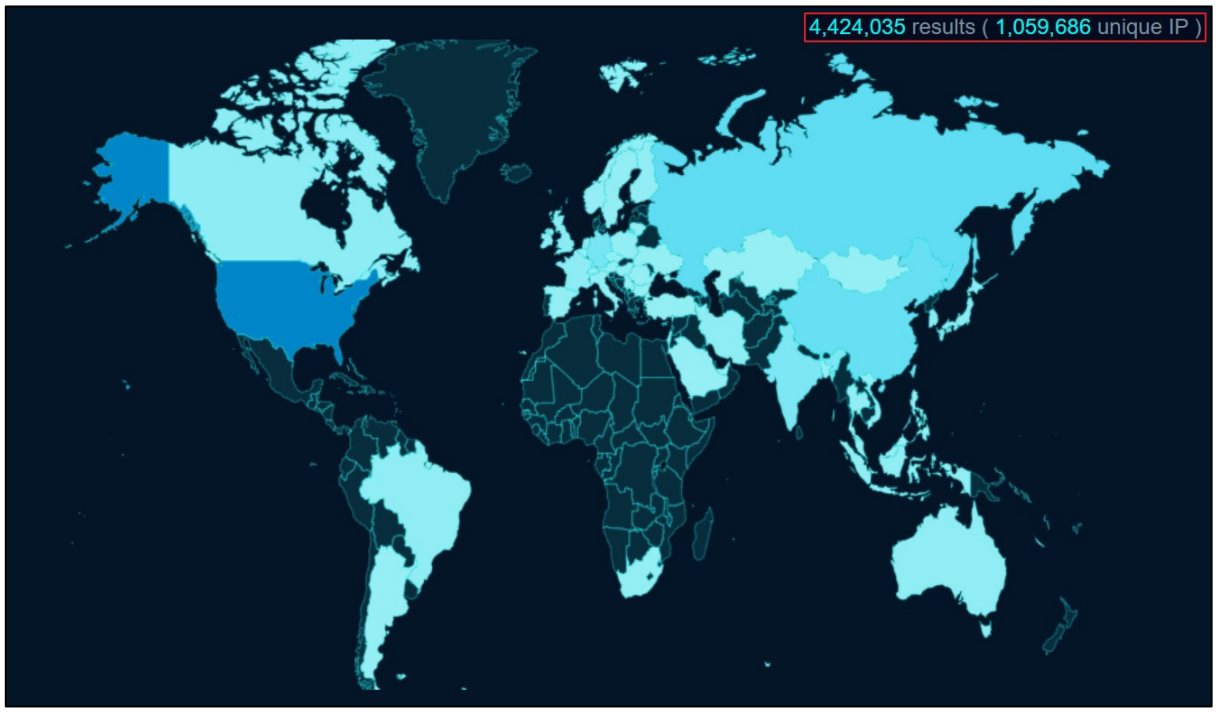
Research & Technique

Next.js middleware 우회 취약점(CVE-2025-29927)

■ 서론

Next.js 는 서버 사이드 렌더링(SSR)[1] 과 정적 웹 페이지 생성(SSG)[2] 을 지원하는 Node.js 기반의 오픈 소스 웹 프레임워크다. 전 세계적으로 널리 사용되는 자바스크립트 라이브러리인 React 는 공식 문서에서 권장하는 툴체인 중 하나로 Next.js 를 소개하고 있다.
OSINT 검색 엔진을 통해 인터넷에 공개된 Next.js 사용 현황을 조사한 결과, 2025 년 4 월 15 일 기준 미국, 러시아, 독일 등 수많은 국가의 총 442 만 개 사이트에서 Next.js 가 사용되고 있는 것으로 확인되었다.



출처: npmjs.com

그림 1. Next.js 사용 통계

[1] 서버 사이드 렌더링 (Server Side Rendering): 서버에서 모든 데이터를 작성하여 클라이언트로 전송, 클라이언트는 해당 데이터를 해석해 웹사이트를 표시하는 웹 통신 방법

[2] 정적 웹 페이지 생성 (Static Site Generation): 페이지 HTML 을 미리 빌드 타임에 생성하여 각 요청마다 재사용하는 방법

2025 년 3 월 21 일, Next.js 의 middleware[3] 우회 취약점(CVE-2025-29927)이 공개됐다. 이 취약점은 특정 헤더 값을 이용해 middleware 가 이미 실행되었는지 여부를 확인하는 Next.js 의 내부 로직을 악용함으로써 발생한다. 공격자는 헤더 값을 조작해 middleware 를 우회할 수 있으며, 인증 절차가 middleware 를 통해 구현되어 있을 경우, 이를 우회해 제한된 페이지에 접근이 가능하다. Next.js 는 광범위하게 사용되는 웹 프레임워크인 만큼, 해당 취약점에 노출되어 있는지 반드시 점검해야 한다.

■ 공격 시나리오

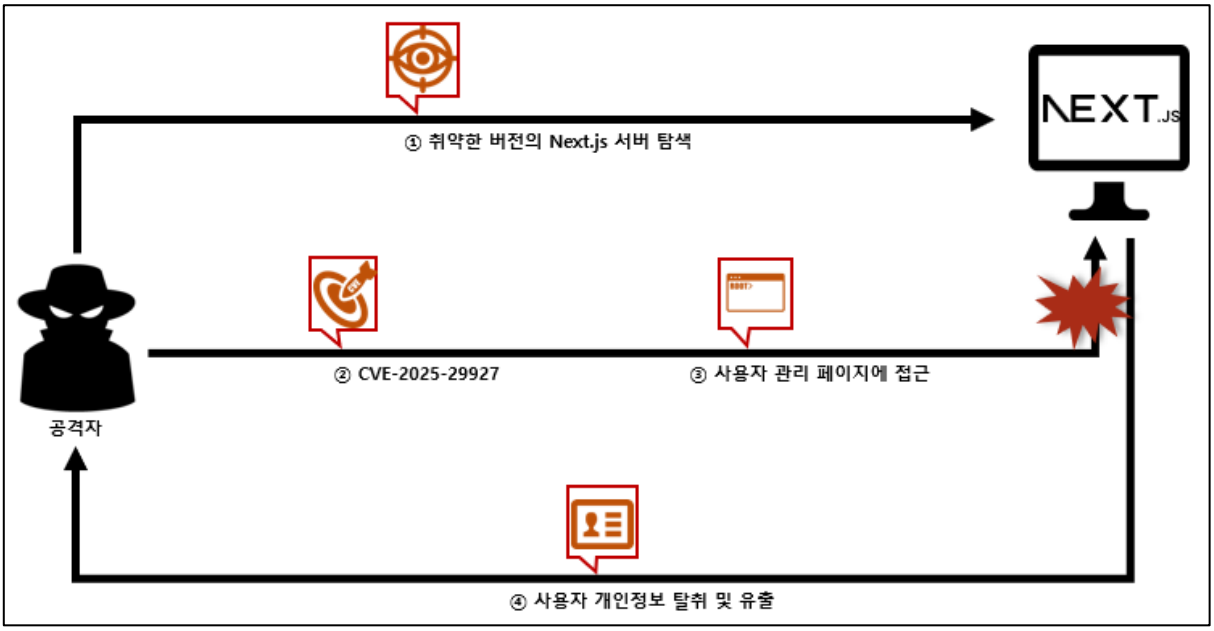


그림 2. CVE-2025-29927 공격 시나리오

① 공격자는 취약한 버전의 Next.js를 사용하는 서버를 탐색
② 이후 CVE-2025-29927 취약점을 이용하여 인증 우회 시도
③ 인증 우회에 성공한 공격자는 피해자의 사용자 관리 페이지에 접근
④ 사용자 관리 페이지에 접근한 공격자는 대량의 개인정보를 열람·탈취하여 외부로 유출

■ 영향받는 소프트웨어 버전

CVE-2025-29927 에 취약한 소프트웨어 버전은 다음과 같다.

S/W 구분	취약 버전
Next.js	v15.2.3 미만
	v14.2.25 미만
	v13.5.9 미만
	v12.3.5 미만
	v11 버전 전체

[3] 미들웨어 (middleware): 요청과 응답 주기에서 들어오는 요청이 도달하기 전에 처리하고 응답이 전송되기 전에 처리하는 개념

■ 테스트 환경 구성 정보

테스트 환경을 구축하여 CVE-2025-29927 의 동작 과정을 살펴본다.

이름	정보
피해자	Next.js v15.1.7 (192.168.0.3)
공격자	Kali Linux (192.168.216.133)

■ 취약점 테스트

Step 1. 환경 구성

피해자 PC 에 Next.js v15.1.7 환경을 설치하고, 취약점 재현을 위한 테스트 환경을 구성한다. 자세한 구성 방법은 아래 GitHub 저장소에서 확인 가능하다.

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

```
# GitHub 저장소 클론
git clone https://github.com/EQSTLab/CVE-2025-29927

# 디렉토리 이동 후 도커 이미지 빌드 및 실행
cd next15
docker build -t nextjs .
docker run -p 3000:3000 --rm -it nextjs
```

웹 브라우저에서 피해자 주소에 접속하여 Next.js 서버가 정상 구동 중인지 확인한다.

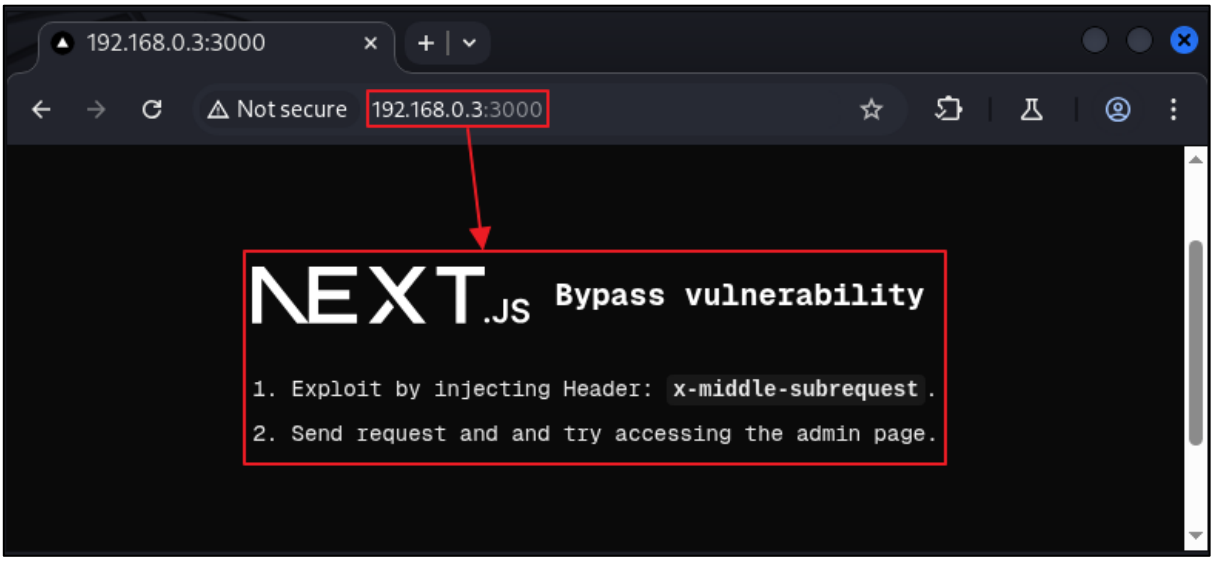


그림 3. 취약한 Next.js 환경 구축 확인

Step 2. 취약점 테스트

Next.js 서버는 /admin 엔드포인트에 대해 인증이 없는 접근을 차단하도록 구성되어 있다.

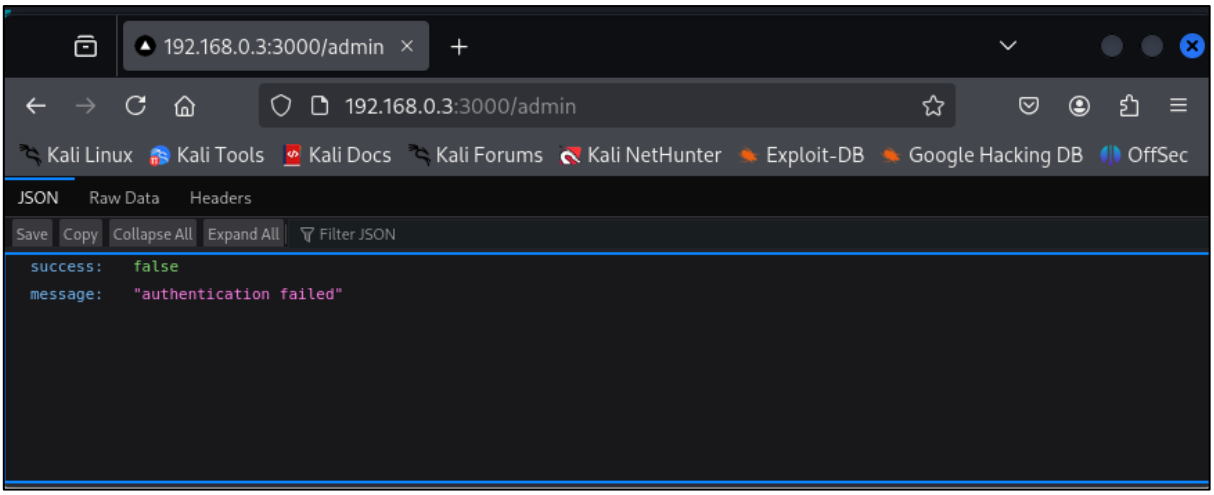


그림 4. /admin 엔드포인트 접근 차단

하지만, 다음과 같은 헤더를 요청에 추가하면 middleware 인증 절차가 우회된다.

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

이 헤더가 포함된 요청을 전송하면, middleware 가 이미 실행된 것으로 처리되어 /admin 엔드포인트에 접근이 가능해진다.

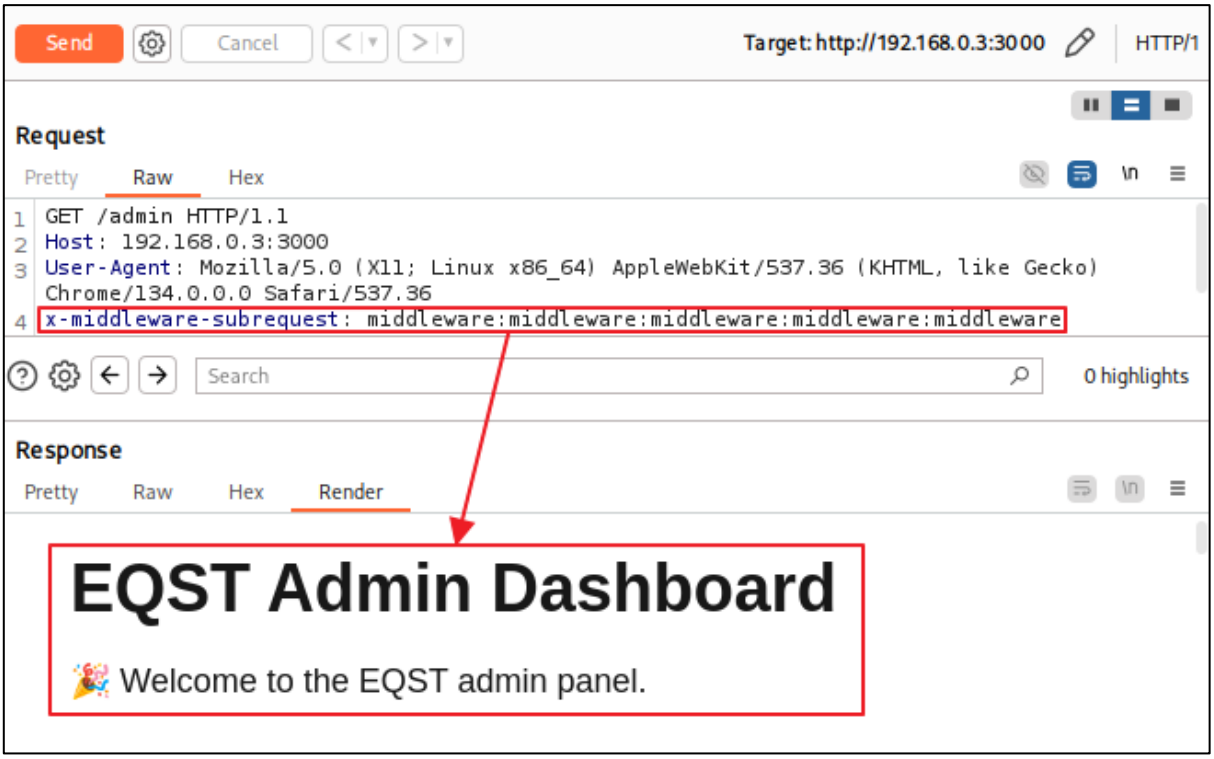


그림 5. middleware 인증 우회 확인

■ 취약점 상세 분석

취약점 상세 분석에서는 CVE-2025-29927 취약점의 발생 원리부터 인증 우회 과정까지 단계적으로 설명한다. Step 1 에서는 Next.js 의 middleware 개념과 버전별 특징을 소개한다. Step 2 에서는 각 버전의 middleware 구현에서 나타난 취약 지점과, 이를 활용한 인증 우회 기법을 설명한다.

Step 1. Next.js middleware

1) Next.js middleware 의 특징

Middleware 는 요청-응답 주기에서 클라이언트의 요청이 애플리케이션에 도달하기 전에 처리되는 중간 계층으로, 응답이 전송되기 전에도 추가적인 처리를 수행할 수 있다.

Next.js 에서는 이 기능을 통해 요청 전 검증, 헤더 재작성, 리디렉션 등을 구현할 수 있으며, middleware.ts 또는 middleware.js 파일을 통해 정의된다. Middleware 파일은 프로젝트 최상위 디렉토리에 app, pages 폴더와 나란히 위치하거나, src 디렉토리 내에 포함될 수 있다.

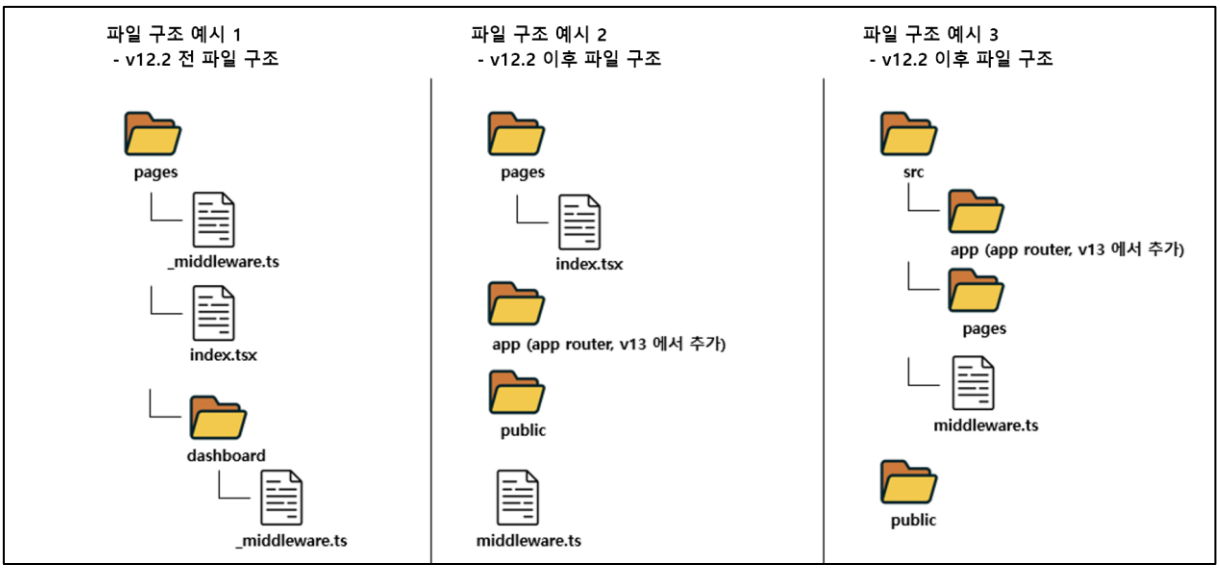


그림 6. 파일 구조별 middleware 위치 예시

2) Next.js 버전별 Middleware 특징

Next.js 는 v12.2 부터 middleware 를 업그레이드하고 패치하여, 이후 middleware 사용 방식에 중요한 변화가 있었다.

(1) v12.2 미만 버전

Next.js v12.2 이전 버전에서는 모든 디렉토리에서 middleware 를 정의할 수 있었다. 이 경우 서버는 요청이 들어오면 상위 디렉토리부터 하위 디렉토리 순으로 middleware 를 차례로 실행한다.

예를 들어, /pages 디렉토리에 middleware 가 정의되어 있으면, 그 index.tsx 와 about.tsx, teams.tsx 전체 요청에 해당 middleware 가 적용된다.

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

그림 7. /pages 디렉토리 내 _middleware 파일

또한 /pages, /pages/about, /pages/about/teams 각 디렉토리에 middleware 가 정의되어 있는 경우, 요청이 들어오면 디렉토리의 계층 구조에 따라 상위부터 하위 순서로 middleware 가 실행된다.

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

그림 8. 중첩된 _middleware 파일 실행 순서

(2) v12.2 이후 버전

Next.js v12.2 부터는 middleware 파일명에서 더 이상 underscore(_)를 사용하지 않으며, 파일명은 middleware.ts 또는 middleware.js 로 변경되었다. 또한, 디렉토리별로 middleware 를 중첩 정의하는 방식이 지원되지 않고, 프로젝트 루트나 pages 디렉토리와 병렬 구조에 단 하나의 middleware 파일만 둘 수 있다.

이처럼 디렉토리 단위의 middleware 적용이 불가능해지면서, 특정 경로에 대해 middleware 를 적용하려면 matcher 설정을 활용해야 한다. 예를 들어, /admin 으로 시작하는 경로에 middleware 를 적용하려면 아래와 같이 matcher 에 해당 패턴을 지정해야 한다.

```
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  }
```

그림 9. matcher 설정 예시 코드

또는 아래와 같이 middleware 내부에 조건문을 작성해, 요청 경로에 따라 동작을 분기할 수도 있다. 이 방식은 matcher 설정만으로는 제어하기 어려운 복잡한 조건이 필요한 경우에 유용하게 활용된다.

```
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  }
```

그림 10. 조건 분기 예시 코드

Step 2. Next.js middleware 인증 구현 및 버전별 우회 방법

1) middleware 를 활용한 인증 구현

Next.js 에서 제공하는 인증 구현 방식 중에는 middleware 를 활용하는 방법도 포함되어 있다. 예를 들어, 아래는 사용자 권한에 따라 특정 경로로 리다이렉션하는 코드의 예시이다. 이러한 방식은 UI 요소를 숨기거나, 권한 및 역할에 따라 접근을 제어하는 등의 빠른 처리가 필요한 상황에서 유용하게 사용된다.

```
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  }
```

그림 11. middleware 를 활용한 Next.js 인증 구현 예시

middleware 를 통해 인증을 구현한 후, 인증 없이 /admin 엔드포인트에 접근을 시도하면, 아래와 같이 요청이 차단되는 것을 확인할 수 있다.

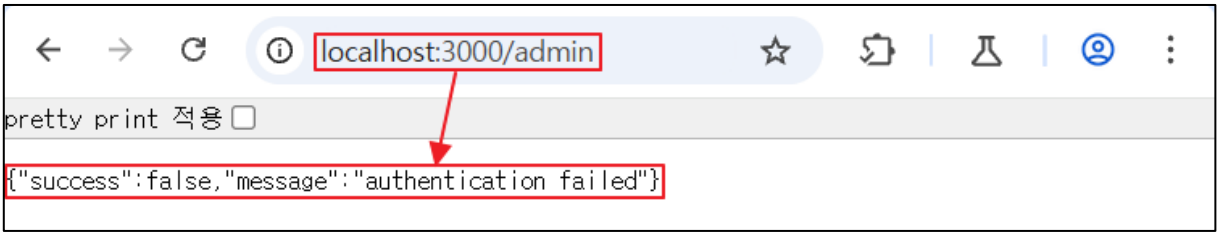


그림 12. x-middleware-request 검증 과정

2) v12.2 미만 버전

Next.js 는 내부적으로 x-middleware-subrequest 헤더를 활용하여 해당 요청에 대해 middleware 가 이미 실행되었는지를 판단한다. 이 로직은 v12.0.1 기준, /packages/next/server/next-server.ts 파일 내에서 확인할 수 있다. 코드의 주요 흐름은 다음과 같다.

```
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     }
```

그림 13. x-middleware-request 검증 과정

① x-middleware-subrequest 헤더를 통해 전달된 값을 colon(:) 기준으로 나눠서 배열로 구성한다.
② 구성된 배열에 middlewareInfo.name 변수의 값이 있는지 확인한다.
③ 만약 해당 값이 존재한다면 NextResponse.next() 함수를 통해 middleware 를 무시하고, 다음 처리 단계로 넘어가 요청을 보낸 경로의 응답을 받는다.

이때, middlewareInfo.name 변수의 값은 /packages/next/server/next-server.ts 파일 내 getMiddlewareInfo 함수를 통해 로드된다. 이 함수는 요청 경로에 대응하는 middleware 정보를 가져오는 역할을 하며, 해당 정보를 바탕으로 x-middleware-subrequest 헤더를 검사하여 mid-dleware 실행 여부를 판단하게 된다.

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

그림 14. getMiddlewareInfo 로 불러오는 middlewareInfo

getMiddlewareInfo 함수는 /packages/next/server/require.ts 파일에 구현되어 있으며, 지정된 경로에 해당하는 middleware 정보를 객체 형태로 반환한다.

```
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    ))
...
```

그림 15. getMiddlewareInfo 내 MIDDLEWARE_MANIFEST 호출

이 로직은 Next.js 가 빌드된 후 생성되는 .next 디렉토리 내의 .next/server/middleware-manifest.json 파일을 참조하여 middleware 정보를 불러온다.

```
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  }
```

그림 16. middleware-manifest.json 내 middleware name 정보

middleware-manifest.json 파일의 name 키는 각 middleware 의 위치 경로를 나타낸다. 이에 따라 middlewareInfo.name 값은 최종적으로 pages/_middleware가 된다. 이는 디버깅 도구를 통해 실행 중인 값을 확인했을 때에도 동일하게 확인된다.

```
RUN AND DEBUG
VARIABLES
Block: runMiddleware
middlewareInfo = {name: 'pages/_middleware', paths: Array(2), env: Array(0), wasm: Array(0)}
  env = (0) []
  name = 'pages/_middleware'
```

그림 17. 디버거를 통해 확인한 middlewareInfo.name 의 값

즉, x-middleware-subrequest 헤더 값이 pages/_middleware 로 설정되면, 해당 요청은 이미 middleware 를 통과한 것으로 처리된다. 이로 인해 middleware 가 실행되지 않으며, 내부에 구현된 인증 절차 역시 우회할 수 있어 취약점이 발생하게 된다.

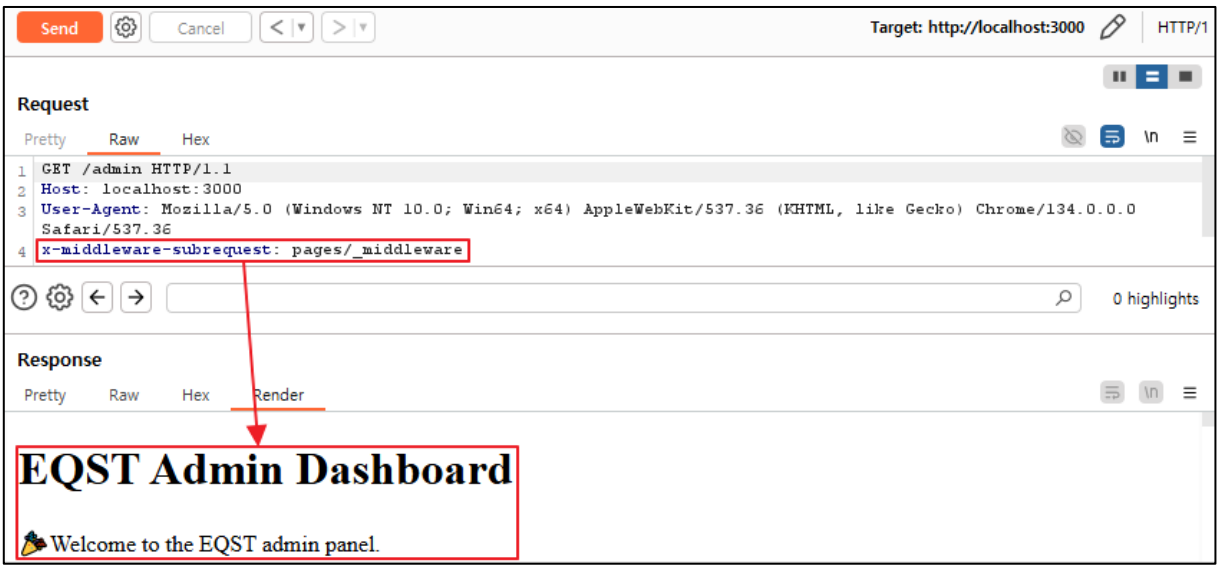


그림 18. middleware 인증 과정 우회 확인

앞서 Step 1 에서 설명했듯, v12.2 이전 버전에서는 디렉토리별로 middleware 를 개별 정의할 수 있다. 이 경우 pages 경로 아래에 정의된 각 middleware 경로를 x-middleware-subrequest 헤더 값에 입력하면 인증 절차를 우회할 수 있다. 그림 8 예시의 경우 pages/about/_middleware 또는 pages/about/teams/_middleware 를 헤더에 지정하면, 해당 middleware 를 우회하는 것이 가능하다.

3) v12.2 이상 v14.2 미만 버전

x-middleware-subrequest 에 대한 로직은 변경되지 않았기 때문에, 이전과 마찬가지로 해당 헤더에 middleware 경로를 설정하면 인증을 우회할 수 있다. v12.2 부터는 middleware 파일명이 middleware.ts 또는 middleware.js 로 변경되었고, 위치도 pages 디렉토리 내부가 아닌 그와 나란한 상위 경로로 통일되었다. 이에 따라 middleware 경로는 기존의 pages/_middleware 가 아닌, 단순히 middleware 로 바뀌었다.

이와 같은 변경 사항은 .next/server/middleware-manifest.json 파일에서 확인할 수 있으며, 이 파일을 통해 middleware 의 경로가 pages/_middleware 에서 middleware 로 변경된 것을 알 수 있다.

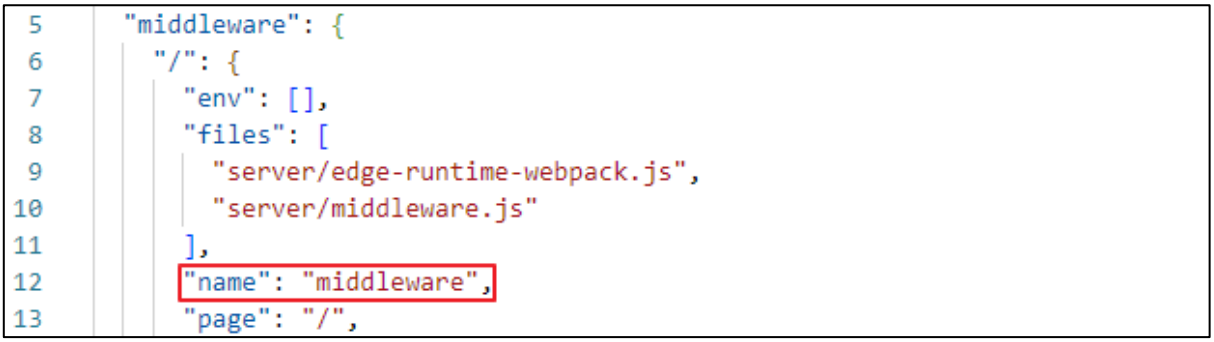


그림 19. middleware-manifest.json 내 middleware name 정보

위의 Next.js v12.2 사례와 마찬가지로, 디버거를 통해 실행 중인 값을 확인하면 해당 값이 middleware 임을 확인할 수 있다.

그림 20. 디버거를 통해 확인한 middlewareInfo.name 의 값

마찬가지로, x-middleware-subrequest 헤더의 값을 middleware 로 설정하면 인증 절차를 우회할 수 있다.

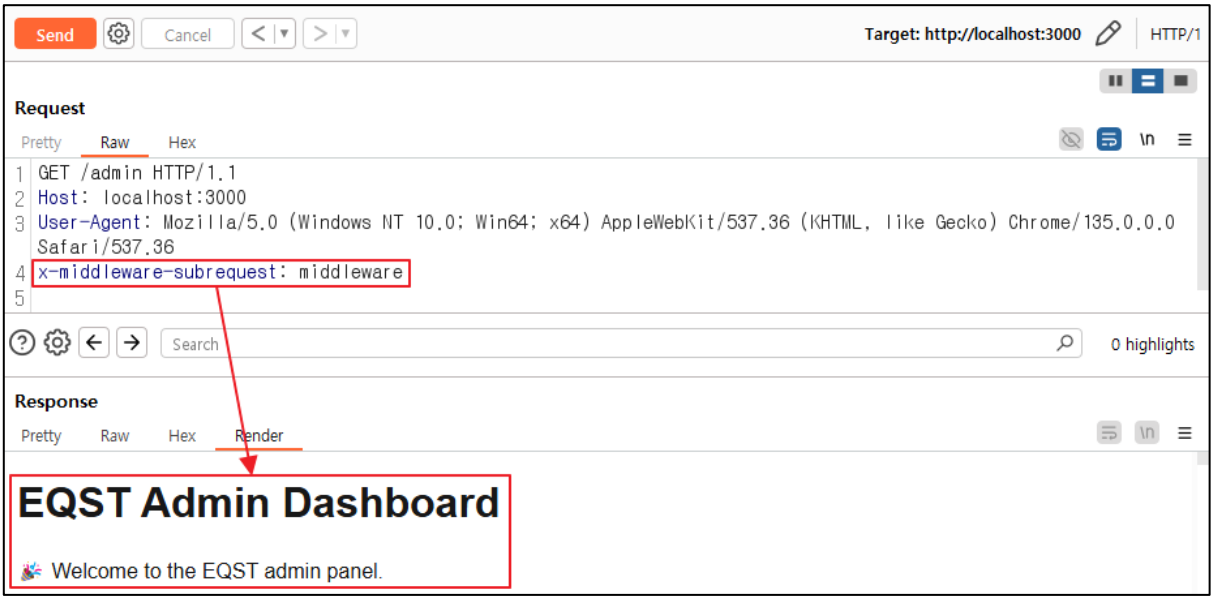


그림 21. middleware 인증 과정 우회 확인

4) v14.2 이후 버전

Next.js v14.2 부터는 middleware 의 x-middleware-subrequest 헤더의 값에 포함된 middleware 경로가 colon(:)을 기준으로 일정 횟수 이상 반복되면, 해당 요청이 middleware 를 우회하도록 코드가 변경되었다.

이는 /packages/next/src/server/web/sandbox/sandbox.ts 코드를 통해 확인할 수 있다.

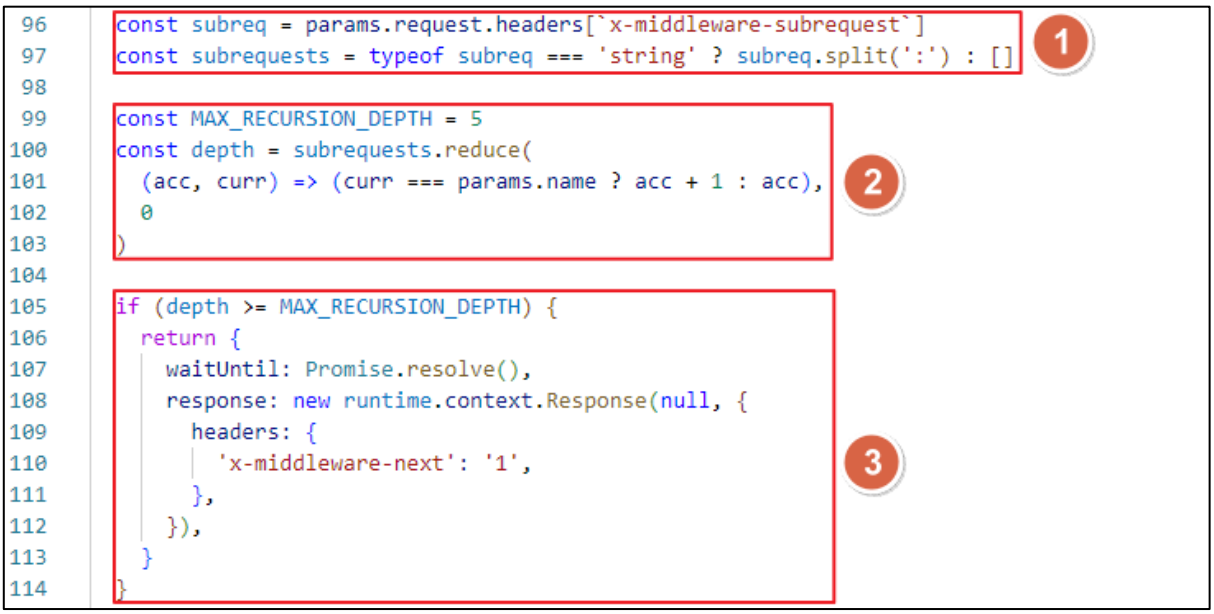


그림 22. 변경된 x-middleware-request 검증 과정

① x-middleware-subrequest 헤더를 통해 전달된 값을 colon(:) 기준으로 나눠서 배열로 구성한다.
② 구성된 배열에 params.name 변수의 값이 몇 개가 있는지 확인한다.
③ 만약, params.name 변수의 값이 5번 이상 반복이 된다면, middleware를 건너뛰고 요청을 보낸 경로의 응답을 받는다.

여기서 params.name 은 앞서 언급한 middleware.name 과 동일한 값으로, 이는 .next/server/middleware-manifest.json 파일과 디버깅을 통해 확인할 수 있다.

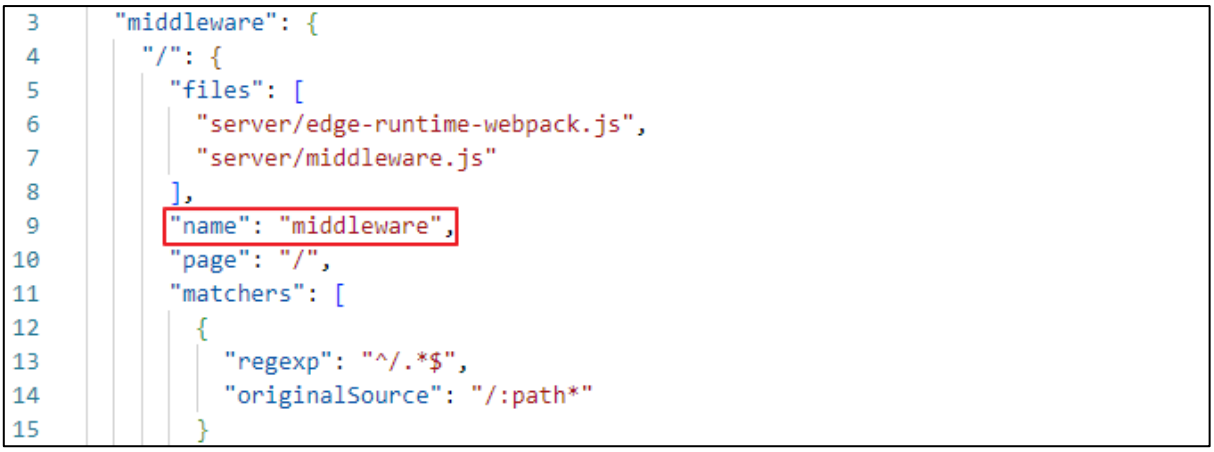


그림 23. middleware-manifest.json 내 middleware name 정보

그림 24. 디버거를 통해 확인한 params.name 의 값

따라서, v14.2 이후에는 colon(:)을 기준으로 middleware 경로가 5 번 이상 반복되어야 하므로, middleware:middleware:middleware:middleware:middleware 와 같은 값을 x-middleware-subrequest 헤더에 입력하면 middleware 검증을 우회할 수 있다.

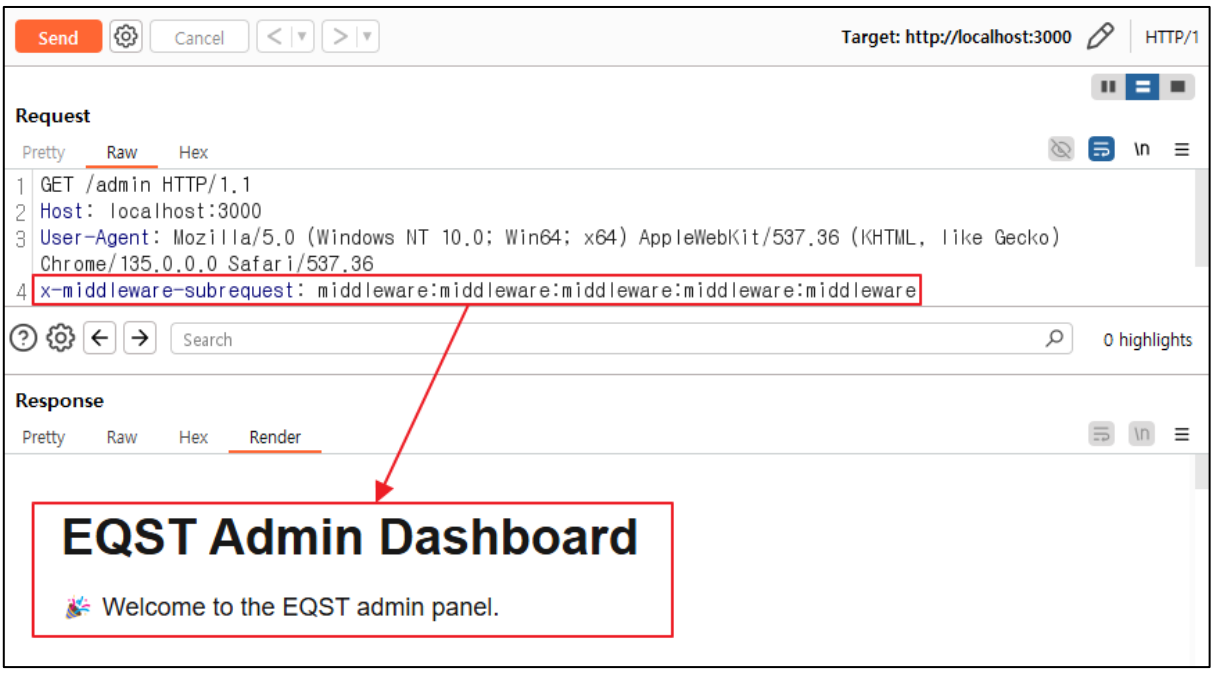


그림 25. middleware 인증 과정 우회 확인

5) 버전별 middleware 우회 payload

그림 6 과 같이, middleware 는 src 디렉토리 내에 위치할 수도 있다. 따라서, x-middleware-subrequest 헤더 값으로 입력할 수 있는 middleware 우회 payload 는 버전별로 아래와 같다.

버전	우회 payload 예시
v12.2 미만	pages/[SubDirectory]/_middleware 또는 ages/[subdirectory]/_middleware
v12.2 이상 v14.2 미만	middleware 또는 src/middleware
v14.2 이상	middleware:middleware:middleware:middleware:middleware 또는 src/middleware:src/middleware:src/middleware:src/middleware:src/middleware

■ 대응 방안

Next.js middleware 인증 우회 취약점인 CVE-2025-29927 은 취약점 연구원인 zhero;와 inzo_에 의해 발견되었으며, 2025 년 2 월 27 일에 처음으로 신고되었다.

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

해당 취약점은 2025 년 3 월 18 일에 패치되었으며, 이에 대한 보안 권고는 3 월 21 일에 공개되었다.

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

v15.2.3 에서 패치된 내역을 살펴보면, 어떤 보안 조치가 취해졌는지 확인할 수 있다.

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

우선, packages/next/src/server/lib/router-server.ts 에서 crypto.getRandomValues 함수를 사용하여 8 바이트의 랜덤 값을 생성한다. 이후 이 값을 @next/middleware-subrequest-id 에 대한 값으로 전역 변수[4] 에 저장한다.

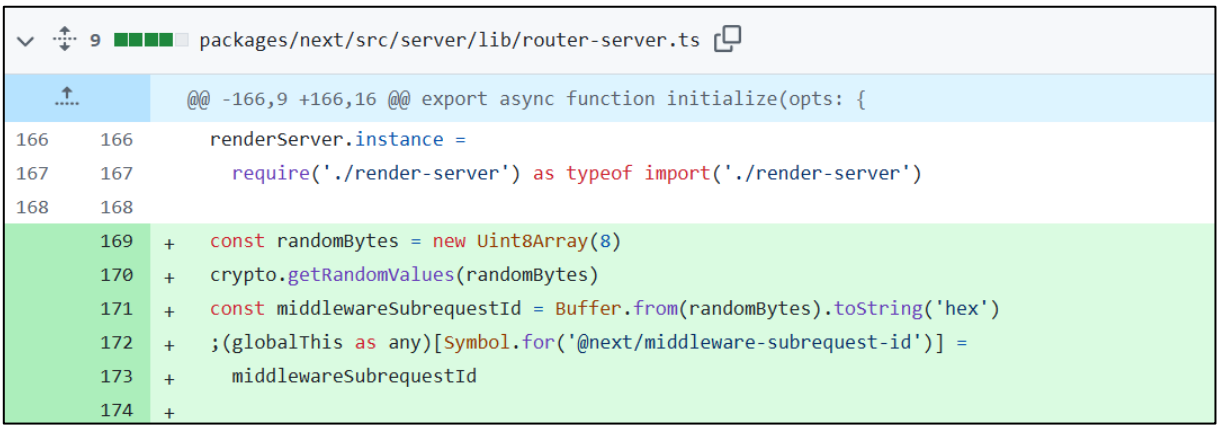


그림 26. packages/next/src/server/lib/router-server.ts 수정 사항

[4] 전역 변수(global variable): 함수의 외부에서 선언된 변수로, 프로그램 어디에서나 접근할 수 있는 변수

이후, packages/next/src/server/lib/server-ipc/utils.ts 에서 구현된 검증 과정은 x-middleware-subrequest 헤더와 x-middleware-subrequest-id 헤더 값이 존재하는지 확인하고, x-middleware-subrequest-id 헤더 값이 앞서 전역 변수에 저장된 @next/middleware-subrequest-id 와 일치하는지 검사한다. 만약 값이 일치하지 않으면, x-middleware-subrequest 헤더를 삭제하여 이를 활용할 수 없도록 막는다.

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  +     }
```

그림 27. packages/next/src/server/lib/server-ipc/utils.ts 수정 사항

사용자는 내부적으로 저장되는 8 바이트 난수 값을 예측할 수 없으므로, 서버 측에서는 x-middleware-subrequest 헤더를 안전하게 활용하여 middleware 내부 실행 여부를 판별할 수 있다.

취약한 버전 사용 여부는 소스코드 내 package.json 파일을 통해 확인할 수 있다. 15.2.3 미만, 14.2.25 미만, 13.5.9 미만, 12.3.5 미만, 11.x 버전이 취약한 버전이므로, 해당 버전을 사용 중이라면 패치를 통해 x-middleware-subrequest 가 공격자에게 악용되지 않도록 차단하는 것이 중요하다.

```
1  {
2    "name": "case4",
3    "version": "0.1.0",
4    "private": true,
     ▷Debug
5    "scripts": {
6      "dev": "next dev",
7      "build": "next build",
8      "start": "next start",
9      "lint": "next lint"
10   },
11   "dependencies": {
12     "react": "^19.0.0",
13     "react-dom": "^19.0.0",
14     "next": "15.2.2"
15   },
16   "devDependencies": {
17     "typescript": "^5",
18     "@types/node": "^20",
19     "@types/react": "^19",
20     "@types/react-dom": "^19",
21     "postcss": "^8",
22     "tailwindcss": "^3.4.1"
23   }
24 }
```

그림 28. 취약한 next.js 버전 사용 예시

다만 11.x 버전은 이미 지원이 종료되어 더 이상 패치가 불가능하므로, 공식 지원이 지속되고 있는 15 버전으로의 마이그레이션을 권장한다. 만약 안전한 버전으로의 패치가 어려운 상황이라면, x-middleware-subrequest 헤더를 포함한 외부 사용자의 요청이 Next.js 애플리케이션에 도달하지 않도록 차단하는 것이 바람직하다.

■ 참고 사이트

• 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