware[3] bypass, identified as CVE-2025-29927, was disclosed. This vulnerability arises from the exploitation of Next.js's internal logic, which checks whether the middleware has already been executed using specific header values. By manipulating these header values, attackers are capable of circumventing the middleware, and if the authentication process is implemented through middleware, this bypass enables access to restricted pages. Given the extensive utilization of Next.js as a web framework, it is imperative to conduct a thorough examination to determine if one's systems are susceptible to this vulnerability.

[3] middleware: concept in the request–response cycle that processes incoming requests before they reach their destination and processes responses before, they are sent.

■ Attack Scenario

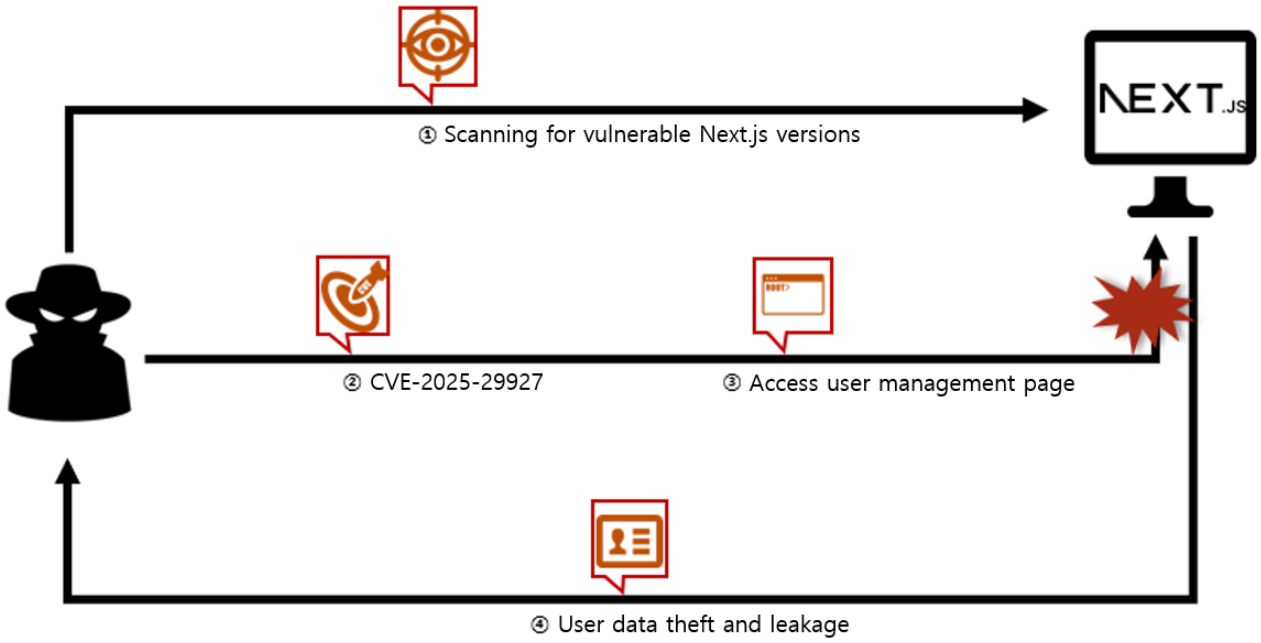


Figure 2. Attack Scenario for CVE-2025-29927

① The attacker scans servers running vulnerable versions of Next.js.
② Exploiting CVE-2025-29927, the attacker bypasses authentication.
③ Bypassing authentication, the attacker accesses the victim's user management page.
④ From there, the attacker harvests and exfiltrates large volumes of personal data.

■ Affected Software Versions

The software versions vulnerable to CVE-2025-29927 are as follows.

Software Component	Vulnerable Version
Next.js	Version prior to v15.2.3 Version prior to v14.2.25 Version prior to v13.5.9 Version prior to v12.3.5 All v11 Version

■ Configuration Information of the Test Environment

A test environment was established to scrutinize the operational process of CVE-2025-29927.

Identifier	Details
Victim	Next.js v15.1.7 (192.168.0.3)
Attacker	Kali Linux (192.168.216.133)

■ Vulnerability Assessment

Step 1. Configuration of the Environment

The victim's PC is configured with the Next.js v15.1.7 environment, and a test environment is established to replicate the vulnerability. Detailed configuration methods can be accessed in the GitHub repository provided below.

• URL: https://github.com/EQSTLab/CVE-2025-29927

```
# Clone a GitHub repository
git clone https://github.com/EQSTLab/CVE-2025-29927

# Move to the directory, then build and run the Docker image
cd next15
docker build -t nextjs .
docker run -p 3000:3000 --rm -it nextjs
```

Upon accessing the victim's address via a web browser, one verifies whether the Next.js server is operating correctly.

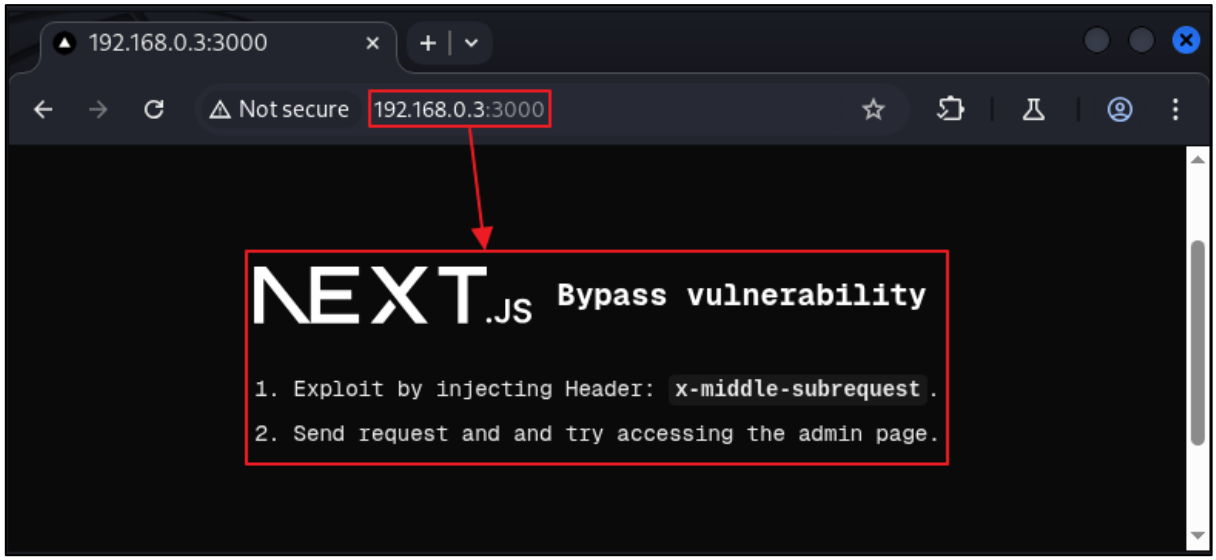


Figure 3. Verification of a Vulnerable Next.js Environment Setup

Step 2. Vulnerability Testing

The Next.js server is configured to block unauthorized access to the /admin endpoint.

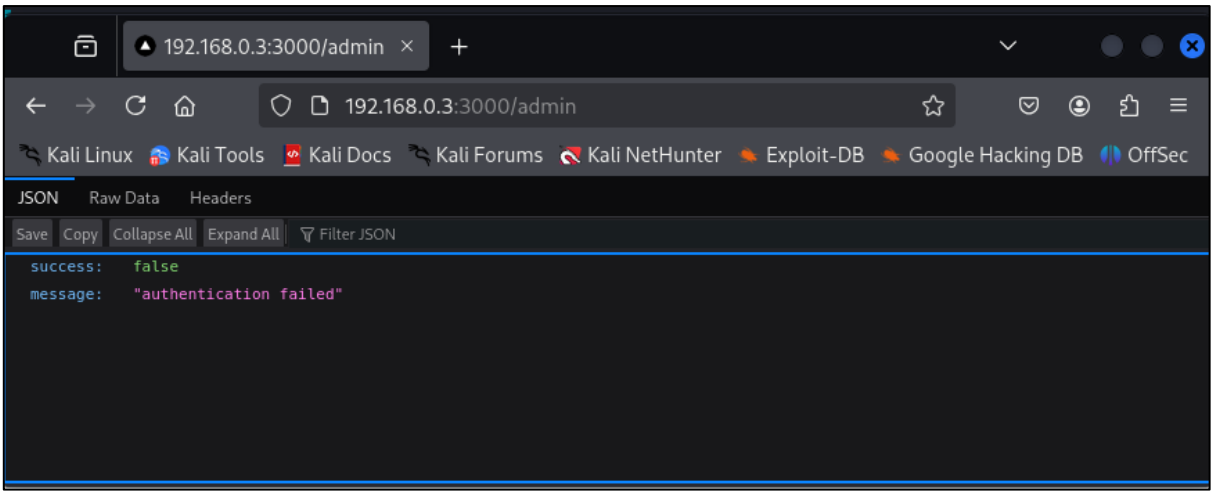


Figure 4. Blocking Access to the /admin Endpoint

However, the addition of the following headers to the request circumvents the middleware authentication process.

```
x-middleware-subrequest: middleware: middleware: middleware: middleware: middleware
```

When a request containing this header is transmitted, the middleware is treated as having already been executed, thereby enabling access to the /admin endpoint.

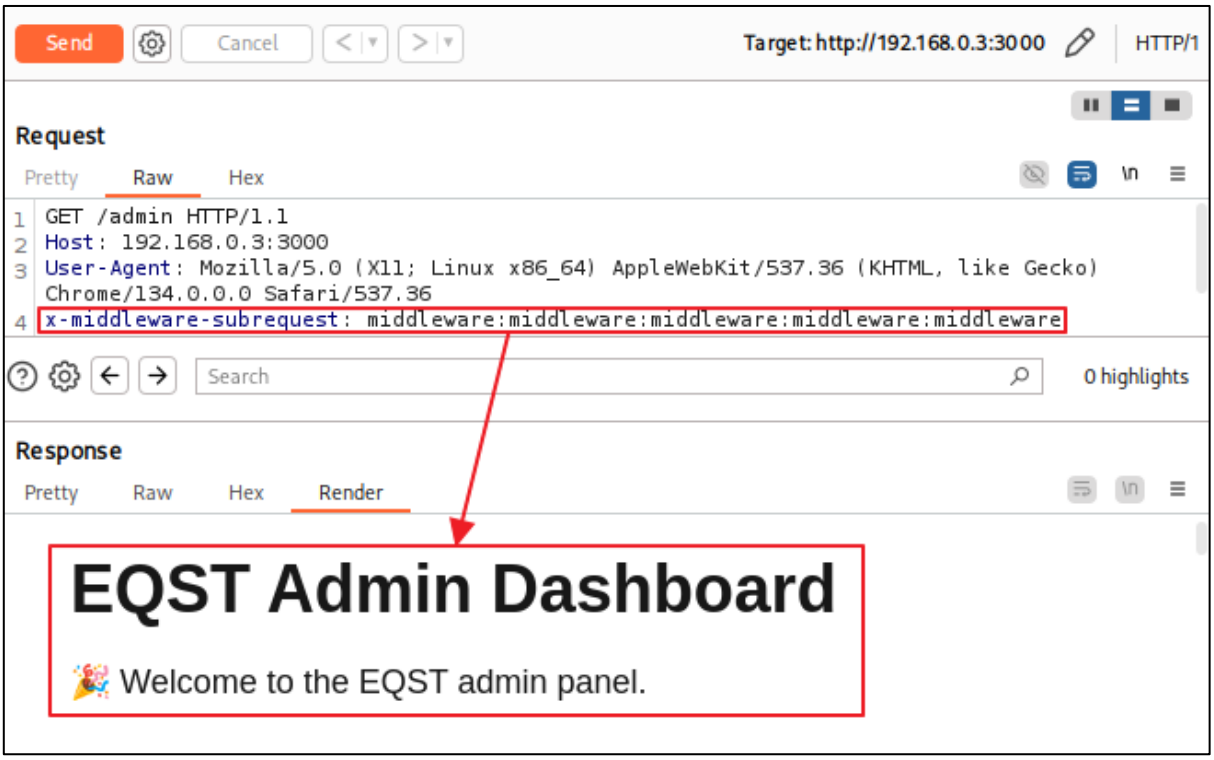


Figure 5. Verification of Middleware Authentication Bypass

■ Detailed Analysis of Vulnerabilities

In the detailed analysis of vulnerabilities, the report methodically elucidates the CVE-2025-29927 vulnerability, from its genesis to the process of authentication circumvention. Step 1 introduces the concept of middleware in Next.js, along with the characteristics unique to each version. Step 2 describes the vulnerabilities observed in the middleware implementations of each version and explains the techniques employed to circumvent authentication by exploiting these vulnerabilities.

Step 1. Implementation of Next.js Middleware

1) Characteristics of Next.js Middleware

Middleware acts as an intermediary layer that processes client requests before they reach the application within the request-response cycle, and is capable of performing additional processing prior to the transmission of responses. In Next.js, this functionality enables the implementation of pre-request validation, header rewriting, and redirections, and is defined through either a middleware.ts or middleware.js file. The middleware file may be positioned alongside the app and pages folders in the top-level directory of the project, or it can be included within the src directory.

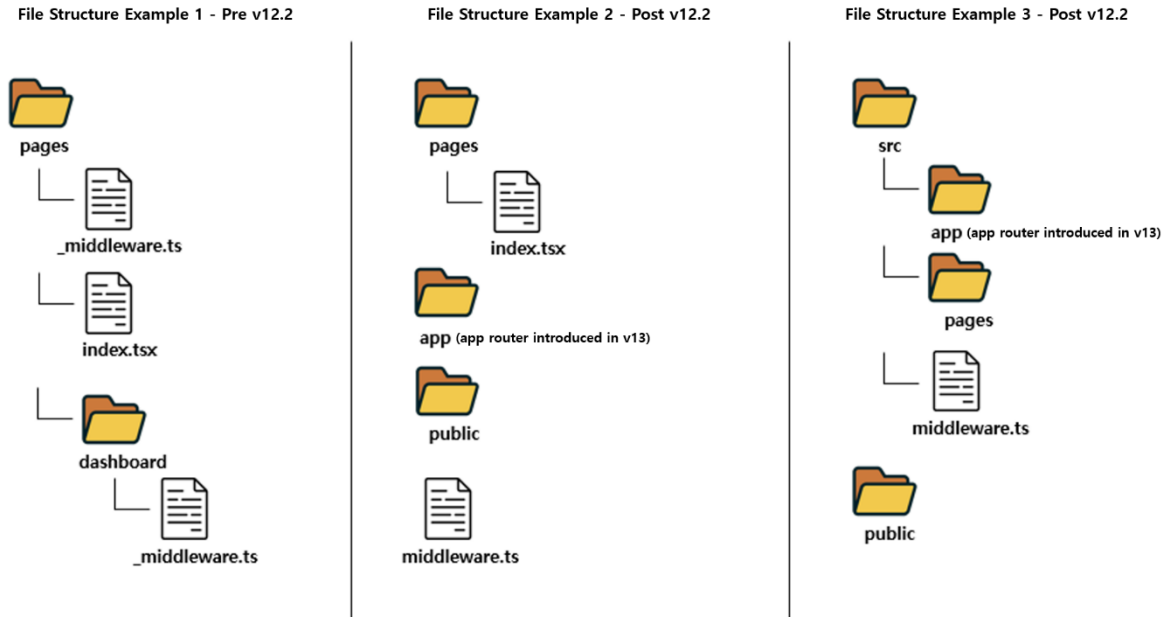


Figure 6. Example of Middleware Placement by File Structure

2) Characteristics of Middleware by Next.js Version

Since version 12.2, Next.js has upgraded and patched its middleware, resulting in significant changes to the manner in which middleware is utilized.

(1) Versions below 12.2

In versions prior to Next.js v12.2, it was possible to define middleware in any directory. In such instances, upon receiving a request, the server sequentially executes the middleware, commencing from the top-level directory and progressing to the subdirectories.

For instance, if middleware is defined in the /pages directory, it is applied to all requests for index.tsx, about.tsx, and teams.tsx within that directory.

```
- package.json
- /pages
    └ _middleware.ts       # Will run on all routes under /pages
    └ index.tsx
    └ about.tsx
    └ teams.tsx
```

Figure 7. The _middleware file within the /pages directory

Additionally, if middleware is defined in the directories /pages, /pages/about, and /pages/about/teams, upon receiving a request, the middleware is executed in a hierarchical order from the upper to the lower levels of the directory structure.

```
- package.json
- /pages
    └ _middleware.ts              # will run first
    └ index.tsx
    └ /about
        └ _middleware.ts          # will run second
        └ about.tsx
        └ /teams
            └ _middleware.ts      # will run third
            └ teams.tsx
```

Figure 8. Execution Sequence of Nested Middleware Files

(2) Versions subsequent to v12.2

From version 12.2 of Next.js, the use of underscores (_) in middleware filenames has been discontinued, and the filenames have been changed to either middleware.ts or middleware.js. Furthermore, the method of defining middleware in a nested manner within directories is no longer supported, and only a single middleware file can be placed in a structure parallel to the project root or the pages directory. As the application of middleware at the directory level becomes untenable, it is necessary to utilize matcher configurations to apply middleware to specific paths. For instance, to apply middleware to paths that begin with /admin, one must specify the corresponding pattern in the matcher as follows.

```
1   import { NextResponse } from 'next/server'
2   import type { NextRequest } from 'next/server'
3
4   export function middleware(request: NextRequest) {
5       return NextResponse.rewrite(new URL('/about-2',  request.url))
6   }
7
8   // Supports both a single string value or an array of matchers
9   export const config = {
10      matcher: ['/about/:path', '/dashboard/:path'],
11  }
```

Figure 9. Example Code for Matcher Configuration

Alternatively, conditional statements can be embedded within the middleware to bifurcate operations based on the request path. This approach proves particularly beneficial in scenarios requiring complex conditions that are challenging to manage solely through matcher configurations.

```
1   import { NextResponse } from 'next/server'
2   import type { NextRequest } from 'next/server'
3
4   export function middleware(request: NextRequest) {
5       if (request.nextUrl.pathname.startsWith('/about')) {
6           return NextResponse.rewrite(new URL('/about-2', request.url))
7       }
8
9       if (request.nextUrl.pathname.startsWith('/dashboard')) {
10          return NextResponse.rewrite(new URL('/dashboard/user', request.url))
11      }
12  }
```

Figure 10. Example Code for Conditional Branching

Step 2. Implementation of Authentication in Next.js Middleware and Methods to Circumvent by Version

1) Implementation of Authentication Using Middleware

Among the authentication implementation methods provided by Next.js, there is also an approach that utilizes middleware. For instance, the example below illustrates code that redirects to specific paths based on user permissions. Such a method proves particularly beneficial in scenarios requiring swift processing, such as concealing UI elements or controlling access based on permissions and roles.

```
1   import { NextRequest, NextResponse } from 'next/server'
2
3   // Protected and Public Routes
4   const protectedRoutes = ['/dashboard']
5   const publicRoutes = ['/login', '/signup', '/']
6
7   export default async function middleware(req: NextRequest) {
8     // Some authentication logic
9     return NextResponse.next()
10  }
11
12  // Routes Middleware should not run on
13  export const config = {
14    matcher: ['/(?!api|_next/static|_next/image|.*\\.png$).*'],
15  }
```

Figure 11. Example of Implementing Authentication in Next.js Using Middleware

After implementing authentication through middleware, it can be observed that attempts to access the /admin endpoint without authentication are subsequently blocked, as demonstrated below.

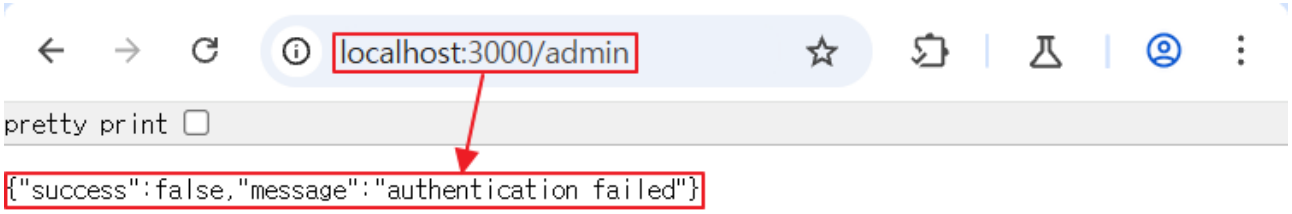


Figure 12. Verification Process of x-middleware-request

2) Versions below 12.2

Next.js internally utilizes the x-middleware-subrequest header to ascertain whether the middleware has already been executed for a particular request. This logic, as of version 12.0.1, can be verified within the /packages/next/server/next-server.ts file. The principal flow of the code is as follows.

```
630     const subreq = params.request.headers[`x-middleware-subrequest`]
631     const subrequests = typeof subreq === 'string' ? subreq.split(':') : []
632     const allHeaders = new Headers()
633     let result: FetchEventResult | null = null
634
...
650
651       if (subrequests.includes(middlewareInfo.name)) {
652         result = {
653           response: NextResponse.next(),
654           waitUntil: Promise.resolve(),
655         }
656         continue
657       }
```

Figure 13. x-middleware-request Verification Process

① The values transmitted via the x-middleware-subrequest header are segmented based on the colon (:) delimiter and are subsequently arrayed.

② Check whether the value of the middlewareInfo.name variable exists in the configured array.

③ If the specified value exists, the NextResponse.next() function is employed to bypass the middleware, thereby advancing to the subsequent processing stage to receive the response from the requested path.

At this juncture, the value of the middlewareInfo.name variable is loaded through the getMiddlewareInfo function located within the /packages/next/server/next-server.ts file. This function is tasked with retrieving middleware information corresponding to the requested path, and based on this information, it evaluates whether to execute the middleware by inspecting the x-middleware-subrequest header.

```
698         const middlewareInfo = getMiddlewareInfo({
699           dev: this.renderOpts.dev,
700           distDir: this.distDir,
701           page: middleware.page,
702           serverless: this._isLikeServerless,
703         })
```

Figure 14. Retrieving middlewareInfo through getMiddlewareInfo

The getMiddlewareInfo function is implemented in the /packages/next/server/require.ts file and returns information pertaining to the middleware corresponding to the specified path in the form of an object.

```
83  export function getMiddlewareInfo(params: {
84    dev?: boolean
85    distDir: string
86    page: string
87    serverless: boolean
88  }): { name: string; paths: string[] } {
89    const serverBuildPath = join(
90      params.distDir,
91      params.serverless && !params.dev ? SERVERLESS_DIRECTORY : SERVER_DIRECTORY
92    )
93
94    const middlewareManifest: MiddlewareManifest = require(join(
95      serverBuildPath,
96      MIDDLEWARE_MANIFEST
97    ))
...
```

Figure 15. Invocation of MIDDLEWARE_MANIFEST within getMiddlewareInfo

This logic references the .next/server/middleware-manifest.json file located within the .next directory generated subsequent to the build of Next.js, thereby importing the middleware information.

```
11  "middleware": {
12    "/": {
13      "env": [],
14      "wasm": [],
15      "files": [
16        "server/middleware-runtime.js",
17        "server/pages/_middleware.js"
18      ],
19      "name": "pages/_middleware",
20      "page": "/",
21      "regexp": "^/(?!_next).*$"
22    }
23  },
24  "version": 1
25  }
```

Figure 16. Information on middleware name within middleware-manifest.json

The "name" key in the middleware-manifest.json file denotes the location path of each middleware. Consequently, the middlewareInfo.name value ultimately becomes pages/_middleware. This is consistently verified when checked through debugging tools during execution.

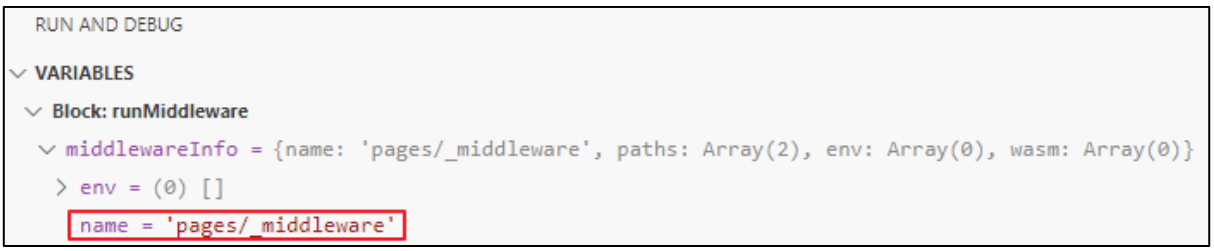


Figure 17. The Value of middlewareInfo.name as Verified through the Debugger

Thus, when the x-middleware-subrequest header value is set to pages/_middleware, the request is treated as having already passed through the middleware. Consequently, the middleware does not execute, and the authentication procedures implemented within can be bypassed, resulting in a vulnerability.

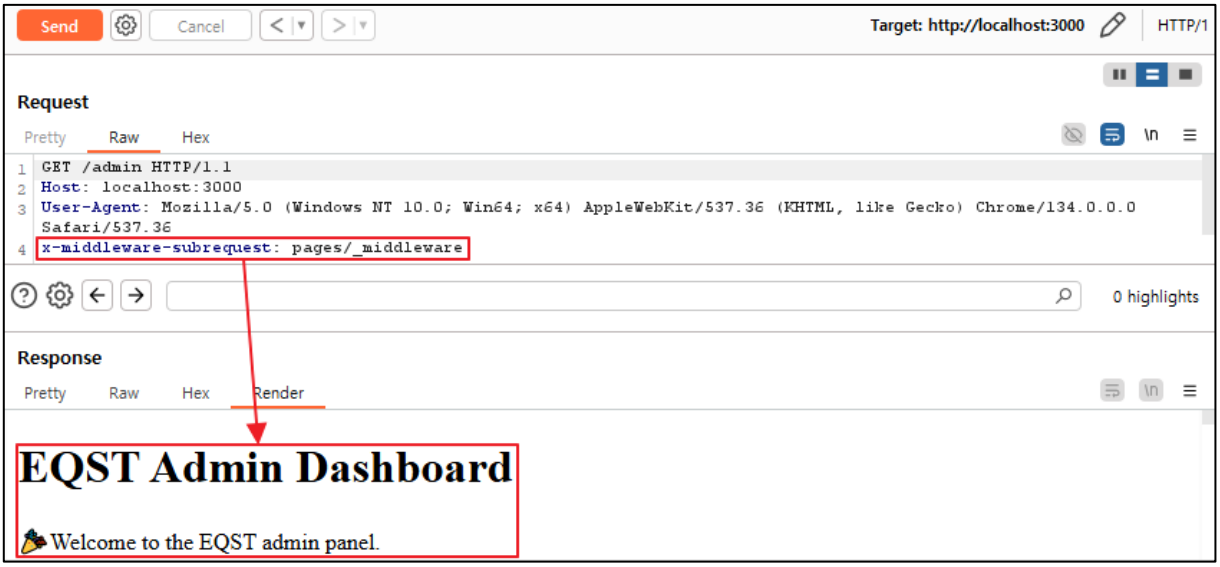


Figure 18. Verification of Middleware Authentication Process Bypass

As elucidated in Step 1, versions prior to v12.2 permit the individual definition of middleware for each directory. In such instances, by entering the path of each middleware defined under the 'pages' directory into the x-middleware-subrequest header value, it becomes feasible to circumvent the authentication process. For instance, as demonstrated in Figure 8, specifying either pages/about/_middleware or pages/about/teams/_middleware in the header enables the bypassing of the respective middleware.

3) Versions above 12.2 and below 14.2

The logic pertaining to x-middleware-subrequest has remained unaltered; consequently, as was the case previously, setting the middleware path in this header still facilitates the bypassing of authentication. From version 12.2 onwards, the naming convention for middleware files has been modified to either middleware.ts or middleware.js, and their location has been standardized to a directory parallel to, rather than inside, the pages directory. Thus, the middleware path has transitioned from the former pages/_middleware to simply middleware.

Such modifications can be observed in the .next/server/middleware-manifest.json file, through which it is discernible that the path of the middleware has been altered from pages/_middleware to middleware.

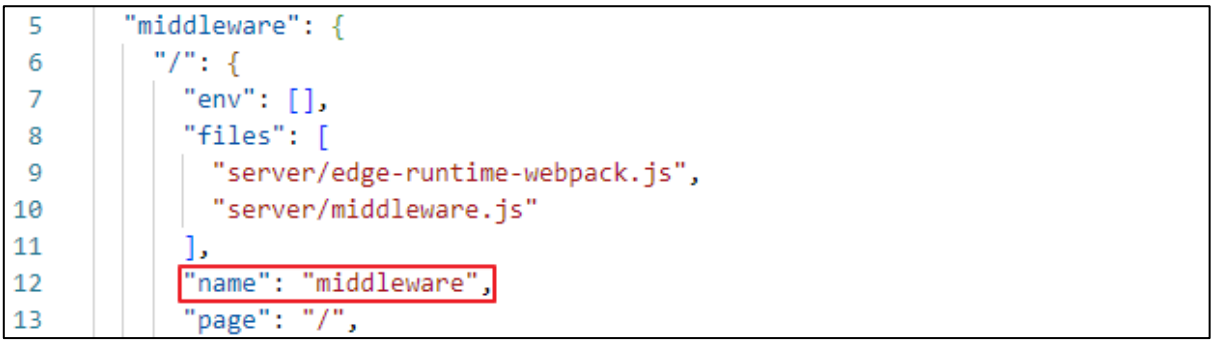


Figure 19. Information on middleware name within middleware-manifest.json

Similarly to the case of Next.js v12.2 mentioned above, by inspecting the values in execution through the debugger, it can be ascertained that the value in question is indeed middleware.

Figure 20. The Value of middlewareInfo.name as Verified through the Debugger

Similarly, by setting the value of the x-middleware-subrequest header to middleware, it is possible to circumvent the authentication process.

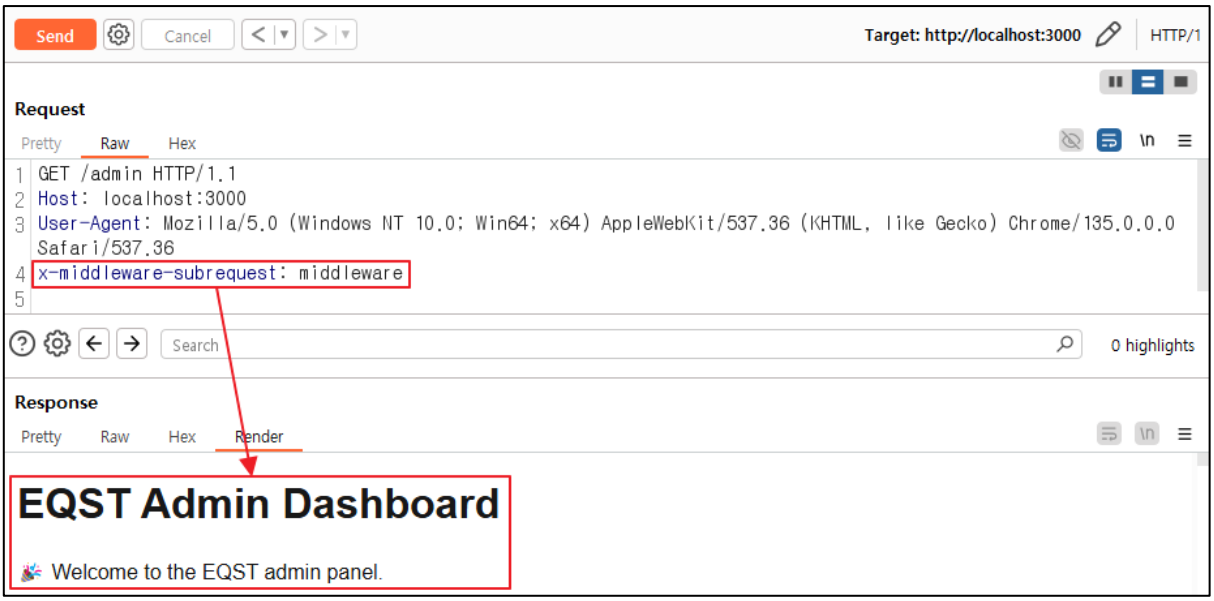


Figure 21. Verification of Bypassing the Middleware Authentication Process

4) Versions subsequent to v14.2

From version 14.2 of Next.js, the code has been modified such that if the value in the x-middleware-subrequest header of the middleware includes a middleware path that repeats beyond a certain number of times based on a colon (:), the request is engineered to bypass the middleware.
This can be verified through the code in /packages/next/src/server/web/sandbox/sandbox.ts.

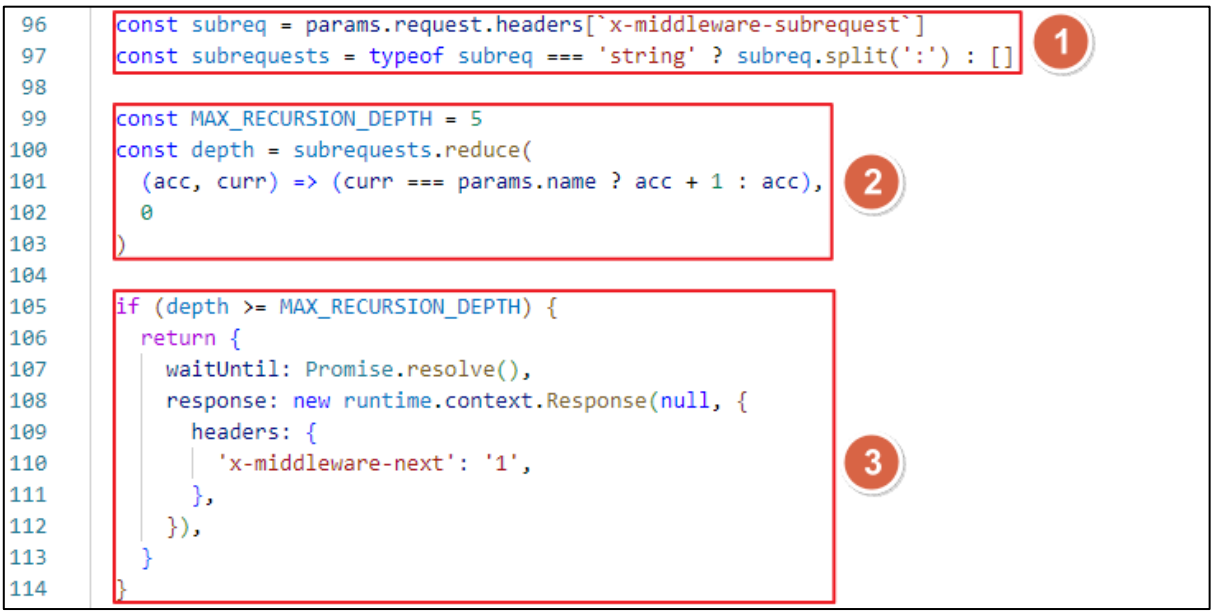


Figure 22. Modified x-middleware-request Verification Process

① The values transmitted via the x-middleware-subrequest header are segmented based on the colon (:) delimiter and subsequently structured into an array.
② The configured array is scrutinized to ascertain the number of occurrences of the value of the variable params.name.
③ If the value of the variable params.name is repeated more than five times, the middleware is bypassed, and the response from the requested path is received.

In this context, params.name corresponds to the previously mentioned middleware.name, a congruence verifiable through the .next/server/middleware-manifest.json file and subsequent debugging.

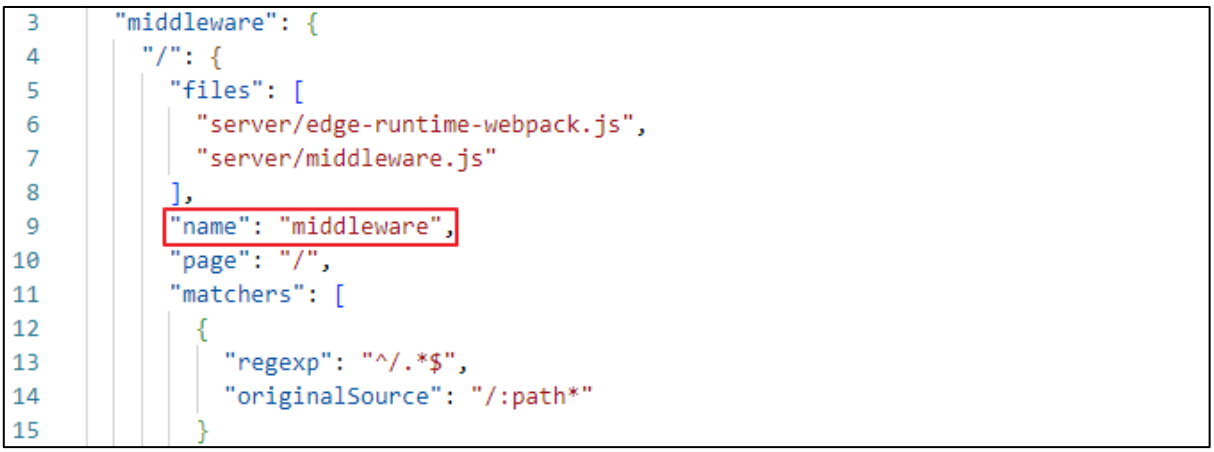


Figure 23. Information on middleware name within middleware-manifest.json

Figure 24. The Value of params.name as Verified Through the Debugger

Consequently, subsequent to version 14.2, it is requisite that the middleware path be reiterated no fewer than five times, delineated by colons (:), as in middleware:middleware:middleware:middleware:middleware. By entering such a value into the x-middleware-subrequest header, one can circumvent the middleware verification process.

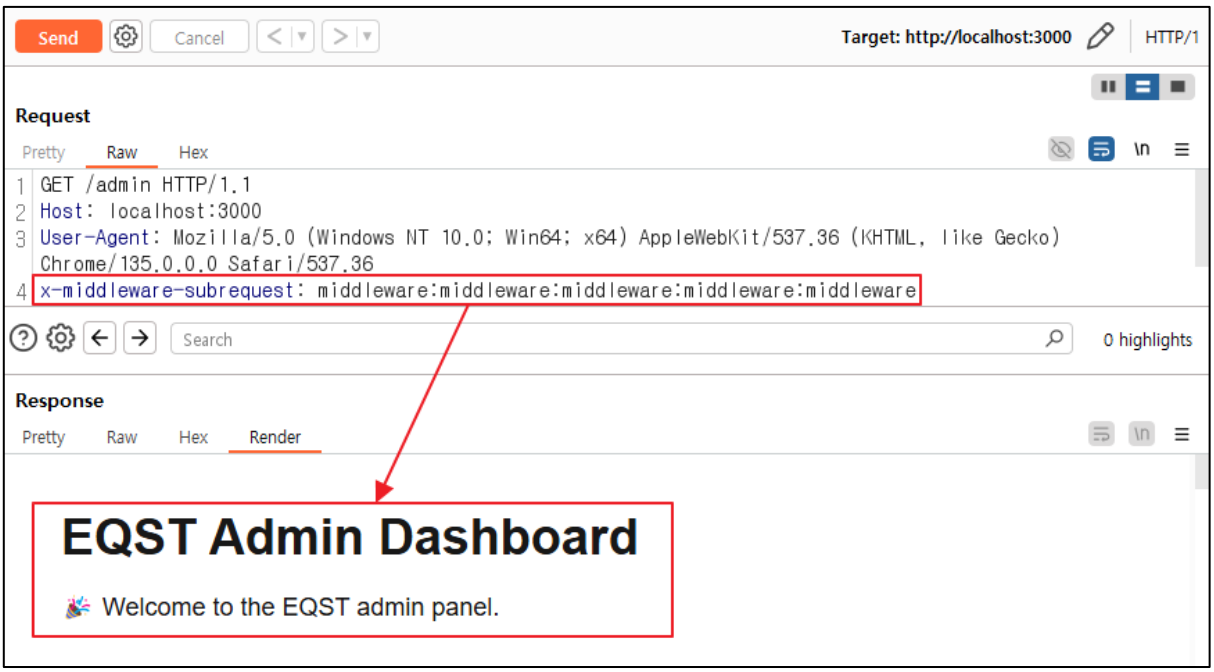


Figure 25. Verification of Bypassing the Middleware Authentication Process

5) Version-specific Middleware Evasion Payloads

As illustrated in Figure 6, the middleware may be situated within the src directory. Consequently, the middleware bypass payload that can be entered via the x-middleware-subrequest header value varies depending on the version as follows.

Version	Bypass Payload Example
version < v12.2	pages/[SubDirectory]/_middleware or ages/[subdirectory]/_middleware
v12.2 ≤ version < v14.2	middleware or src/middleware
version ≥ v14.2	middleware:middleware:middleware:middleware:middleware or src/middleware:src/middleware:src/middleware:src/middleware:src/middleware

■ Response Measures

The vulnerability CVE-2025-29927, which pertains to the bypassing of authentication in Next.js middleware, was identified by researchers zhero; and inzo_. It was initially reported on February 27, 2025.

•URL:https://zhero-web-sec.github.io/research-and-things/nextjs-and-the-corrupt-middleware#section-7

The vulnerability in question was patched on March 18, 2025, and the corresponding security advisory was made public on March 21, 2025.

•URL: https://github.com/vercel/next.js/security/advisories/GHSA-f82v-jwr5-mffw

Upon examining the patch notes for version 15.2.3, one can ascertain the specific security measures that have been implemented.

•URL: https://github.com/vercel/next.js/compare/v15.2.2...v15.2.3

Initially, in packages/next/src/server/lib/router-server.ts, the function crypto. getRandomValues is utilized to generate an 8-byte random value. Subsequently, this value is stored as a global variable[4] for the @next/middleware-subrequest-id.

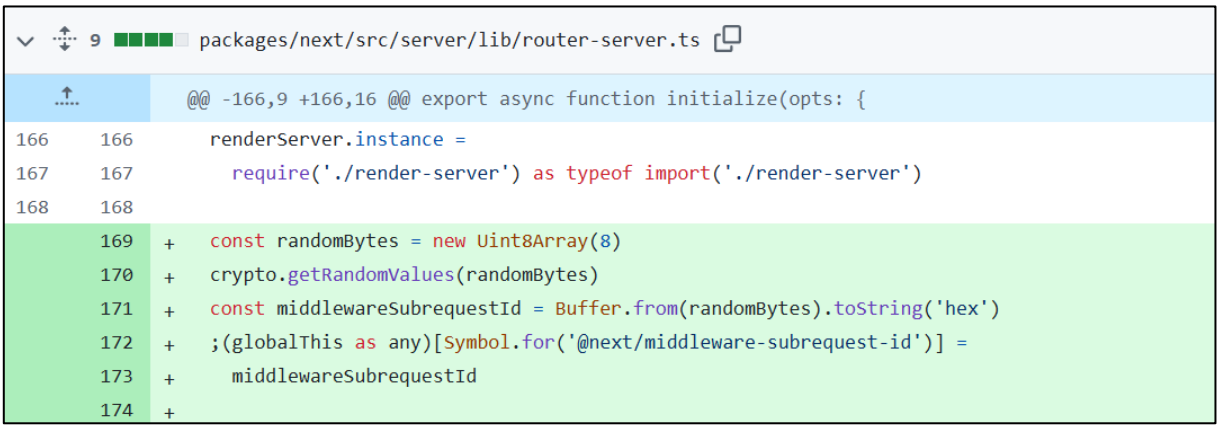


Figure 26. Modifications to packages/next/src/server/lib/router-server.ts

[4] global variable: A variable declared outside of a function that can be accessed from anywhere in the program.

Subsequently, the verification process implemented in packages/next/src/server/lib/server-ipc/utils.ts ascertains the presence of the x-middleware-subrequest and x-middleware-subrequest-id headers. It further scrutinizes whether the value of the x-middleware-subrequest-id header corresponds with the @next/middleware-subrequest-id stored previously in a global variable. Should there be a discrepancy in the values, the x-middleware-subrequest header is expunged to preclude its utilization.

11 packages/next/src/server/lib/server-ipc/utils.ts

```
@@ -57,5 +57,16 @@ export const filterInternalHeaders = (
57  57      if (INTERNAL_HEADERS.includes(header)) {
58  58        delete headers[header]
59  59      }
    60  +
    61  +   // If this request didn't origin from this session we filter
    62  +   // out the "x-middleware-subrequest" header so we don't skip
    63  +   // middleware incorrectly
    64  +   if (
    65  +     header === 'x-middleware-subrequest' &&
    66  +     headers['x-middleware-subrequest-id'] !==
    67  +       (globalThis as any)[Symbol.for('@next/middleware-subrequest-id')]
    68  +   ) {
    69  +     delete headers['x-middleware-subrequest']
    70  +   }
```

Figure 27. Modifications to packages/next/src/server/lib/server-ipc/utils.ts

Users are incapable of predicting the 8-byte random values stored internally; consequently, the server side can securely utilize the x-middleware-subrequest header to ascertain whether execution within the middleware is taking place.

The determination of whether vulnerable versions are in use can be ascertained by examining the package.json file within the source code. Versions below 15.2.3, 14.2.25, 13.5.9, 12.3.5, and all 11.x versions are susceptible to vulnerabilities. Consequently, it is imperative to apply patches to these versions if they are in use, to prevent the exploitation of x-middleware-subrequest by malicious actors.

```
{
  "name": "case4",
  "version": "0.1.0",
  "private": true,
  ▷Debug
  "scripts": {
    "dev": "next dev",
    "build": "next build",
    "start": "next start",
    "lint": "next lint"
  },
  "dependencies": {
    "react": "^19.0.0",
    "react-dom": "^19.0.0",
    "next": "15.2.2"
  },
  "devDependencies": {
    "typescript": "^5",
    "@types/node": "^20",
    "@types/react": "^19",
    "@types/react-dom": "^19",
    "postcss": "^8",
    "tailwindcss": "^3.4.1"
  }
}
```

Figure 28. Example of Using a Vulnerable Next.js Version

It should be noted that version 11.x is no longer supported, rendering patching unfeasible; thus, migration to the continuously supported version 15 is recommended. Should patching to a secure version prove challenging, it is advisable to block external user requests that include the x-middleware-subrequest header from reaching the Next.js application.

■ Reference Sites

• Wikipedia (Next.js) : https://en.wikipedia.org/wiki/Next.js
• Wikipedia (Static web page) : https://en.wikipedia.org/wiki/Static_web_page
• Wikipedia (Server-side rendering) : https://en.wikipedia.org/wiki/Server-side_scripting
• Project structure and organization (Next.js docs) : https://nextjs.org/docs/app/getting-started/project-structure
• middleware (Next.js docs) : https://nextjs.org/docs/app/building-your-application/routing/middleware
• Next.js and the corrupt middleware: the authorizing artifact (zhero_web_security) : https://zhero-web-sec.github.io/research-and-things/nextjs-and-the-corrupt-middleware
• Authorization Bypass in Next.js Middleware (GitHub Security Advisory) : https://github.com/vercel/next.js/security/advisories/GHSA-f82v-jwr5-mffw