

SwampCTF – Hidden Message-Board

Présentation du challenge

Le challenge consiste à analyser une **application web de type forum**, dans laquelle les utilisateurs peuvent **soumettre des messages** visibles publiquement.

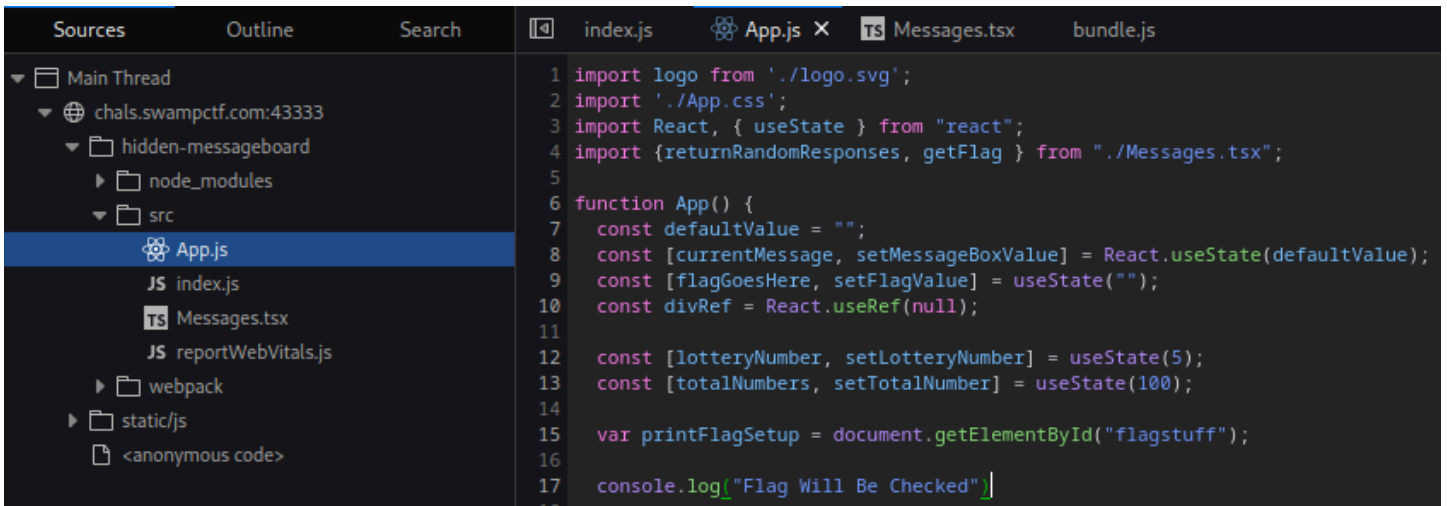
L'objectif est d'**extraire un flag caché** en exploitant une vulnérabilité **XSS (Cross-Site Scripting)**. Pour cela, il faut **injecter du code JavaScript** qui sera exécuté dans le contexte du navigateur de la victime (ici, potentiellement nous-mêmes).

Analyse initiale

Architecture technique

- Le frontend repose sur le framework **React**, utilisé pour générer dynamiquement l'interface utilisateur.
- Le backend s'appuie sur un **serveur Python**, qui gère les messages.

Le code JavaScript côté client, dans le fichier `App.js`, contient une fonction particulièrement révélatrice :



```
1 import logo from './logo.svg';
2 import './App.css';
3 import React, { useState } from "react";
4 import {returnRandomResponses, getFlag } from "../Messages.tsx";
5
6 function App() {
7   const defaultValue = "";
8   const [currentMessage, setMessageBoxValue] = React.useState(defaultValue);
9   const [flagGoesHere, setFlagValue] = useState("");
10  const divRef = React.useRef(null);
11
12  const [lotteryNumber, setLotteryNumber] = useState(5);
13  const [totalNumbers, setTotalNumber] = useState(100);
14
15  var printFlagSetup = document.getElementById("flagstuff");
16
17  console.log("Flag Will Be Checked")
18 }
```

```
var printFlagSetup = document.getElementById("flagstuff");

async function checkCode(){
  if(printFlagSetup != undefined){
    if(printFlagSetup.getAttribute("code") === "G1v3M3Th3Fl@g!!!!"){
      const flag = await getFlag();
      setFlagValue("[flag]: " + flag);
    }
  }
}
```

```
<h1><u>All User Messages</u></h1>
{flagGoesHere}
<div ref={divRef}></div>
```

Ce fragment indique que si l'élément HTML ayant pour ID `flagstuff` possède un attribut `code` égal à la chaîne `"G1v3M3Th3Fl@g!!!!"`, alors une fonction asynchrone est déclenchée pour **recupérer et afficher le flag**.

🎯 Objectif technique

Pour obtenir le flag, il faut :

1. Réussir à **assigner dynamiquement** l'attribut `code="G1v3M3Th3Fl@g!!!!"` à l'élément `#flagstuff`.
 2. Provoquer l'exécution de la fonction `checkCode()` dans un contexte où la condition est remplie.
-

🔧 Approche méthodologique

Afin de déterminer la meilleure manière d'injecter cet attribut, plusieurs essais ont été menés.

L'idée derrière cette série de tests est la suivante :

- Vérifier si un contenu injecté dans les messages est **interprété comme HTML**, ou traité comme du **texte brut**.
 - Observer si la modification directe du DOM via un message est **persistée** ou bien **réinitialisée par le frontend React**.
 - Tester différentes formes de **payloads** pour identifier celles pouvant contourner les éventuels mécanismes de réinitialisation ou de filtrage.
-

🧱 Tentatives exploratoires

✗ Texte brut

```
G1v3M3Th3Fl@g!!!!
```

→ Résultat attendu : rien ne se passe. Confirmé, car c'est simplement affiché sans effet sur le DOM.

✗ Attribut isolé

```
code="G1v3M3Th3Fl@g!!!!"
```

→ Toujours interprété comme du texte, aucun impact sur l'attribut cible.

✗ Injection de balise HTML

```
<div id="flagstuff" code="G1v3M3Th3Fl@g!!!!"></div>
```

→ Le DOM affiche la balise, mais le site semble **réinitialiser dynamiquement l'attribut `code`**, probablement via React. Cette méthode est donc inefficace.

✓ Payload final retenu

Face à ces limitations, une autre piste a été envisagée : **découpler la modification de l'attribut d'une exécution immédiate**. En clair, plutôt que d'injecter directement un élément modifié, on injecte un **lien JavaScript** qui effectue la modification **au clic de l'utilisateur**.

```
<a href="javascript:document.getElementById('flagstuff').setAttribute('code', 'G1v3M3T
```

🧠 Justification du choix

Cette approche repose sur plusieurs constats :

1. Le contenu des messages est rendu via `innerHTML`, donc **le navigateur interprète les balises HTML insérées**.
2. Les scripts dans les balises `<script>` semblent désactivés, mais les **liens JavaScript** sont exécutés au clic.
3. Le framework React réinitialise les attributs dynamiquement à chaque rendu, mais **ne s'oppose pas à une modification déclenchée par une interaction utilisateur**.
4. En injectant une action différée (ici, via un lien cliquable), on **évite toute annulation automatique**, car la modification de l'attribut se fait après le rendu.

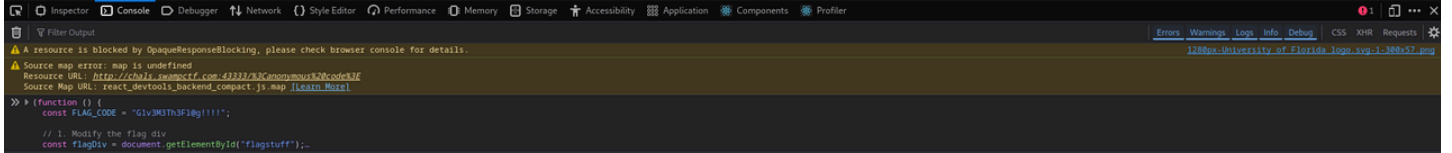
Ce lien, lorsqu'il est cliqué, appelle la méthode `setAttribute()` sur l'élément cible, ce qui attribue la valeur attendue au champ `code`. Dès lors, `checkCode()` reconnaît cette valeur et affiche le flag.

New Message

Post

All User Messages

[flag]: swampCTF{Cr0ss_S1t3_Scr1pt1ng_0r_XSS_c4n_ch4ng3_w3bs1t3s}
[anon]: Yes



Flag obtenu

```
swampCTF{Cr0ss_S1t3_Scr1pt1ng_0r_XSS_c4n_ch4ng3_w3bs1t3s}
```