# Sec

- Syn flood
- Inj SQL / XSS / CSRF

# Syn flood

Une attaque **SYN flood** (attaque semi-ouverte) est un type d’attaque par déni de service (DDoS) qui vise à rendre un serveur indisponible pour le trafic légitime en consommant toutes les ressources serveur disponibles.

Progression of a SYN flood.

---

# Three-way handshake

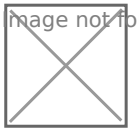
L’attaque **SYN Flood** exploite le principe du **three-way handshake** du protocole **TCP**.

Lors d’une connexion classique entre un **client** et un **serveur** il y a 3 étapes :

Le client envoie un paquet **SYN**.

Le serveur répond ensuite avec un paquet **SYN-ACK** accusant la réception.

Le client envoie un paquet **ACK** et la connexion **TCP** est donc établie.




---

# Déroulement de l’attaque

En envoyant à plusieurs reprises des paquets de demande de connexion initiale (**SYN**), l’attaquant est en mesure de submerger tous les ports disponibles sur une machine serveur ciblée, ce qui oblige l’appareil ciblé à répondre lentement au trafic légitime, ou l’empêche totalement de répondre.

SYN flood — Wikipédia

Les **SYN Flood** sont fréquemment effectuées par des bots se connectant à partir d'**adresses IP usurpées** afin de rendre l'attaque plus difficile à identifier et à atténuer. Les **botnets** peuvent lancer des **SYN Flood** en tant qu'attaques par déni de service distribué (DDoS).

---

Voilà un exemple d'attaque **SYN Flood** & **DNS Flood**:

> Imperva mitigates a 38 day-long SYN flood and DNS flood multi-vector DDoS attack.
> *Imperva mitigates a 38 day-long SYN flood and DNS flood* multi-vector DDoS attack.

Image not found or type unknown

---

# Protéger votre serveur contre le SYN Flood

Dans cet exemple, nous avons **deux machines** en local :

Une machine attaquante : **192.168.1.27**

Une machine victime : **192.168.1.23**

---

Avant de vouloir protéger notre machine, nous allons voir un filtre **wireshark** nous permettant de détecter une attaque **SYN Flood**.

| | |
|---|---|
| 1 | `tcp.flags.syn == 1 and tcp.flags.ack == 0` |

Image not found or type unknown

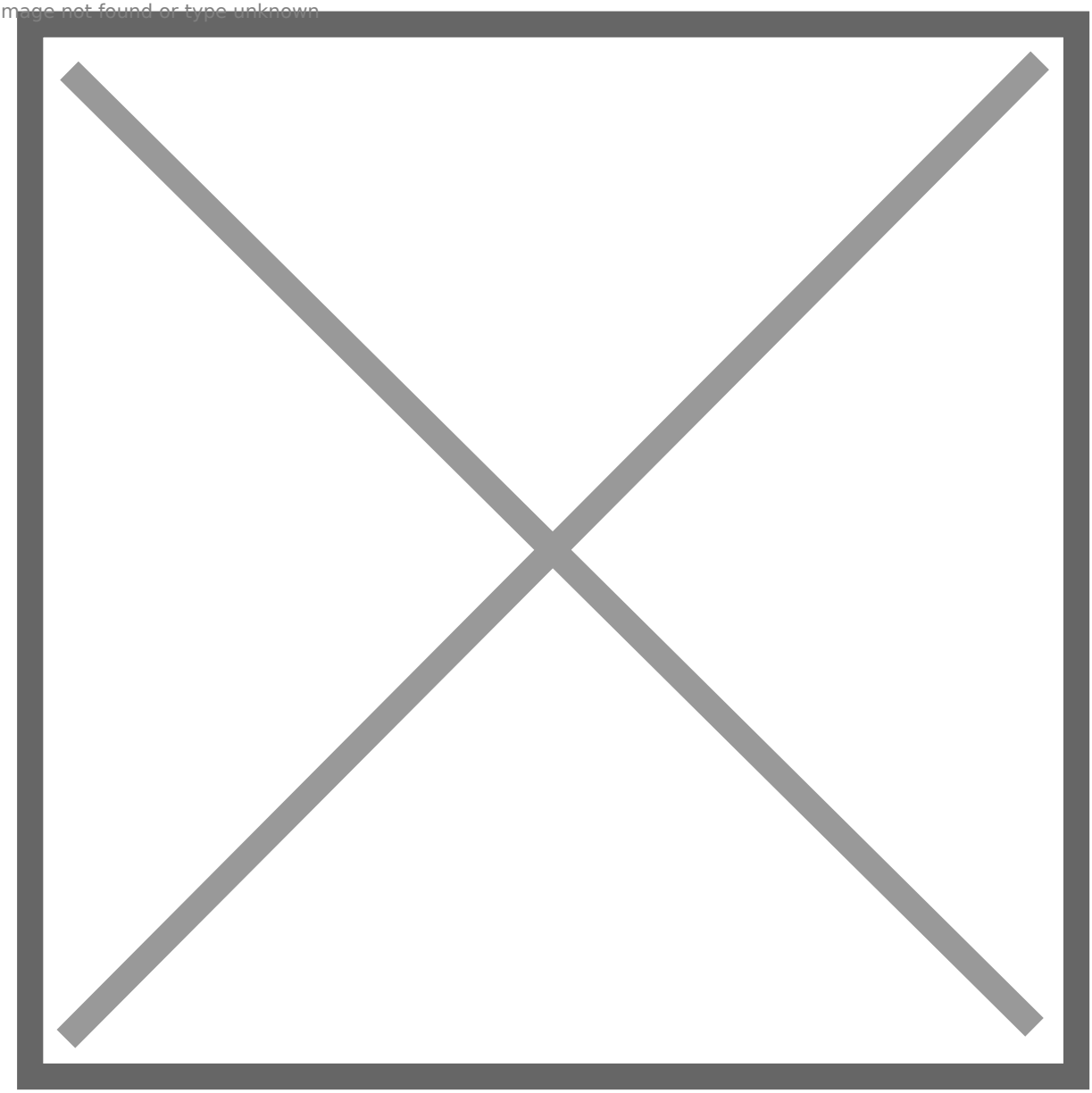
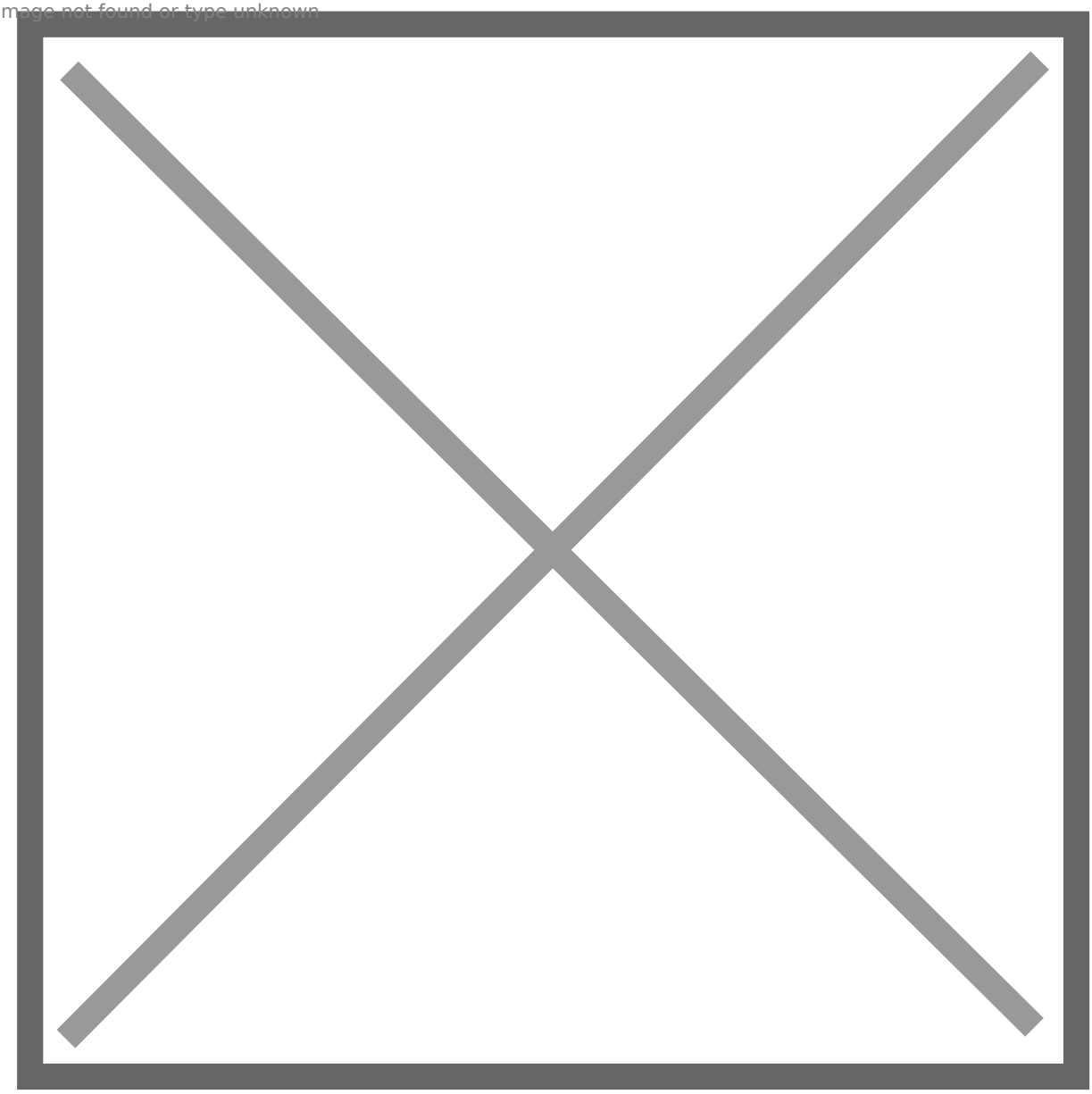


Image not found or type unknown



Comme nous pouvons le voir, nous avons un très grand nombre de **paquets SYN** en destination de notre victime qui est **192.168.1.23** et en provenance d'**adresse IP usurpées**.

---

Nous pouvons aussi détecter une attaque **SYN Flood** à l'aide de la commande suivante qui va nous renvoyer le nombre de connexion dans l'état **SYN_RECV** :

| | |
|---|---|
| 1 | `netstat -npt \| awk '{print $6}' \| sort \| uniq -c \| sort -nr \| head` |

---

Il est temps d'ajouter des paramètres nous permettant de **limiter** ce type d'attaque.

Dans un premier temps, nous allons travailler avec le fichier **/etc/sysctl.conf** puis changer les variables suivantes :

```
net.ipv4.tcp_syncookies = 1
net.ipv4.tcp_max_syn_backlog = 2048
net.ipv4.tcp_synack_retries = 3
net.ipv4.conf.all.rp_filter = 1
```

Pour plus d'informations sur ces dernières je vous recommande cet article.

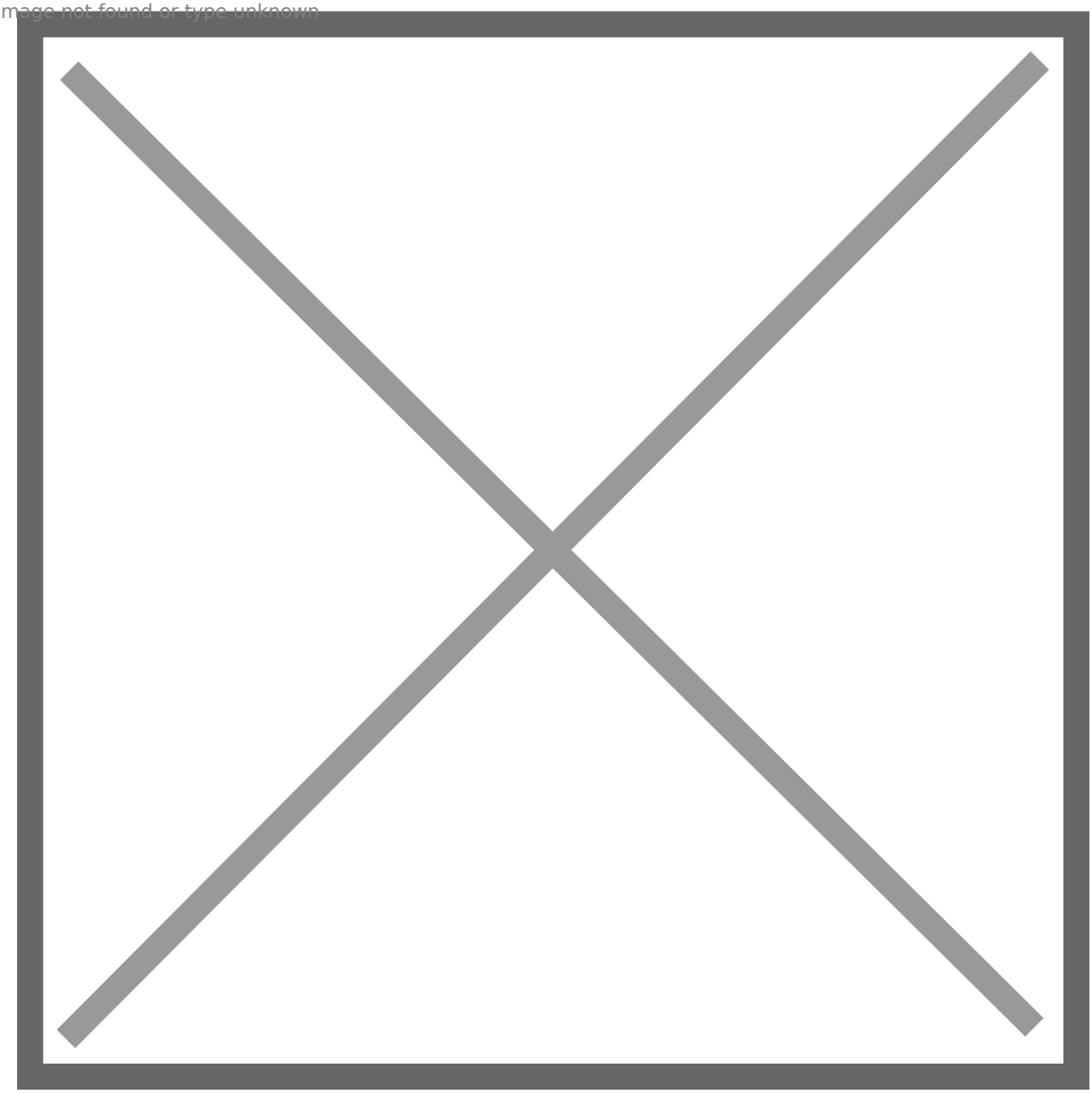
---

Nous pouvons également mettre en place des règles **iptables** permettant de **limiter** ceci comme par exemple :

```
# Création d'une nouvelle chaîne nommée syn_flood
iptables -N syn_flood

# Match les segments TCP pour cette dernière
iptables -A INPUT -p tcp --syn -j syn_flood

# Si la limite match alors on continue à lire les autres règles
iptables -A syn_flood -m limit --limit 1/s --limit-burst 3 -j RETURN

# Si ça ne match pas on drop le paquet
iptables -A syn_flood -j DROP
```

Après avoir mis en place ceci on constate que la chaîne **syn_flood** commence à **DROP** des paquets.

Image not found or type unknown



# Inj SQL / XSS / CSRF

Sécurité web : l'indispensable à savoir pour développeur

- 8 mars 2021
- Indéveloppement, développeur, développeuse, sécurité, technique, web

# Sécurité web : l'indispensable à savoir

Laisser un énorme trou de sécurité est l'un des pires trucs que tu peux faire. Plus que jamais, les développeurs ignorent presque tout du sujet. Ils sont pourtant en première ligne de front face à des hackers doués, réactifs et chirurgicaux.

## Disclaimer

Même avec les protections automatiques des navigateurs et frameworks modernes, des sites sont piratés en permanence. C'est très facile à produire une faille de sécurité.

**Et c'est pas la faute de l'outil quand on sait pas ce qu'on fait.**

Cet article s'adresse à toi ho développeur web des Internets mondiaux. On ne va pas traiter de la partie sécurisation du serveur web. Cette partie concerne plus les SysAdmins et les DevOps (ou DevSecOps si tu préfères). **Aujourd'hui, on parle de la sécurisation de ton application web.**

On veut rendre la tâche extrêmement difficile aux hackers qui s'approchent trop près de toi et de ton site.


Et j’insiste sur l’expression **“rendre la tâche extrêmement difficile”**. Dans le monde de la cybersécurité, tout -ou presque tout- peut être piraté. À condition d’y mettre les moyens.

Ceci dit, à part si tu gères un site gouvernemental, comprendre les principaux vecteurs d’attaque va te protéger contre 99% des menaces qui te concernent.

Malgré le fait que les principales attaques ne changent pas depuis des années, elles sont toujours autant meurtrières. Les statistiques sont déprimantes. Les développeurs font toujours autant les mêmes erreurs.

**Faisons en sorte que tu ne fasses pas partie des statistiques.**

# Injections de code

L’une des mes premières missions lors de mon premier stage en tant que développeur -JADIS- était la construction d’une petite zone d’admin. Une toute petite page en PHP pour manipuler la base de données. Parfait pour commencer en douceur.

Je m’exécute donc. Rapidement j’arrive à un résultat fonctionnel que j’envoie immédiatement à mon supérieur.

Dans cette boite, tout le monde hébergeait son travail sur un serveur web sur sa machine. Avec la bonne IP locale, tu peux accéder au travail local de tout le monde.

Mon supérieur me fait alors venir à son bureau. Il me dit que ça ne marche pas mon truc. Je regarde son écran et là c’est l’horreur. **Non seulement plus aucune mise à jour ne fonctionne, mais en plus apparemment la base de données est vide !**

Quand mon supérieur m’a demandé des explications, je ne savais pas quoi lui dire.


En voyant ma gueule, il s'est immédiatement mis à rigoler comme un phoque. **En fait, il venait de faire une attaque d'injection SQL via le formulaire de mon admin.**

Mais qu'est ce que c'est et comment ça fonctionne ?

# SQL

Si je commence par ça, c'est que malgré le fait que presque tout le monde en a entendu parler, c'est toujours la plus exploitée. C'est ce trou de sécurité que les développeurs laissent le plus ! C'est de loin la plus dangereuse, même aujourd'hui.

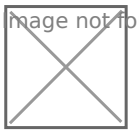
**Et c'est pas moi qui le dis, mais la très sérieuse OWASP.**

Je te montre en PHP, mais on s'en fout du langage, le problème est le même pour tous les langages. Typiquement ce que tu veux surtout pas faire ressemble à ça.

```
<?php
$mysqli = new mysqli("localhost", "username", "username", "dbname");
$sql = "SELECT * FROM users WHERE email='" . $email . "' AND encrypted_password='" .
$password . "'";
$result = $mysqli->query($sql);
$mysqli->close();
?>
```

Ça, c'est une satanerie. **C'est le mal car les variables (ligne 4) sont passées directement dans la requête sans échapper les caractères de contrôle SQL.** Pour mieux comprendre cette phrase, regardons de plus près ce qui se passe lors de l'attaque.

Un utilisateur normal va rentrer son login et son mot de passe dans ton formulaire de cette façon :



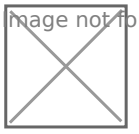
Et ça va générer cette requête SQL.

--
SELECT * FROM users WHERE email='email@email.com' AND encrypted_password='password'
--

Et dans ce scénario, tout va bien. L'utilisateur est authentifié si le login et le mot de passe sont bons. C'est ce qu'on veut.

**Un utilisateur qui te veut du mal va faire les choses différemment.**

Dans le formulaire de login il va tenter de modifier ta requête SQL.
Et sans protection il va facilement y arriver.




Qui va générer cette requête SQL.

--
SELECT * FROM users WHERE email='email@email.com'--' AND encrypted_password='password'
--

Le caractère de contrôle ' ferme la condition du where. Le caractère de contrôle — fait ignorer la suite de la requête. Et comme les caractères de contrôle ne sont pas échappés, le SQL les interprète comme commandes valides !

**L'attaquant vient de se connecter sur ton site -à la place de quelqu'un d'autre- sans avoir besoin de mot passe.**


D'ailleurs, en général, il s'arrête pas là.


--
SELECT * FROM users WHERE email='email@email.com';DROP TABLE users;--' AND encrypted_password='password'

--

Ce qui va gentiment supprimer ta base de données d’utilisateurs.

## Atténuation

Pour éviter cet enfer, il faut juste que tu “échappes” les caractères de contrôle SQL des variables. **Concrètement ça veut dire que SQL va les considérer comme des strings, pas un caractère de contrôle SQL.** Et pour faire ça, il faut utiliser les fonctions intégrées dans ton langage.

```
<?php
$mysqli = new mysqli("localhost", "username", "username", "dbname");
$email = $mysqli->real_escape_string($email);
$password = $mysqli->real_escape_string($password);
$sql = "SELECT * FROM users WHERE email='" . $email . "' AND encrypted_password='" .
$password . "'";
$result = $mysqli->query($sql);
$mysqli->close();
?>
```

Encore une fois ici c’est du PHP, mais tu as des systèmes intégrés d’échappement de caractère dans tous les langages, frameworks possibles !

Cette injection est la plus courante, mais il existe d’autres types d’injection de code.

## Cross-site scripting (XSS)

Dans la nuit de samedi 22 à dimanche 23 juillet 2015, le site jeuxvideo.com n’était pas du tout dans son état normal. **La plupart des pages étaient remplies d’images parodiques, voire pornographiques.** Ce grand site de jeux vidéos français était la victime d’une attaque aussi simple que courante.

À ce moment-là, n’importe qui postant un commentaire sur le site pouvait la déclencher. Autant te dire que personne s’est gêné. Les dev en charge de fermer la faille ont dû passer un mauvais moment.


Pour comprendre ce qui s’est passé, mettons-nous en situation.

## XSS stocké

**On parle d’une attaque XSS.** Et plus précisément, pour le cas de jeuxvideo.com, une attaque XSS stockée.

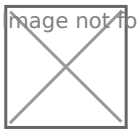
Si tu donnes le pouvoir aux utilisateurs de ton site de poster du contenu, par exemple via une section commentaire, tu t’exposes à cette faille. Comme précédemment, regardons l’attaque en action pour mieux la comprendre.

Prenons l’exemple de Marc. **Marc veut exprimer son amour pour ton site.** Il va rentrer un commentaire dans ton formulaire de commentaire et déclencher un flow d’actions simples et sans danger :


Marc est bienveillant. Il rentre un commentaire normalement. Ce qui va le stocker dans la base de données du site. La page est rafraichie et son commentaire -désormais stocké dans la base de données- va s’afficher pour tout le monde sur la page.

**Encore une fois, un utilisateur qui te veut du mal va faire les choses différemment.**

C’est le cas de Darlene. Elle veut tester si tu t’es protégé ou pas contre le XSS.




Darlene se rend compte que tu n’as mis aucune protection. Elle est très déçue de ton travail.


Comme tu peux le voir, l'attaque est très simple. Comme précédemment, le commentaire est stocké -cette fois avec la balise de script- dans la base de données. Quand la page se rafraichit, la balise script est alors interprétée par le navigateur qui ouvre une pop-up Javascript d'alert pour tout le monde.

**Et si une alert passe, ça veut dire que n'importe quel script Javascript passe.**

Inutile de t'expliquer le potentiel dévastateur ici. Du vol de session à la redirection vers de faux sites pour faire de l'hameçonnage. C'est la porte ouverte à toutes les fenêtres.

Voilà à quoi ressemblait une des attaques faites sur le site jeuxvideos.com lors de cette fameuse soirée.


## XSS réfléchi

Pour l'attaque XSS réfléchie, c'est le même principe. Sauf que le script n'est pas stocké dans la base de données. **Il est passé directement dans l'URL**.

Imagine que ton site permet de faire une recherche. Cette recherche passe une requête HTTP GET et un paramètre d'URL. Dans cette page tu affiches le résultat de la recherche, mais aussi le terme recherché.

Dans un monde idéal, ça se passe de cette façon dans ton site.


Lors d'une attaque XSS réfléchie, l'attaquant va tenter d'injecter du site directement dans la requête de cette façon.

L'attaquant n'a plus qu'à envoyer le lien -avec l'injection de script dans l'url- à quelqu'un. **La personne clique dessus et se fait attaquer via ton site.** En général, l'attaquant va utiliser des sites de minification d'url comme bit.ly pour cacher un petit peu son attaque dans le lien de base.

Et pour te donner une idée de la créativité des hackers autour de cette attaque voici une liste d'attaques XSS recensée par l'OWASP. **Ils ont beaucoup d'imagination.** Il nous faut une solution globale.

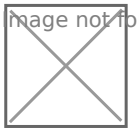
## Atténuation

Pour se protéger de cette attaque, **la principale solution est d'échapper toutes balises HTML qui pourraient venir du client**. Voir carrément supprimer toutes balises quand c'est possible.

Tout ce qui vient du client doit être traité comme du texte, sinon c'est la catastrophe.

Tu peux aussi regarder du côté des Content-Security-Policy pour interdire tout script inline et qui ne vient de ton propre domaine. Mais c'est une mesure en plus. Ce n'est pas en remplacement de la première.

Le plus tu auras de protection, le mieux ça sera.




Il y a un dernier type d'injection XSS via le DOM. J'ai décidé de pas t'en parler ici car elle n'est pas sur le podium. **Par contre, j'ai une recommandation pour toi en fin d'article avec toutes les attaques possibles dedans.**

Avant ça, passons à la troisième attaque la plus commune sur les Internets.

# CSRF

On finit sur la fameuse attaque de falsification de demande intersite ou cross-site request forgery (CSRF).

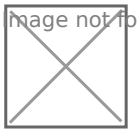
**Cette fois le principe n’est pas d’injecter du code mais de faire des requêtes HTTPs à la place de la victime**. En passant par un site tiers, l’attaquant peut faire faire des actions sur ton site à toi. Et la victime en question n’est même pas au courant que ça se fait !

Dernier petit dessin pour expliquer tout ça.


- **1 :** Un internaute visite un site tenu par un hacker. Sur ce site, des requêtes HTTP faites par l’attaquant sont en place. Ça peut être tout verb HTTP. Les plus courantes avec cette attaque sont le GET, POST et PUT.

- **2** : Ces requêtes sont alors lancées sur ton site, la cible de l’attaque. Involontairement, c’est donc le visiteur -via le site du hacker- qui fait des requêtes chez toi.
- **3 :** Les requêtes arrivent alors sur ton site, et sans protection, elles provoqueront des actions. La gravité de ces actions va être selon ce qu’il est possible de faire sur ton site avec des requêtes HTTP.

Tu peux faire confiance au hacker en question pour trouver des choses intéressantes à faire. Ce sont des pros pour taper là ou ça fait mal. Ils le font avec beaucoup de discrétion.




## Atténuation

La première chose à faire est de suivre les principes REST. **Un GET ne devrait en aucun cas provoquer des changements de status sur ton site.** C’est une satanerie de faire ça. Donc en arrêtant les sataneries, tu vas éviter toute une partie des attaques CSRF.

Pour les autres verbes HTTP, ça demande un peu plus de travail.

D'abord, il faut que tu implémentes des cookies anti-CSRF. Tu as sûrement déjà vu ce genre de choses en inspectant le code de formulaires sur des sites. Ça ressemble à ça.

```
<form action="/update_profile" method="POST">
<input type="text" name="name" />
<input type="submit" value="Submit" />
<!-- Anti-CSRF -->
<input type="hidden" name="token_csrf" value="48rtyu9962dd4s3assa" />
</form>
```

Le token CSRF que tu vois est généré de façon random par le serveur web et intégré au formulaire au moment de la création de ce dernier. **Quand le formulaire est soumis, si le serveur web ne reconnait pas le token, la requête est refusée.** C'est efficace pour bloquer la plupart des attaques. Mais c'est pas assez. Il faut ajouter une dernière protection.

Il faut faire en sorte que toutes tes requêtes aient bien l'attribut SameSite: Strict ou SameSite: Lax dans le Set-Cookie. La plupart des navigateurs récents sont en Lax de base, mais pas tous. Et surtout les navigateurs plus vieux ne sont pas protégés du tout. Cette configuration sur les cookies va te permettre de restreindre quel domaine peut faire des requêtes sur ton domaine.

- En Lax, seules les requêtes GET d'autres domaines peuvent faire des requêtes chez toi. Et c'est pas un problème si tu respectes les principes REST comme vu précédemment.
- En Strict, aucune requête qui ne vient pas de ton propre domaine ne peut faire des requêtes chez toi.

# Aller plus loin

On voit beaucoup de choses aujourd'hui, l'article est déjà très long, donc je m'arrête là. T'imagines bien que les menaces sur Internet sont TRÈS LOIN de s'arrêter là. Et si la sécurité est un sujet important pour toi et ton client, il faut que tu sois préparé à tout.

C'est là qu'intervient ma recommandation du jour : **Web Security for Developers: Real Threats, Practical Defense**

Dès l'introduction, l'auteur te met dans le bain en t'invitant à attaquer ton propre site. Il t'explique comment faire avec "un test de pénétration". C'est incroyablement simple et rapide à faire.

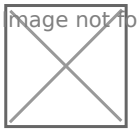
**J'ai été surpris de voir que le blog que tu lis en ce moment comportait une petite faille que je me suis empressé de fermer dans la minute.**

Ensuite la première partie t'explique comment Internet, les requêtes, les headers, les cookies, les paquets, le navigateur et les serveurs webs fonctionnent. Indispensable pour pas être perdu si tout ça est nouveau. Un nécessaire rafraichissement de la mémoire pour les plus habitués.

La seconde partie rentre dans le gras. Dans cet article, je te parle des trois plus grosses attaques sur internet. C'est vraiment juste l'indispensable dont je te parle.

**Ce bouquin t'explique en détail douze autres attaques possibles contre ton site.**

Je savais qu'il y avait beaucoup de danger. Mais pas à ce point-là.




Il y a même une partie plus organisationnelle -presque psychologique- qui t'explique comment **tu peux toi-même être un outil pour certains hackers**. Une des parties les plus intéressantes de ce bouquin à mon sens.

Il s'adresse à tous les développeurs web, peu importe leur niveau. Il part de la base et va progressivement tout t'expliquer. **Évidemment, il va être plus utile aux développeurs avec un besoin de sécurité dans leurs applications.** Mais est-ce que toutes les applications n'ont pas un besoin de sécurité ?

À toi de répondre à cette question.

# Épilogue

Tu ne pourras jamais te protéger contre 100% des attaques. Les hackers sont bien trop talentueux. Mais leur rendre la vie difficile va limiter grandement le danger. Plus ça sera long et complexe de compromettre ton site, moins tu auras de soucis à te faire.