

# Analyse et Exploitation de la CVE-2025-55182 (React2Shell)

Devoir 59589 - Sécurité des Systèmes d'Information

Étudiants ENSIMAG

18 janvier 2026

## Table des matières

<b>1</b>	<b>Description de la Vulnérabilité (Critère 1)</b>	<b>3</b>
1.1	Présentation . . . . .	3
1.2	Origine et Fonctionnement . . . . .	3
<b>2</b>	<b>Architecture et Analyse (Critère 2)</b>	<b>4</b>
2.1	Architecture Typique Vulnérable . . . . .	4
2.2	Flux d'Exploitation (Détail de la Vulnérabilité) . . . . .	4
2.3	Détails de l'Infrastructure . . . . .	4
<b>3</b>	<b>Analyse de la Cible de Sécurité (Critère 4)</b>	<b>6</b>
3.1	Biens (Assets) à Protéger . . . . .	6
3.2	Menaces (Threats) . . . . .	6
3.3	Fonctions de Sécurité (Security Functions) . . . . .	6
<b>4</b>	<b>Contre-mesures Administrateur (Critère 3)</b>	<b>7</b>
4.1	Actions Correctives (Patching) . . . . .	7
4.2	Limitation de l'Impact (Défense en Profondeur) . . . . .	7
4.3	Surveillance et Filtrage . . . . .	7
<b>5</b>	<b>Expérimentation (Critère 5)</b>	<b>8</b>
5.1	Mise en Oeuvre du Laboratoire . . . . .	8
5.2	Scénario d'Exploitation . . . . .	8
5.2.1	Objectif . . . . .	8
5.2.2	Exécution de l'Attaque . . . . .	8
5.2.3	Résultat . . . . .	8
5.2.4	Validation . . . . .	8
<b>6</b>	<b>Glossaire et Références (Critère 6)</b>	<b>10</b>
6.1	Glossaire . . . . .	10
6.2	Références . . . . .	10

# 1 Description de la Vulnérabilité (Critère 1)

## 1.1 Présentation

La CVE-2025-55182, surnommée "**React2Shell**", est une vulnérabilité critique d'exécution de code à distance (RCE) affectant les versions récentes de React (19.x) et Next.js (15.x/16.x) utilisant l'architecture "App Router". Elle a obtenu un score CVSS de **10.0** (Critique).

## 1.2 Origine et Fonctionnement

La faille réside dans le mécanisme de désérialisation du protocole **React Server Components (RSC) Flight**.

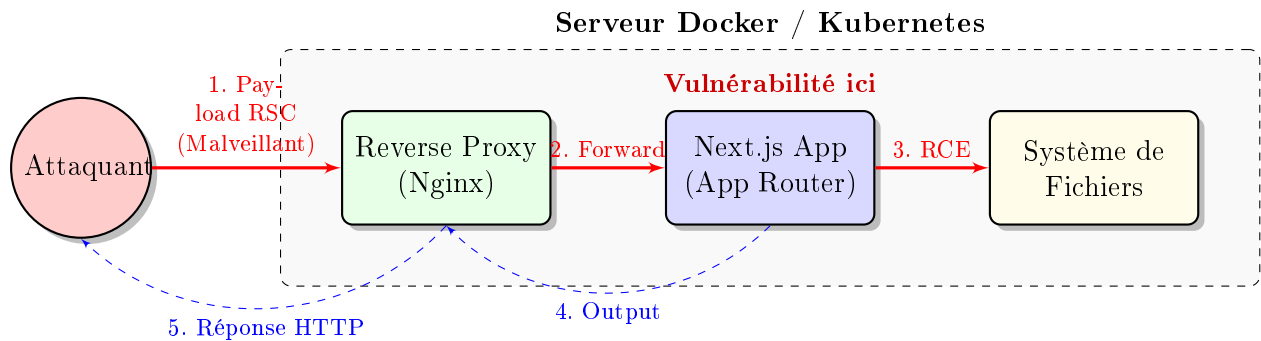
- **Contexte** : Next.js permet aux clients d'envoyer des objets sérialisés au serveur (pour les Server Actions par exemple).
- **Défaut** : Le désérialiseur serveur ne vérifie pas correctement les types des objets reçus, permettant une "Pollution de Prototype" ou l'injection d'objets malveillants lors de la reconstruction de l'arbre de composants.
- **Exploitation** : Un attaquant peut envoyer un payload JSON spécialement conçu qui, une fois désérialisé, incite le serveur à :
  1. Charger des modules arbitraires (via 'module.require').
  2. Exécuter des commandes système (via 'child\_process.execSync').

Ce processus se déroule **avant** toute authentification applicative, rendant tout serveur exposé immédiatement vulnérable.

## 2 Architecture et Analyse (Critère 2)

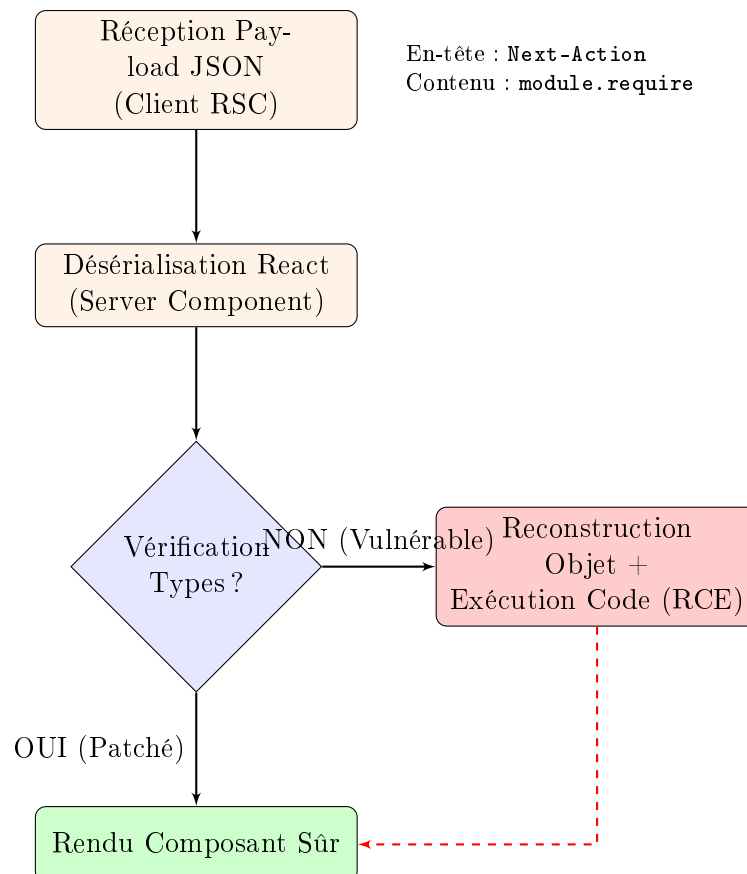
### 2.1 Architecture Typique Vulnérable

L'architecture cible est une application web moderne basée sur Next.js, souvent déployée via Docker ou Kubernetes. Le schéma ci-dessous illustre l'infrastructure réseau et le flux de l'attaque.



### 2.2 Flux d'Exploitation (Détail de la Vulnérabilité)

Le schéma suivant détaille le processus interne de la désérialisation défaillante.



### 2.3 Détails de l'Infrastructure

- **Machines concernées** : Le serveur hébergeant l'application Node.js.
- **Services associés** :

- **Next.js Server** : Gère le rendu (SSR) et les requêtes API. C'est ici que la désérialisation a lieu.
- **Reverse Proxy (Optionnel)** : Peut ne pas filtrer les requêtes RSC car elles ressemblent à du trafic légitime.
- **Réseau** : L'application est généralement exposée sur le port 3000 (interne) ou 80/443 (externe).

### 3 Analyse de la Cible de Sécurité (Critère 4)

Une analyse formelle selon la définition de Cible de Sécurité (Security Target).

#### 3.1 Biens (Assets) à Protéger

1. **Intégrité du Serveur** : Le système d'exploitation hôte ou le conteneur ne doit pas être modifié par des tiers non autorisés.
2. **Confidentialité des Données** : Fichiers de configuration (`.env`, clés API), bases de données, code source.
3. **Disponibilité du Service** : L'application doit rester accessible aux utilisateurs légitimes (risque de Déni de Service via RCE).

#### 3.2 Menaces (Threats)

- **T.RCE (Remote Code Execution)** : Un attaquant exécute des commandes arbitraires avec les privilèges du processus Node.js.
- **T.EXFIL (Exfiltration)** : Vol de données sensibles (clés AWS, mots de passe BDD) via lecture de fichiers ou variables d'environnement.
- **T.LATERAL (Mouvement Latéral)** : Utilisation du serveur compromis comme pivot pour attaquer le réseau interne.

#### 3.3 Fonctions de Sécurité (Security Functions)

- **SF.DESERIALIZATION (Déficiente)** : Le mécanisme de désérialisation RSC doit valider strictement les types d'objets entrants. (*C'est la fonction défaillante ici*).
- **SF.ISOLATION** : Utilisation de conteneurs (Docker) et d'utilisateurs non-privilégiés pour limiter l'impact d'une compromission.
- **SF.FILTERING** : WAF pour bloquer les payloads suspects contenant des chaînes comme `'child_process'` ou `'proto.'`.

## 4 Contre-mesures Administrateur (Critère 3)

En tant qu'administrateur système ou DevOps, voici les dispositions à prendre pour limiter ou éviter les conséquences.

### 4.1 Actions Correctives (Patching)

La mesure la plus efficace est la mise à jour des composants affectés.

- **Next.js** : Mettre à jour vers la version **16.0.7** ou supérieure (ou 15.2.4+).
- **React** : Utiliser les versions patchées **19.0.1**, **19.1.2** ou **19.2.1**.

### 4.2 Limitation de l'Impact (Défense en Profondeur)

#### 1. Principe de Moindre Privilège :

- Ne jamais faire tourner le processus Node.js en **root**.
- Créer un utilisateur dédié (ex : 'node') dans le Dockerfile.

#### 2. Système de Fichiers en Lecture Seule :

- Monter le rootfs en **read-only** pour empêcher l'installation de malwares ou backdoors persistantes.

#### 3. Segmentation Réseau :

- Isoler le conteneur applicatif dans un VLAN ou un réseau Docker restreint.
- Bloquer les connexions sortantes (egress filtering) pour empêcher les reverse shells.

### 4.3 Surveillance et Filtrage

- **WAF** : Configurer des règles pour bloquer les requêtes contenant des signatures d'exploitation RSC (ex : en-tête 'Next-Action' suspect couplé à des mots clés JSON malveillants).
- **Logging** : Surveiller les journaux pour détecter des processus enfants inattendus ('sh', 'bash', 'curl') lancés par Node.js.

## 5 Expérimentation (Critère 5)

Nous avons mis en place un laboratoire complet pour reproduire cette faille.

### 5.1 Mise en Oeuvre du Laboratoire

L'environnement utilise Docker Compose pour orchestrer :

- **vulnerable-app** : Un conteneur Next.js 16.0.6 (vulnérable).
- **exploit-client** : Une machine attaquante avec les scripts Python.

```
1 # D marrage via le script fourni
2 ./lab.sh start
3
4 # V rification
5 curl -s http://localhost:3000
```

Listing 1 – Démarrage du Lab

### 5.2 Scénario d'Exploitation

#### 5.2.1 Objectif

Lire le fichier secret `/flag.txt` situé sur le serveur ("Capture the Flag"), puis soumettre le résultat dans l'interface web prévue à cet effet.

#### 5.2.2 Exécution de l'Attaque

Nous utilisons un script Python (`exploit.py`) qui forge une requête HTTP POST multipart.

```
1 python3 exploit/exploit.py http://localhost:3000 "cat /flag.txt"
```

Listing 2 – Commande d'exploitation

#### 5.2.3 Résultat

Le serveur exécute la commande et renvoie le résultat dans la réponse d'erreur RSC.

```
[*] Target: http://localhost:3000
[*] Command: cat /flag.txt
[*] Sending exploit payload...
```

```
[+] VULNERABLE! Command output:
ENSIMAG{R34CT_S3RV3R_COMPON3NTS_RCE}
```

#### 5.2.4 Validation

L'utilisateur copie ensuite ce flag (`'ENSIMAG...'`) dans le formulaire de l'application web pour valider le succès de l'opération.



## Illustration de l'Interface Web

### ⚠️ Labo Vulnérable CVE-2025-55182

#### 🔴 ATTENTION

Cette application est **intentionnellement vulnérable** à la CVE-2025-55182 (React2Shell). Elle est conçue uniquement à des fins de recherche en sécurité et d'éducation.

#### 🎯 Votre Mission

Un fichier secret nommé `flag.txt` a été caché à la racine du système de fichiers du conteneur (`/flag.txt`).

**Objectif :** Utilisez la vulnérabilité RCE pour lire le contenu de ce fichier !

Valider

✅ Bravo ! Vous avez trouvé le bon flag !

## 6 Glossaire et Références (Critère 6)

### 6.1 Glossaire

- **RSC (React Server Components)** : Composants React rendus exclusivement sur le serveur.
- **RCE (Remote Code Execution)** : Capacité d'un attaquant à exécuter du code arbitraire sur une machine distante.
- **SSR (Server-Side Rendering)** : Génération du HTML côté serveur.
- **CVE (Common Vulnerabilities and Exposures)** : Liste publique des failles de sécurité.

### 6.2 Références

1. **NIST NVD**, "CVE-2025-55182 Detail", <https://nvd.nist.gov/vuln/detail/CVE-2025-55182>
2. **Wiz Research**, "React2Shell : Pwning Next.js servers remotely", <https://www.wiz.io/blog/nextjs-cve-2025-55182-react2shell-deep-dive>
3. **React Team**, "Security Advisory : RSC Payload Deserialization", <https://react.dev/security>
4. **Projet GitHub du Lab**, *Code source fourni avec ce rapport.*