Next.js 미들웨어 기반 인증 우회 취약점 분석

(CVE-2025-29927)

최신 취약점에 대한 기술 문서 리뷰

Name : 조민혁

Class Number : 5

Phone Number : 010-4923-2198

목 차

1. Next.js 미들웨어 기반 인증 우회 취약점

1-1. 내용 요약

Next.js는 Vercel이 개발한 React 기반 웹 프레임워크로 서버 사이드 렌더링, 정적 사이트 생성, API 라우팅 등 다양한 기능을 제공하여 널리 사용되고 있다. 이 중 미들웨어 기능은 요청 처리 전에 인증, 로깅, 리다이렉션 등의 작업을 처리하는데 핵심적인 역할을 하고 있다.

2025년 3월, Next.js에서 미들웨어 기반 인증 우회 취약점 (CVE-2025-29927)이 공개되었다. 이 취약점은 Next.js의 미들웨어가 HTTP 요청 헤더 중 하나인 'x-middleware-subrequest'를 신뢰하고 검증 없이 처리하는 방식에서 비롯되며 공격자가 이 헤더를 악의적으로 조작함으로써 인증 로직을 우회할 수 있다.

Next.js의 미들웨어는 '클라이언트 요청' -> '미들웨어 검증 및 처리' -> '실제 페이지 라우팅'의 프로세스를 따른다. 이 과정에서 내부 서브 요청이 무한히 중첩되어 재귀 요청이 발생되지 않도록 하기 위해 Next.js는 x-middleware-subrequest 헤더를 사용한다. 이 헤더가 존재하면 미들웨어는 해당 요청을 내부 요청으로 간주하고 추가적인 인증 검사나 리다이렉션을 수행하지 않는다.

그러나, 이 헤더를 외부 사용자가 직접 조작할 수 있다는 점이다. Next.js는 기본적으로 이 헤더가

내부 요청에서만 사용된다고 가정하지만 실제로는 외부에서 요청 시 자유롭게 삽입할 수 있는 일반적인 HTTP 헤더로써 존재한다는 점이 취약점이 발생하게 된 근본적인 원인이다. 이로 인해 공격자는 인증이 요구되는 경로에 대해 x-middleware-subrequest 헤더를 포함한 요청을 전송함으로써 인증 미들웨어를 통과하고 최종 리소스에 비인가 접근이 가능해진다. 이는 권한 상승, 민감 정보 노출, 계정 탈취 등의 보안 사고로 이어질 수 있는 치명적인 취약점이다.

1-2. 취약점의 근본 원인

Next.js 미들웨어 기반 인증 우회 취약점(CVE-2025-29927)은 x-middleware-subrequest 라는 HTTP 헤더를 신뢰하고 검증 없이 내부 요청 식별에 사용하는 설계상의 결함에서 비롯된다. 이 헤더는 미들웨어 내에서 요청이 내부적으로 파생된 것인지 외부 사용자의 직접 요청인지 판단하는 기준으로 사용되며 내부 요청으로 간주될 경우 인증 검사와 리다이렉션 처리 등을 생략하도록 설계되어 있다. 예를 들어, GET /protected 요청에 x-middleware-subrequest: 1 헤더가 포함되어 있으면 해당 요청은 내부 호출로 처리되어 인증 로직을 통과하게 된다. 이러한 처리 방식은 인증 검사의 생략 여부가 오직 헤더의 존재 유무에만 의존한다는 점에서 보안상 매우 취약한 구조이며 공격자는 이 헤더를 클라이언트에서 직접 조작함으로써 인증이 필요한 페이지에도 접근할 수 있게 된다. 실제로는 HTTP 요청 헤더는 curl, Postman, 브라우저 개발자 도구 등을 통해 외부에서 손쉽게 추가하거나 수정할 수 있는 입력 요소임에도 불구하고 Next.js 는 이 헤더를 안전한 내부 전용 플래그로 오인하고 해당 입력값을 신뢰하는 설계를 유지해 왔다. 인증 로직 또한 보안적으로 취약한 FLAG 기반 설계에 의존하고 있다.

예를 들어

```
if (!req.headers['x-middleware-subrequest'])
  {  /* 인증 로직 */
  }
```

와 같은 코드 구조는 단순히 헤더가 존재하지 않을 때만 인증을 수행하도록 되어 있어 공격자가 해당 헤더만 포함시킴으로써 인증 과정을 완전히 우회할 수 있도록 만든다. 이는 인증 절차가 오직 단일 외부 입력값에 의존하게 되는 구조적 문제이며 본질적으로 보안 경계를 잘못 설정한 데서 발생한 것이다. 특히 내부 개발자는 이 헤더가 코드 내부에서만 설정되고 사용될 것이라는 전제를 두고 인증 로직을 구성했으나 실제 운영 환경에서는 클라이언트가 HTTP 요청 헤더를 자유롭게 조작할 수 있는 개방형 구조임을 고려하지 못했다. 이러한 설계상의 가정과 현실 간의 불일치는 시맨틱 갭(Semantic Gap) 문제로 귀결되며 이는 신뢰 경계의 혼동, 공격자 모델의 과소평가, 입력 검증 부재 등 다수의 보안 원칙 위반으로 이어진다. 아울러 Next.js 는 기본 설정에서도 해당 헤더에 대한 별도의 차단이나 검증 없이 인증 로직을 구성할 수 있도록 되어 있어 보안의 기본 원칙 중 하나인 "Secure by Default" 원칙 역시 위반하고 있다. 종합적으로 볼 때 이 취약점은 외부 입력값을 신뢰하고 로직 제어의 핵심 기준으로 사용하는 잘못된 설계, FLAG 기반 인증 우회 구조,

신뢰 경계 설정 실패, 시맨틱 갭으로 인한 보안 맥락 왜곡, 기본값의 위험성 등 여러 보안 설계 실패가 복합적으로 작용한 결과라 할 수 있다.

1-3. 공격 방법

Next.js의 미들웨어 인증 우회 취약점(CVE-2025-29927)은 x-middleware-subrequest라는 HTTP 헤더가 신뢰 기반으로 처리되는 설계상의 허점을 악용하여 인증 절차를 우회하는 방식으로 발생한다. 이 헤더는 내부적으로 재귀 호출 제한을 위해 사용되며, 클라이언트가 해당 값을 조작하여 미들웨어의 보안 로직을 건너뛰도록 만들 수 있다.

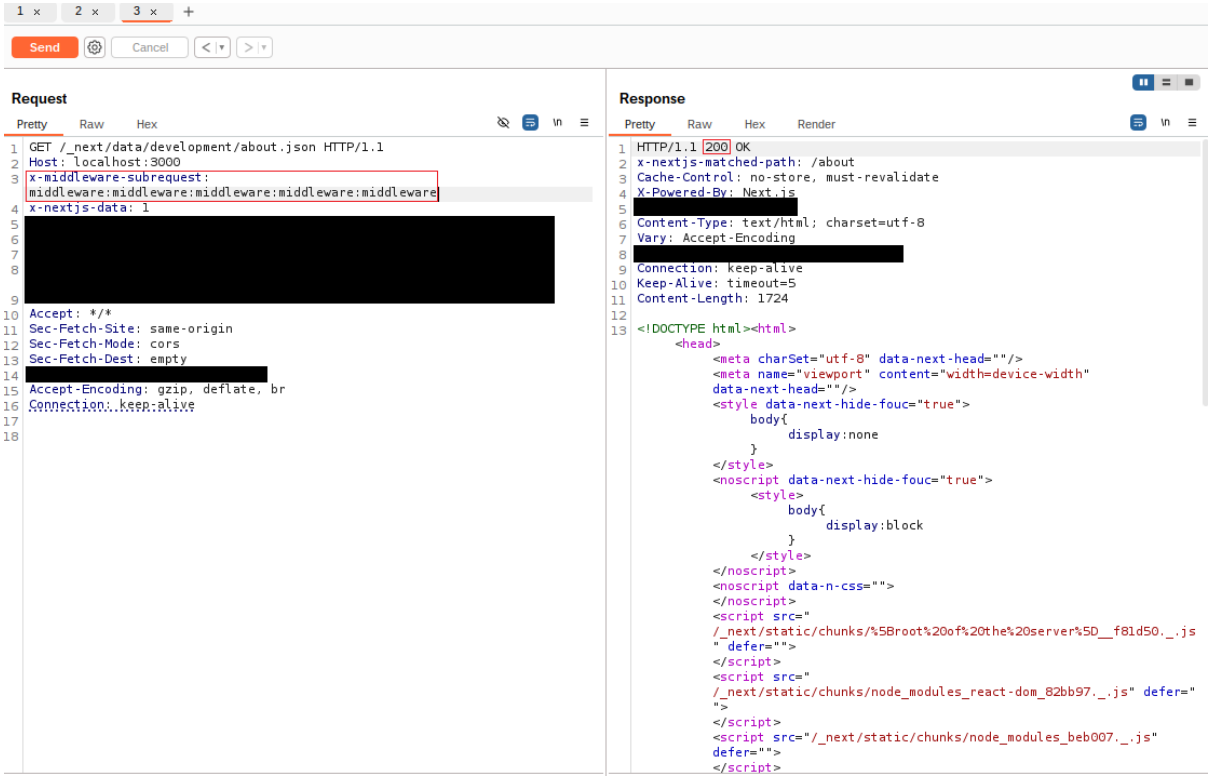
```
export const run = withTaggedErrors(async function runWithTaggedErrors(params) {
  const runtime = await getRuntimeContext(params)
  const subreq = params.request.headers[`x-middleware-subrequest`]
  const subrequests = typeof subreq === 'string' ? subreq.split(':') : []

  const MAX_RECURSION_DEPTH = 5 // [1]
  const depth = subrequests.reduce(
    (acc, curr) => (curr === params.name ? acc + 1 : acc),
    0
  )

  if (depth >= MAX_RECURSION_DEPTH) { //[2]
    return {
      waitUntil: Promise.resolve(),
      response: new runtime.context.Response(null, {
        headers: {
          'x-middleware-next': '1',
        },
      }),
    }
```

Next.js 의 취약한 코드는 위와 같다. 해당 코드는 헤더 x-middleware-subrequest 를 : 기호를 기준으로 분리되어 배열로 처리된다. MAX_RECURSION_DEPTH = 5 는 재귀적으로 미들웨어가 실행되는 최대 깊이를 제한한 것이다. 즉 x-middleware-subrequest 헤더의 각 항목 중 현재 실행 중인 미들웨어의 경로와 일치하는 값이 있을 때마다 depth 값이 1 씩 증가한다. 만약 depth >= MAX_RECURSION_DEPTH 조건이 만족되면 해당 미들웨어는 실행되지 않고 요청이 바로 다음 단계로 전달된다.

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

만약 미들웨어 경로가 middleware 라면 위와 같이 헤더를 삽입하여 우회 가능하다.

또한 일반적인 요청 흐름에서는 인증이 실패하거나 토큰이 존재하지 않을 경우, 서버는 /redirect 또는 /login 페이지로 HTTP 302 리다이렉션을 수행한다. 그러나 공격자가 x-middleware-subrequest 헤더를 포함한 요청을 보낼 경우 미들웨어가 이를 내부 요청으로 오인하게 되어 인증 절차가 생략되며 실제로는 인증되지 않은 요청임에도 HTTP 응답 코드 200(정상)이 반환된다. 공격자가 보호된 경로에 접근했음에도 리다이렉션 없이 정적 페이지나 API 응답을 바로 수신할 수 있게 되는 것이다.



결론적으로 이 취약점은 인증 미들웨어가 단순히 헤더 조작만으로 무력화될 수 있으며 공격자는 인증되지 않은 상태로도 민감한 리소스에 접근하거나 세션을 가장하여 시스템의 무결성을 위협할 수 있다. 이는 보안 로직이 클라이언트 제어 가능한 입력 값에 과도하게 의존할 때 발생할 수 있는 대표적인 구조적 취약 사례다.

1-4. 공격의 파급도

Next.js 미들웨어 기반 인증 우회 취약점은 Security 9.1/10 의 Rating 으로 Critical 하여 매우 치명적인 취약점이다.

즉, 해당 취약점은 단순한 코드상의 결함을 넘어 전체 인증 체계를 무력화하고 서비스의 핵심 보호 영역에 직접 접근 가능하게 만드는 치명적인 보안 위협이다. 이 취약점은 인증을 우회하여 웹 애플리케이션의 신뢰 모델 전반을 붕괴시키는 결과로 이어질 수 있으며 특히 다음과 같은 다양한 측면에서 심각한 파급 효과를 초래한다.

우선, 해당 취약점은 인증 및 인가 절차를 우회할 수 있도록 하므로 공격자가 보호된 리소스에 세션이나 토큰 없이 직접 접근할 수 있게 된다. 예를 들어 /admin, /dashboard, /user/settings, /internal/api 등의 경로는 일반적으로 로그인 또는 권한 확인을 거쳐야 접근 가능한 경로이나, x-middleware-subrequest 헤더 조작만으로 이를 완전히 우회할 수 있다. 이는 관리자 인터페이스, 민감한 사용자 설정 페이지, 내부 API 문서 페이지 등으로의 비인가 접근을 의미하며 계정 탈취, 데이터 노출, 내부 기능 악용 등으로 직결될 수 있다.

둘째, 해당 취약점은 인증 뿐만 아니라 미들웨어에서 수행되는 모든 보안 조치를 우회하게 만든다는 점에서 그 영향력이 확장된다. Next.js 의 미들웨어는 인증 뿐만 아니라 CORS 설정, 보안 헤더 삽입(Content-Security-Policy, X-Frame-Options 등), 리다이렉션 처리, 접근 권한 분기 처리, 사용자의 권한 수준에 따른 흐름 제어도 수행한다. 그러나 x-middleware-subrequest 헤더가 존재하는 경우 이러한 모든 로직이 생략될 수 있으므로, 공격자는 클라이언트의 권한 검증 없이도 보다 넓은 범위의 시스템 통제를 회피할 수 있게 된다.

셋째, 인증 우회를 통해 공격자가 사용자 세션을 가장하거나 내부 API 에 무단 접근하게 되면 이는 시스템 전체의 신뢰성(Trust Model) 훼손으로 이어진다. 공격자는 API 요청에 대해 인증된 사용자인 척 행동할 수 있고 이로 인해 시스템은 잘못된 보안 상태를 기반으로 잘못된 처리를 수행하게 된다. 이는 데이터베이스 수정, 민감 정보 접근, 제 3 자 행위 위장 등으로 이어질 수 있으며 사고 발생 시 책임 추적과 포렌식이 어려워지는 문제도 동반된다.

넷째, 해당 취약점은 공격 난이도가 매우 낮고 재현이 쉬우며 자동화가 가능하다는 점에서 실질적인 위험성이 더욱 크다. 단순히 하나의 HTTP 헤더만 조작하면 되며 인증 우회를 위한 별도의 세션 탈취, XSS, CSRF, SQLi 와 같은 복잡한 익스플로잇 과정 없이 단일 요청만으로 취약점을 유효화할 수 있다. 이는 공격자에게 매우 매력적인 표적이 되며 실제로 공개된 PoC 코드나 블로그, 도구를 기반으로 자동화된 스캔과 익스플로잇이 빠르게 이루어질 수 있다.

다섯째, 영향 범위가 넓다. Next.js 는 널리 사용되는 프레임워크이며, 특히 정적 사이트 생성과 API 라우팅 기능을 통합적으로 사용하는 Enterprise 레벨의 SaaS, B2B, 커머스 플랫폼 등에서 빈번히 사용된다. 이 취약점은 Vercel 과 같은 Managed Hosting 환경에서는 발생하지 않지만, 자체 호스팅(Self-hosted) 환경에서는 방어 수단이 취약한 경우가 많아 대규모 서비스에 치명적이다.

마지막으로, 보안 관점에서 이 취약점은 보안 정책 이행의 실패(SPoF: Single Point of Failure)를 야기하며, 서비스 운영자가 보안 통제를 어디서, 어떤 방식으로 수행해야 하는지에 대한 전략적

혼란을 초래할 수 있다. 인증 검증이 프론트엔드, 미들웨어, API 서버 등 다양한 레이어에 중복적으로 배치되지 않으면 하나의 우회 지점만으로 전체 서비스가 무력화될 수 있기 때문이다.

종합하자면 CVE-2025-29927 은 인증 미비 그 자체를 넘어 웹 서비스의 보안 신뢰 체계와 다중 계층 접근 제어를 동시에 붕괴시킬 수 있는 구조적 위협이며 기술적 파급력과 실무적 피해 가능성 모두에서 매우 위험한 취약점으로 평가된다. 따라서 모든 Self-hosted Next.js 애플리케이션은 즉시 보안 패치를 적용하고, 미들웨어 설계 전반을 다시 검토할 필요가 있다.

1-5. 방어 방법

Next.js 미들웨어 인증 우회 취약점은 x-middleware-subrequest 라는 HTTP 헤더에 대한 신뢰 기반 로직으로 인해 발생한 구조적 결함이다. 이에 따라 방어 방법은 단순한 코드 수정 뿐만 아니라 프레임워크 업데이터, 헤더 검증 및 필터링, 인증 아키텍처 보강, 다중 계층 검증 전략 등 다양한 측면에서 종합적으로 접근할 필요가 있다.

가장 먼저 조치해야할 것은 Next.js 프레임워크를 보안 패치가 반영된 최신 버전으로 업데이트 하는 것이다. 업데이트 버전에서는 trustedHeader 검사 로직이 삽입되어 외부 요청에서 삽입된 헤더는 자동 무시되는 특징이 있기에 반드시 업데이트를 통해 방어를 해야한다.

만약 보안 패치 적용이 어렵거나 시간이 필요한 경우 임시 대응 방안으로 x-middleware-subrequest 헤더를 제거하거나 차단하는 필터링 규칙을 적용하여 공격자의 헤더 조작을 무력화하는 것이다.

개발자가 미들웨어에서 직접 헤더를 사용해야 한다면 해당 헤더의 출처와 유효성을 검사할 필요가 있다. 서버 내부의 재귀 호출인지, 클라이언트 직접 요청인지 판별하거나 메타데이터 적용을 통해 신뢰할 수 있는 헤더인지 검증할 필요가 있다.

또한 미들웨어 로직은 클라이언트 요청의 전처리 수준에서만 동작하므로 실제 인증 및 인가 검증을 서버에서 이중 확인하는 구조로 설계할 필요가 있다. 즉, 보호된 리소스에 접근하기 위해 서버 측에서 반드시 세션 및 토큰을 재검증하도록 구성하는 다단계 보안 모델을 도입할 필요도 있다.

2. 취약점 관련 자료

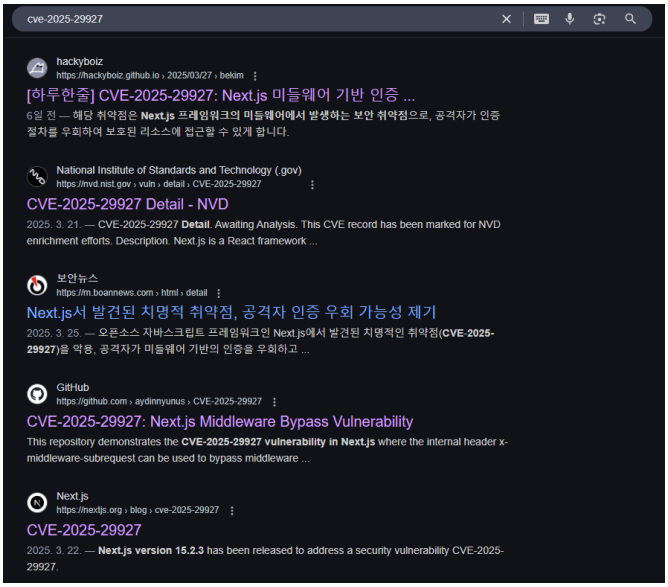
2-1. Next.js 미들웨어 기반 인증 우회 취약점 관련 자료

본 레포트를 작성하면서 참고한 Next.js 미들웨어 기반 인증 우회 취약점 관련 자료는 아래와 같다.

[1] https://nvd.nist.gov/vuln/detail/CVE-2025-29927
[2] https://github.com/aydinnyunus/CVE-2025-29927
[3] https://zhero-web-sec.github.io/research-and-things/nextjs-and-the-corrupt-middleware
[4] https://nextjs.org/docs/app/building-your-application/routing/middleware
[5] https://nextjs.org/blog/cve-2025-29927
[6] http://www.openwall.com/lists/oss-security/2025/03/23/3
[7] http://www.openwall.com/lists/oss-security/2025/03/23/4
[8] https://github.com/vercel/next.js/security/advisories/GHSA-f82v-jwr5-mffw
[9] https://security.netapp.com/advisory/ntap-20250328-0002/

2-2. 인터넷 상에서 찾은 방법

본 레포트를 작성하기 위해 참고한 자료는 인터넷 상에서 먼저 구글링을 통해 자료를 조사하였다. CVE-2025-29927 에 대한 조사를 하기 위해 구글에 CVE-2025-29927 을 검색하였고 검색 결과 아래와 같은 결과 화면이 나타났다.



해당 결과 화면에서 우선적으로 관련 포스팅 사이트를 들어갔다. 왜냐하면 블로그를 통해 해당 취약점에 대해 이해가 우선이라 생각되어 해당 블로그에서 배경 지식을 쌓은 후 Next.js에서 공식적인

글들을 이해하며 깊이 있는 배경지식을 쌓은 후 github에서 코드를 전부는 아니여도 일부를 이해할 수 있었다. 또한 블로그에서 제시한 References를 통해 하나의 블로그에서 여러 개의 참고자료를 찾을 수 있어 이에 대해 모두 하이퍼링크를 통해 조사를 할 수 있었다. 또한 NVD는 기술적 제한 있는 자료지만 해당 사이트에서도 관련 참고 References를 달아 두었기에 관련 참고 자료를 쉽게 찾아 조사하고 정리할 수 있었다. 결과적으로 해당 과제는 취약점과 관련된 기술을 조사할 수 있었고 취약점을 이해하기 위한 방법을 스스로 구축하게 하였으면 참고 자료를 찾을 수 있는 효율적인 방법을 스스로 구축할 수 있게 도움을 준 과제였다.